AD-A273 644

DTIC
ELECTE
DEC 13 1993
S A

REUSE-DRIVEN SOFTWARE PROCESSES GUIDEBOOK

SPC-92019-CMC

VERSION 02.00.03

NOVEMBER 1993

93-29999

This document has been approved for public release and sale; its distribution is unlimited.

93 12 8 053

Best
Available
Copy

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Technical Report

4. TITLE AND SUBTITLE
Reuse-Driven Software Processes Guidebook

5. FUNDING NUMBERS
G MDA972-92-J-1018

6. AUTHOR(S)
Produced by Software Productivity Consortium under contract to Virginia Center of Excellence.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Virginia Center of Excellence
SPC Building
2214 Rock Hill Road
Herndon, VA 22070

8. PERFORMING ORGANIZATION REPORT NUMBER
SPC-92019-CMC,
Version 02.00.03

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ARPA/SISTO
Suite 400
801 N. Randolph Street
Arlington, VA 22203

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Supersedes the Domain Engineering Guidebook (DTIC # ADA 259404). Can be used in conjunction with the Reuse Adoption Guidebook (DTIC # ADA 259405)

12a. DISTRIBUTION / AVAILABILITY STATEMENT
No Restrictions

12b. DISTRIBUTION CODE
1

13. ABSTRACT (*Maximum 200 words*)

The Reuse-Driven Software Processes Guidebook describes the Synthesis family of reuse-driven software development processes. In particular, it prescribes the practice for two example members of this family, illustrating the different stages of reuse capability that can be addressed by this family. Development organizations choose the process that matches their reuse goals.

Synthesis is a comprehensive, business-area-level solution to problems of software productivity and quality in the building of systems to meet diverse and changing needs. Synthesis is most appropriate for development organizations that produce many similar systems or single unique systems with highly-volatile requirements during system development and maintenance. Synthesis is a systematic approach to software development founded on the belief that the resources of a business-area organization should be managed not only to meet the immediate needs of customers, but also as an investment in future capability. This guidebook serves as a detailed guide to the practice of Synthesis, addressing all activities and work products related to the production of software, in support of the needs and objectives of a business-area organization and its customers.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software reuse, application engineering, domain engineering, reuse-driven development, reuse-oriented development, iterative process, process family	397
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

REUSE-DRIVEN SOFTWARE PROCESSES GUIDEBOOK

SPC-92019-CMC

VERSION 02.00.03

NOVEMBER 1993

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

DTIC QUALITY INSPECTED 3

Copyright © 1992, 1993, Software Productivity Consortium Services Corporation, Herndon, Virginia. Permission to use, copy, modify, and distribute this material for any purpose and without fee is hereby granted consistent with 48 CFR 227 and 252, and provided that the above copyright notice appears in all copies and that both this copyright notice and this permission notice appear in supporting documentation. This material is based in part upon work sponsored by the Advanced Research Projects Agency under Grant #MDA972-92-J-1018. The content does not necessarily reflect the position or the policy of the U. S. Government, and no official endorsement should be inferred. The name Software Productivity Consortium shall not be used in advertising or publicity pertaining to this material or otherwise without the prior written permission of Software Productivity Consortium, Inc. SOFTWARE PRODUCTIVITY CONSORTIUM, INC. AND SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION MAKE NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THIS MATERIAL FOR ANY PURPOSE OR ABOUT ANY OTHER MATTER, AND THIS MATERIAL IS PROVIDED WITHOUT EXPRESS OR IMPLIED WARRANTY OF ANY KIND.

ADARTS℠ is a service mark of the Software Productivity Consortium, Inc.

Interleaf is a trademark of Interleaf, Inc.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

WordPerfect is a registered trademark of WordPerfect Corporation.

CONTENTS

ACKNOWLEDGMENTS ..... **xix**

PART SYN: OVERVIEW

OV.1. INTRODUCTION **SYN-1**

1.1 Document Purpose, Scope, and Audience Syn-1

1.2 Document Status and Evolution Syn-2

1.3 Related Publications Syn-2

1.4 Document Structure Syn-3

1.5 Using This Guidebook Syn-4

1.6 Conventions and Notation Syn-7

OV.2. FUNDAMENTALS OF SYNTHESIS **SYN-11**

2.1 Key Principles Syn-11

2.1.1 Program Families Syn-11

2.1.2 Iterative Processes Syn-11

2.1.3 Specifications Syn-12

2.1.4 Abstraction-Based Reuse Syn-12

2.2 Context for Synthesis Syn-12

2.2.1 Business Objectives Syn-13

2.2.2 System Engineering Practices Syn-13

2.2.3 Objectives of a Software Engineering Process Syn-13

2.3 An Overview of Synthesis Syn-14

2.3.1 Defining a Tailored Process Syn-16

2.3.2 An Opportunistic Synthesis Process Syn-17

2.3.3 An Integrated Synthesis Process Syn-18

2.3.4 A Leveraged Synthesis Process Syn-20

2.3.5 An Anticipating Synthesis Process Syn-22

PART OPP: OPPORTUNISTIC SYNTHESIS

OV. OVERVIEW OF AN OPPORTUNISTIC SYNTHESIS PROCESS Opp-1

1. Underlying Assumptions Opp-1

2. Software Development With an Opportunistic Synthesis Process Opp-3

2.1. Organizational Perspective Opp-3

2.2. Application Engineering Perspective Opp-4

2.3. Domain Engineering Perspective Opp-4

2.4. An Example Scenario Opp-6

DE. DOMAIN ENGINEERING OVERVIEW Opp-9

1. Getting Started Opp-9

2. Product Description Opp-10

3. Process Description Opp-10

4. Interactions With Other Activities Opp-13

DE.1. DOMAIN MANAGEMENT ACTIVITY Opp-15

1. Getting Started Opp-15

2. Product Description Opp-16

3. Process Description Opp-17

4. Interactions With Other Activities Opp-22

DE.2. DOMAIN ANALYSIS ACTIVITY Opp-25

1. Getting Started Opp-25

2. Product Description Opp-26

3. Process Description Opp-26

4. Interactions With Other Activities Opp-29

DE.2.1. DOMAIN DEFINITION ACTIVITY Opp-31

1. Getting Started Opp-31
2. Product Description Opp-31
3. Process Description Opp-38
4. Interactions With Other Activities Opp-43

DE.2.2. DOMAIN SPECIFICATION ACTIVITY Opp-45

1. Getting Started Opp-45
2. Product Description Opp-46
3. Process Description Opp-46
4. Interactions With Other Activities Opp-50

DE.2.2.1. DECISION MODEL ACTIVITY Opp-53

1. Getting Started Opp-53
2. Product Description Opp-53
3. Process Description Opp-56
4. Interactions With Other Activities Opp-59

DE.2.2.2. PRODUCT REQUIREMENTS ACTIVITY Opp-61

1. Getting Started Opp-61
2. Product Description Opp-61
3. Process Description Opp-64
4. Interactions With Other Activities Opp-67

DE.2.2.3. PROCESS REQUIREMENTS ACTIVITY Opp-69

1. Getting Started Opp-69
2. Product Description Opp-69
3. Process Description Opp-70
4. Interactions With Other Activities Opp-72

DE.2.2.4. PRODUCT DESIGN ACTIVITY Opp-75

1. Getting Started Opp-75
2. Product Description Opp-75

3. Process Description Opp-76

4. Interactions With Other Activities Opp-78

DE.2.2.4.1. PRODUCT ARCHITECTURE ACTIVITY Opp-81

1. Getting Started Opp-81

2. Product Description Opp-82

3. Process Description Opp-84

4. Interactions With Other Activities Opp-86

DE.2.2.4.2. COMPONENT DESIGN ACTIVITY Opp-89

1. Getting Started Opp-89

2. Product Description Opp-89

3. Process Description Opp-91

4. Interactions With Other Activities Opp-93

DE.2.2.4.3. GENERATION DESIGN ACTIVITY Opp-95

1. Getting Started Opp-95

2. Product Description Opp-96

3. Process Description Opp-98

4. Interactions With Other Activities Opp-100

DE.2.3. DOMAIN VERIFICATION ACTIVITY Opp-103

1. Getting Started Opp-103

2. Product Description Opp-104

3. Process Description Opp-104

4. Interactions With Other Activities Opp-108

DE.3. DOMAIN IMPLEMENTATION ACTIVITY Opp-109

1. Getting Started Opp-109

2. Product Description Opp-110

3. Process Description Opp-110

4. Interactions With Other Activities Opp-111

DE.3.1. PRODUCT IMPLEMENTATION ACTIVITY Opp-113

1. Getting Started Opp-113
2. Product Description Opp-114
3. Process Description Opp-114
4. Interactions With Other Activities Opp-117

DE.3.1.1. COMPONENT IMPLEMENTATION ACTIVITY Opp-119

1. Getting Started Opp-119
2. Product Description Opp-120
3. Process Description Opp-122
4. Interactions With Other Activities Opp-127

DE.3.1.2. GENERATION IMPLEMENTATION ACTIVITY Opp-129

1. Getting Started Opp-129
2. Product Description Opp-129
3. Process Description Opp-130
4. Interactions With Other Activities Opp-132

DE.3.2. PROCESS SUPPORT DEVELOPMENT ACTIVITY Opp-135

1. Getting Started Opp-135
2. Product Description Opp-136
3. Process Description Opp-137
4. Interactions With Other Activities Opp-141

DE.4. PROJECT SUPPORT ACTIVITY Opp-143

1. Getting Started Opp-143
2. Product Description Opp-144
3. Process Description Opp-144
4. Interactions With Other Activities Opp-147

AE. APPLICATION ENGINEERING OVERVIEW Opp-149

1. Getting Started Opp-149

2. Product Description Opp-150

3. Process Description Opp-150

4. Interactions With Other Activities Opp-156

PART LEV: LEVERAGED SYNTHESIS

OV. OVERVIEW OF A LEVERAGED SYNTHESIS PROCESS Lev-1

1. Underlying Assumptions Lev-1

2. Software Development With a Leveraged Synthesis Process Lev-2

2.1. Organizational Perspective Lev-3

2.2. Application Engineering Perspective Lev-3

2.3. Domain Engineering Perspective Lev-3

2.4. An Example Scenario Lev-5

DE. DOMAIN ENGINEERING OVERVIEW Lev-7

1. Getting Started Lev-7

2. Product Description Lev-8

3. Process Description Lev-8

4. Interactions With Other Activities Lev-11

DE.1. DOMAIN MANAGEMENT ACTIVITY Lev-13

1. Getting Started Lev-13

2. Product Description Lev-14

3. Process Description Lev-16

4. Interactions With Other Activities Lev-22

DE.2. DOMAIN ANALYSIS ACTIVITY Lev-25

1. Getting Started Lev-25

2. Product Description Lev-26

3. Process Description Lev-26

4. Interactions With Other Activities Lev-29

DE.2.1. DOMAIN DEFINITION ACTIVITY Lev-31

1. Getting Started Lev-31
2. Product Description Lev-32
3. Process Description Lev-39
4. Interactions With Other Activities Lev-45

DE.2.2. DOMAIN SPECIFICATION ACTIVITY Lev-47

1. Getting Started Lev-47
2. Product Description Lev-48
3. Process Description Lev-48
4. Interactions With Other Activities Lev-51

DE.2.2.1. DECISION MODEL ACTIVITY Lev-55

1. Getting Started Lev-55
2. Product Description Lev-55
3. Process Description Lev-57
4. Interactions With Other Activities Lev-61

DE.2.2.2. PRODUCT REQUIREMENTS ACTIVITY Lev-63

1. Getting Started Lev-63
2. Product Description Lev-64
3. Process Description Lev-66
4. Interactions With Other Activities Lev-70

DE.2.2.3. PROCESS REQUIREMENTS ACTIVITY Lev-73

1. Getting Started Lev-73
2. Product Description Lev-74
3. Process Description Lev-75
4. Interactions With Other Activities Lev-78

DE.2.2.4. PRODUCT DESIGN ACTIVITY Lev-81

1. Getting Started Lev-81
2. Product Description Lev-81

3. Process Description Lev-82

4. Interactions With Other Activities Lev-84

DE.2.2.4.1. PRODUCT ARCHITECTURE ACTIVITY Lev-87

1. Getting Started Lev-87

2. Product Description Lev-88

3. Process Description Lev-90

4. Interactions With Other Activities Lev-93

DE.2.2.4.2. COMPONENT DESIGN ACTIVITY Lev-97

1. Getting Started Lev-97

2. Product Description Lev-97

3. Process Description Lev-99

4. Interactions With Other Activities Lev-101

DE.2.2.4.3. GENERATION DESIGN ACTIVITY Lev-103

1. Getting Started Lev-103

2. Product Description Lev-104

3. Process Description Lev-106

4. Interactions With Other Activities Lev-108

DE.2.3. DOMAIN VERIFICATION ACTIVITY Lev-111

1. Getting Started Lev-111

2. Product Description Lev-112

3. Process Description Lev-112

4. Interactions With Other Activities Lev-116

DE.3. DOMAIN IMPLEMENTATION ACTIVITY Lev-117

1. Getting Started Lev-117

2. Product Description Lev-117

3. Process Description Lev-118

4. Interactions With Other Activities Lev-119

DE.3.1. PRODUCT IMPLEMENTATION ACTIVITY Lev-121

1. Getting Started Lev-121
2. Product Description Lev-122
3. Process Description Lev-122
4. Interactions With Other Activities Lev-124

DE.3.1.1. COMPONENT IMPLEMENTATION ACTIVITY Lev-127

1. Getting Started Lev-127
2. Product Description Lev-128
3. Process Description Lev-130
4. Interactions With Other Activities Lev-135

DE.3.1.2. GENERATION IMPLEMENTATION ACTIVITY Lev-137

1. Getting Started Lev-137
2. Product Description Lev-137
3. Process Description Lev-138
4. Interactions With Other Activities Lev-140

DE.3.2. PROCESS SUPPORT DEVELOPMENT ACTIVITY Lev-143

1. Getting Started Lev-143
2. Product Description Lev-144
3. Process Description Lev-147
4. Interactions With Other Activities Lev-153

DE.4. PROJECT SUPPORT ACTIVITY Lev-155

1. Getting Started Lev-155
2. Product Description Lev-156
3. Process Description Lev-156
4. Interactions With Other Activities Lev-159

AE. APPLICATION ENGINEERING OVERVIEW Lev-161

1. Getting Started Lev-161

2. Product Description Lev-162

3. Process Description Lev-162

4. Interactions With Other Activities Lev-166

APPENDIX: MATURITY ASSESSMENT AND FUTURE EVOLUTION App-1

LIST OF ABBREVIATIONS AND ACRONYMS Abb-1

GLOSSARY Glo-1

REFERENCES Ref-1

FIGURES

Figure OV.2-1.	A Synthesis Process	Syn-15
Figure OV.2-2.	A Simple Example of an Application Engineering Process for Opportunistic Reuse	Syn-19
Figure OV.2-3.	A Domain Engineering Process for Opportunistic Reuse (Simplified)	Syn-20
Figure OV.2-4.	A Prototypical Application Engineering Process for Leveraged Reuse	Syn-22
Figure OV.2-5.	A Domain Engineering Process for Leveraged Reuse	Syn-23
Figure OV-1.	Example Application Engineering and Domain Engineering Interaction	Opp-5
Figure OV-2.	Blowup of a Work Product Family Development Activity Group	Opp-6
Figure OV-3.	Relationship Between Work Product Family Development and Domain Engineering	Opp-7
Figure DE-1.	Domain Engineering	Opp-11
Figure DE.1-1.	Domain Management Process	Opp-18
Figure DE.1-2.	A Risk-Based Process for Increment Management	Opp-20
Figure DE.2-1.	Domain Analysis Process	Opp-27
Figure DE.2.1-1.	Domain Definition Process	Opp-38
Figure DE.2.2-1.	Domain Specification Process	Opp-47
Figure DE.2.2.1-1.	Decision Model Process	Opp-56
Figure DE.2.2.2-1.	Product Requirements Process	Opp-64
Figure DE.2.2.3-1.	Process Requirements Process	Opp-71
Figure DE.2.2.4-1.	Product Design Process	Opp-77
Figure DE.2.2.4.1-1.	Product Architecture Process	Opp-84

Figure DE.2.2.4.2-1.	Component Design Process	Opp-92
Figure DE.2.2.4.3-1.	Generation Design Process	Opp-98
Figure DE.2.3-1.	Domain Verification Process	Opp-104
Figure DE.3-1.	Domain Implementation Process	Opp-111
Figure DE.3.1-1.	Product Implementation Process	Opp-115
Figure DE.3.1.1-1.	Component Implementation Process	Opp-123
Figure DE.3.1.2-1.	Generation Implementation Process	Opp-131
Figure DE.3.2-1.	Process Support Development Process	Opp-138
Figure DE.4-1.	Project Support Process	Opp-144
Figure AE-1.	A Prototypical Application Engineering Process	Opp-151
Figure OV-1.	Interaction Between Application Engineering and (Simplified) Domain Engineering	Lev-4
Figure DE-1.	Domain Engineering	Lev-9
Figure DE.1-1.	Domain Management Process	Lev-16
Figure DE.1-2.	A Risk-Based Process for Increment Management	Lev-19
Figure DE.2-1.	Domain Analysis Process	Lev-27
Figure DE.2.1-1.	Domain Definition Process	Lev-40
Figure DE.2.2-1.	Domain Specification Process	Lev-49
Figure DE.2.2.1-1.	Decision Model Process	Lev-59
Figure DE.2.2.2-1.	Instantiating Product Requirements	Lev-66
Figure DE.2.2.2-2.	Product Requirements Process	Lev-67
Figure DE.2.2.3-1.	Process Requirements Process	Lev-76
Figure DE.2.2.4-1.	Product Design Process	Lev-83
Figure DE.2.2.4.1-1.	Product Architecture Process	Lev-92
Figure DE.2.2.4.2-1.	Component Design Process	Lev-100
Figure DE.2.2.4.3-1.	Generation Design Process	Lev-106
Figure DE.2.3-1.	Domain Verification Process	Lev-112

Figure DE.3-1.	Domain Implementation Process	Lev-118
Figure DE.3.1-1.	Product Implementation Process	Lev-123
Figure DE.3.1.1-1.	Component Implementation Process	Lev-131
Figure DE.3.1.2-1.	Generation Implementation Process	Lev-139
Figure DE.3.2-1.	Process Support Development Process	Lev-148
Figure DE.4-1.	Project Support Process	Lev-156
Figure AE-1.	A Prototypical Application Engineering Process	Lev-163

TABLES

Table App-1. Maturity Scheme for Part Opp, Opportunistic Synthesis App-2

Table App-2. Maturity Scheme for Part Lev, Leveraged Synthesis App-3

EXAMPLES

Example OV.1-1.	Hierarchy of Domain Engineering Activities	Syn-5
Example OV.1-2.	Activity Description Outline	Syn-6
Example DE.2.1-1.	Fragment of TLC Domain Synopsis	Opp-33
Example DE.2.1-2.	Fragment of TLC Domain Glossary	Opp-34
Example DE.2.1-3.	Fragment of TLC Commonality Assumptions	Opp-36
Example DE.2.1-4.	Fragment of TLC Variability Assumptions	Opp-37
Example DE.2.2.1-1.	Fragment of TLC Decision Model for the System/Segment Work Product Family	Opp-55
Example DE.2.2.2-1.	Fragment of TLC Product Requirements for the System/Segment Specification Work Product Family	Opp-63
Example DE.2.2.4.1-1.	Fragment of TLC Product Architecture for the System/Segment Work Product Family	Opp-83
Example DE.2.2.4.2-1.	Fragment of TLC Component Design for the System/Segment Specification Work Product Family	Opp-91
Example DE.2.2.4.3-1.	Fragment of TLC Generation Design for the System/Segment Specification Work Product Family	Opp-97
Example DE.3.1.1-1.	Fragment of the TLC Component Implementation for the System/Segment Specification Work Product Family	Opp-122
Example DE.2.1-1.	Fragment of TLC Domain Synopsis	Lev-33
Example DE.2.1-2.	Fragment of TLC Domain Glossary	Lev-34
Example DE.2.1-3.	Fragment of TLC Commonality Assumptions	Lev-36
Example DE.2.1-4.	Fragment of TLC Variability Assumptions	Lev-37
Example DE.2.2.1-1.	Fragment of TLC Decision Model	Lev-58
Example DE.2.2.2-1.	Fragment of TLC Product Requirements	Lev-65

Example DE.2.2.4.1-1. Fragment of TLC Product Architecture Lev-89

Example DE.2.2.4.1-2. Fragment of TLC Product Architecture Lev-90

Example DE.2.2.4.1-3 Fragment of TLC Product Architecture Lev-91

Example DE.2.2.4.2-1. Fragment of TLC Component Design Lev-99

Example DE.2.2.4.3-1. Fragment of TLC Generation Design Lev-105

Example DE.2.2.4.3-2. Fragment of TLC Generation Design Lev-106

Example DE.3.1.1-1. Fragment of TLC Component Implementation Lev-130

ACKNOWLEDGMENTS

Rich McCabe was the project manager for this guidebook. Grady Campbell was the principal architect of the Synthesis methodology and this guidebook. Neil Burkhard was the lead author for this release, which is an extension of the 1992 Domain Engineering guidebook. Steve Wartik, Jim O'Connor, Joe Valent, and Jeff Facemire were major contributors in writing and refining this or previous versions.

The Consortium wishes to thank the participants in the Rockwell pilot project for helping to refine the ideas of Synthesis. Significant improvements have also come from interactions with the participants of the Boeing STARS pilot project and attendees of Synthesis seminars.

The work of Roger Williams and Ted Davis on the Reuse Adoption Process and Reuse Capability Model provided key insights on the context for reuse-driven software processes and how to characterize a family of processes. Special thanks to Kirsten Blakemore, Bob Hofkin, and Patricia Remacle for help in creating common definitions for shared Evolutionary Spiral Process and Reuse-Driven Software Processes terminology.

In addition, Kirsten Blakemore, Steve Wartik, and Gary Moore provided insightful reviews and helped to improve many aspects of this release of the guidebook. The Environment and Support Services group of the Software Productivity Consortium provided superb help in producing the document.

This page intentionally left blank.

PART SYN: OVERVIEW

This page intentionally left blank.

OV.1. INTRODUCTION

Synthesis is a methodology for constructing software systems as instances of a family of systems that have similar descriptions (Campbell, Faulk, and Weiss 1990). This guidebook provides an introduction to the practice of the Synthesis methodology of software development. To the degree that you understand the essential similarities and variations in the systems you build, Synthesis enables you to exploit those similarities to eliminate redundant work. A mature organization will be able to satisfy the needs of its customers by answering the questions that are left open because of variations.

Synthesis focuses on your need both to deliver high quality products to customers and to accomplish this profitably. To this end, a Synthesis process consists of two subprocesses: Application Engineering and Domain Engineering. Application Engineering is how a group (or project) in your organization creates a product to meet customer requirements. Domain Engineering is how your organization improves productivity by creating a product family and a supporting Application Engineering process, tailored for projects in your business area. The details of these subprocesses will differ depending on the capabilities of your organization to practice reuse effectively.

Since projects in the same business area tend to build systems that satisfy similar needs, these systems can be thought of as instances of a family. A family of systems is a basis for a flexible approach to standardization that can accommodate diverse and changing needs. A business area whose objectives are fulfilled by a family is a domain. Both the mission of an organization and the changing needs of its customers determine the objectives of that business-area organization (i.e., product line). Synthesis is a comprehensive, business-area-level solution to problems of software productivity, product quality, manageability, and responsiveness in the building of systems to meet diverse and changing needs. Synthesis is a systematic approach to software development founded on the belief that the resources of a business-area organization should be managed not only to meet the immediate needs of customers, but also as an investment in future capability.

This guidebook describes two instances of a Synthesis family of processes, one that is opportunistic in character, the other that is leveraged. The opportunistic process is oriented toward organizations having modest reuse needs and capabilities. The leveraged process is oriented toward organizations that can make a greater commitment to reuse and that have more advanced needs and capabilities.

1.1 DOCUMENT PURPOSE, SCOPE, AND AUDIENCE

This guidebook defines Synthesis, a sound approach for effective family-oriented software development. It serves as a detailed guide to the practice of Synthesis and helps you begin to practice it. As you gain experience in practicing Synthesis, you will be able to refine and modify this guidance to meet the specific needs of your organization more effectively.

The scope of this guidebook includes all activities and work products related to production of software and support of the needs and objectives of a business-area organization and its customers. Synthesis

activity is initiated, via a Reuse Adoption process (Software Productivity Consortium 1992c), with the establishment of organizational business objectives for a domain. In addition, various Synthesis activities require you to standardize management, requirements, design, implementation, and verification and validation practices throughout your organization. Synthesis is an integrating framework for the methods you choose to standardize your practices. Detailed descriptions of particular methods are generally outside the scope of this guidebook. Such descriptions are available to you, often in other Consortium publications that are referenced, where appropriate, throughout this guidebook. Standardization of these activities in terms of purpose and technique, using methods of your choice, is essential to an effective Synthesis practice.

The audience for this guidebook includes business-area managers, project managers, and engineers of all disciplines who work together to accomplish the objectives of a business-area organization. Readers should be knowledgeable and experienced in the standard (or prevailing) software development methods used in their organization, but are assumed to have no experience with Synthesis. Practical use of this release of the guidebook requires the participation of knowledgeable technologists. Pilot projects are a requisite first step in transitioning to production use.

1.2 DOCUMENT STATUS AND EVOLUTION

Release of this guidebook ends the third year of a multiyear effort to develop a guidebook for a family of Synthesis software processes for use by business-area organizations. This guidebook describes the family, in overview, and two representative processes, in detail. The first of these processes is opportunistic in character and assumes a low degree of organizational reuse capability and commitment; the second process is oriented to leveraged reuse and assumes a moderately high degree of reuse capability and commitment. Either process may be adopted as-is or tailored to fit the particular needs and capabilities of your organization.

This guidebook is a revision of the *Synthesis Guidebook* (Software Productivity Consortium 1991a) and its successor, the *Domain Engineering Guidebook* (Software Productivity Consortium 1992a). The current name was chosen because the guidebook covers the entire software life-cycle process for a domain, comprising Application Engineering as well as Domain Engineering. Major extensions in this version include an expanded characterization of the Synthesis process family, better integration with work on the Evolutionary Spiral Process (ESP) Model (Software Productivity Consortium 1992b), additional and improved activity descriptions, and improved examples.

Appendix A describes the levels of maturity through which the guidebook and each of its sections will progress, and the current status of each. The Consortium sponsors pilot projects with industrial and government partners for the exploration and adoption of Synthesis. Such projects are the Consortium's primary vehicle for improving the content and quality of this guidebook.

1.3 RELATED PUBLICATIONS

Publication of this guidebook follows 4 years of work on Synthesis at the Consortium. The Consortium has delivered a continuing series of other work products that describe various aspects of Synthesis. Information about Synthesis is evolving as experience accumulates from pilot projects. Recently delivered work products include:

- *Systematic Reuse: The Competitive Edge* (Software Productivity Consortium 1991b), a video that briefly explains the concepts and rationale for Synthesis with an orientation to the concerns of executive management.

- *Introduction to Synthesis* (Campbell, Faulk, and Weiss 1990), which provides an informal explanation of the Synthesis vision and its foundations.

- *Introducing Systematic Reuse to the Command and Control Systems Division of Rockwell International* (O'Connor and Mansour 1992), which describes the experience of a pilot project, at Rockwell, which adopted and is proceeding to institutionalize use of a Synthesis process in their business organization.

- *Domain Engineering Validation Case Study—Synthesis for the Air Traffic Display/Collision Warning Monitor Domain* (Burkhard 1992), which provides examples illustrating Synthesis practices, nominally for the leveraged Synthesis process in Part Lev of this guidebook but representative in many aspects of any Synthesis process.

Several recent publications are of interest, not only with respect to Synthesis but for reuse in general:

- *Reuse Adoption Guidebook* (Software Productivity Consortium 1992c), which describes a process by which an organization can adopt a reuse process and the factors that guide definition of the appropriate process.

- *Criteria for Comparing Reuse-Oriented Domain Analysis Approaches* (Wartik and Prieto-Díaz 1992), which discusses the factors that differentiate alternative approaches to domain analysis.

- Introducing Megaprogramming at the High School and Undergraduate Levels (Eward and Wartik, to be published in 1994), which describes a project that worked with high schools and universities to develop curricula and materials for introducing software reuse as megaprogramming, in the form of a Synthesis process.

1.4 DOCUMENT STRUCTURE

This guidebook is organized into three parts. Part Syn is an overview of the Synthesis methodology and associated family of processes. It includes:

- An introduction

- A description of the context and principles of a Synthesis practice

- An explanation of how a particular Synthesis process is derived compatible with the organization's reuse capabilities

This introduction defines standard notations and conventions used throughout all parts.

Parts Opp and Lev are each complete guidebooks for a particular Synthesis process. Part Opp describes an opportunistic Synthesis process. Part Lev describes a leveraged Synthesis process. Each of these parts presently consists of four sections with the fourth section, Advanced Topics, reserved for future use.

- Overview (OV)

- Domain Engineering (DE)

- Application Engineering (AE)
- Advanced Topics (AT)

An Overview section provides a brief description of the applicable Synthesis process as a whole.

A Domain Engineering section defines the activities you follow to standardize the process, work products, and practices of Application Engineering for a business-area organization. The activities of Domain Engineering are organized and presented hierarchically (see Eample OV.1-1). This hierarchy reflects a grouping based on the knowledge and expertise needed to perform each activity, rather than the order in which activities may be performed. A description of an aggregate activity is primarily a summarization and roadmap to its subactivities. Each aggregate activity, as well as each of its subactivities, is described in its own section. Each activity is described according to the outline shown in Example OV.1-2.

An Application Engineering section defines a prototypical process, activities, and work products for Application Engineering. A primary objective of Domain Engineering is to refine this definition to satisfy the needs and objectives of supported projects. Each activity of Application Engineering is described according to the outline shown in Example OV.1-2.

An Advanced Topics section is reserved for future use in presenting discussions of issues requiring advanced understanding of Synthesis. The emphasis of such discussions will be on the refinement of a Synthesis process and its guidebook description to meet particular needs.

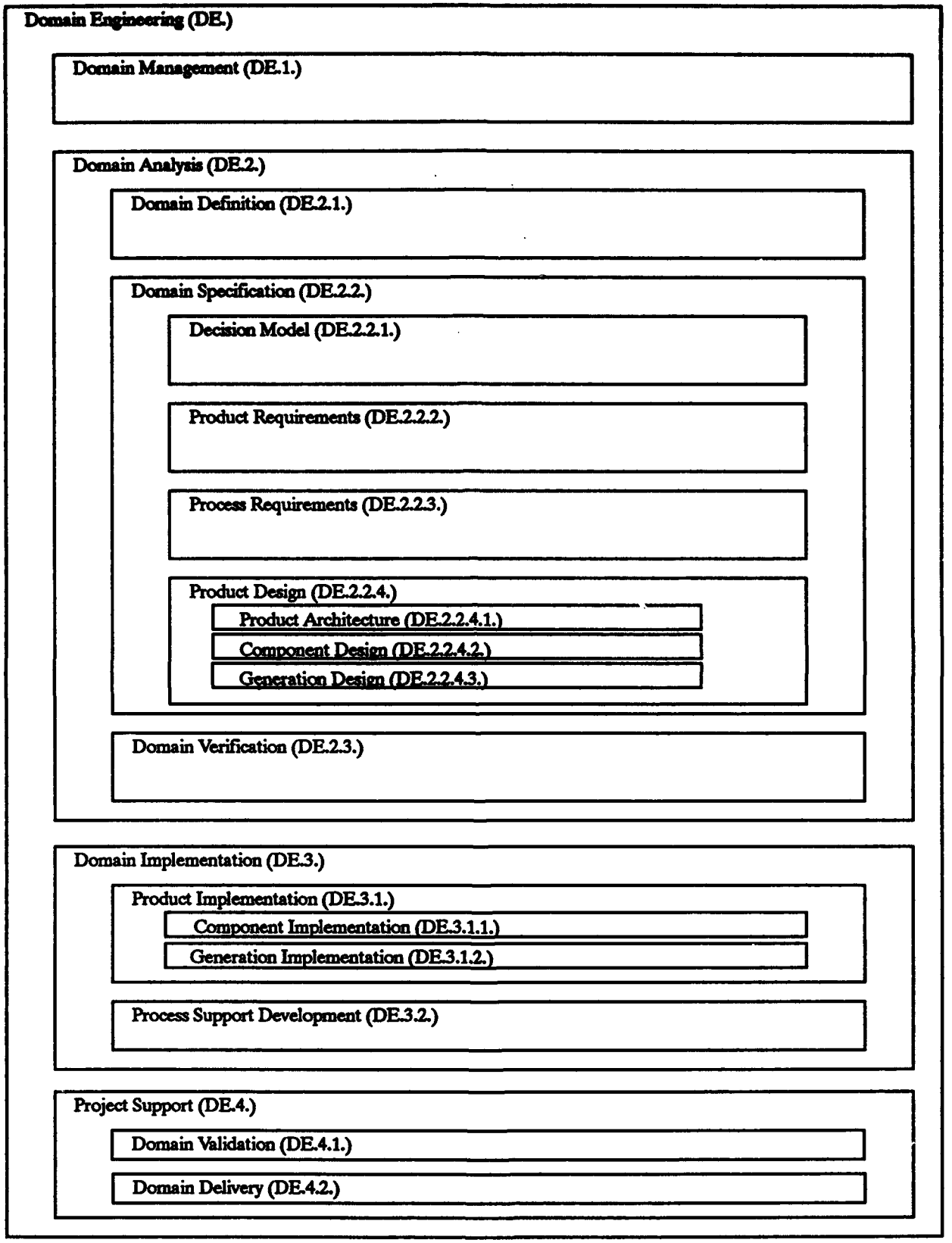
Some supporting material follows Part Lev. The Appendix shows the completeness and maturity of the material in Parts Opp and Lev. A List of Abbreviations and Acronyms, Glossary, and References follow the Appendix.

1.5 USING THIS GUIDEBOOK

This version of the guidebook is intended as a reference for practicing managers and engineers. The guidance can also be tailored to reflect an organization's standard and prevailing software development methods. As a rule, detailed sections are meant to be sufficiently complete for full-scale use. For smaller, more exploratory pilot projects, you may decide to expend less effort on activities related to management, process definition, or opportunities for automation; however, no activities should be skipped entirely.

The guidebook is also organized for use as a graduated introduction to Synthesis. A reading of Section 2 in this Part and Sections DE and AE (in either Part Opp or Part Lev as appropriate) is sufficient to gain a basic understanding of the concepts of Synthesis. Example OV.1-1 serves as a guide to which sections you should read for a deeper understanding of particular aspects of Domain Engineering. You may choose to skip sections lower in the hierarchy when more detail is not of interest. This hierarchy is valid for both Part Opp and Part Lev.

Activity descriptions in the Domain Engineering and Application Engineering sections are not cookbook recipes on how to perform Synthesis. The Synthesis process family is a sophisticated approach to help solve a complex problem in partially understood domains where the experienced engineer must use judgement and intuition to properly interpret Synthesis for his or her given situation. The process diagrams found within this guidebook are intended to aid the reader's understanding of



Example OV.1-1. Hierarchy of Domain Engineering Activities

1. ***Getting Started.*** Describe when the activity is relevant and can be performed.

   - Objectives: Explain the objectives that you should achieve in the performance of this activity.
   - Required information: Describe baselined Synthesis work products or other information upon which some or all of the steps of this activity depend.
   - Required knowledge and experience: Describe business-area, domain, and general software knowledge and experience you need to effectively accomplish the required tasks of this activity.

2. ***Product Description.*** Describe the work product that results from completion of the activity.

   - Purpose: Describe what is accomplished by producing this work product.
   - Content: Describe the information content of this work product.
   - Form and structure: Describe the structure of the work product and the form in which its content is to be presented.
   - Verification criteria: Describe how the consistency/completeness, correctness, and quality of the work product will be judged. Provide review questions and metrics that support that evaluation.

3. ***Process Description.*** Describe a process that achieves the objectives of the activity.

   - Steps: Describe the actions, with associated inputs and results, that are required to accomplish the objectives of the activity. Suggest heuristics for performing each step more effectively.
   - Risk management: Identify risks that may arise to prevent successful, timely completion of the activity, and describe strategies for mitigating those risks. Use checkpoints, reviews, and metrics to reveal flaws and misconceptions.

4. ***Interactions With Other Activities.*** Describe interactions that may occur with other activities as a result of using a work product:

   - Describe feedback to the activities that provide required information to this activity.
   - Describe feedback from other activities concerning the adequacy to them of this activity's work product.

   Describe how those interactions stimulate product evolution. For each potential problem, describe:

   - Contingency: The nature of the problem that may be found in the use of a work product.
   - Source: The activity that provides/uses the work product of concern.
   - Response: The appropriate, alternative responses within this activity to the contingency.

5. ***Advanced Topics.*** (Reserved for future use only.) Provide guidance on complex aspects of this activity in short, topical discussions. More expansive papers appear in the Advanced Topics section. Discusses tailoring to particular needs and the use of alternative, immature, but potentially more robust approaches.

Example OV.1-2. Activity Description Outline

Synthesis. The process diagrams are not meant to be formal and complete process models. Instead, these diagrams depict only those entities and relationships important to the overall spirit of Synthesis. Other aspects of Synthesis have been suppressed where they were considered to be irrelevant to the purpose of the supported text. Hence, a process diagram's purpose is to depict:

- Steps, including subactivities, that make up an activity
- The work product, or component of a work product, produced by each step
- How work products are used within the activity
- Which other activities are the primary users of the work product produced by the activity in the process diagram

Conversely, the process diagrams do not show:

- Control (entities that constrain/enable performance of the activity or its steps)
- Mechanisms (roles that perform the steps of the activity and methods that support that performance)
- Feedback/interaction within or among activities

Iteration within an activity and between other Synthesis activities is determined by the management method. Iteration is only indicated in the top-level process diagram (Figure OV.2-1 in the Fundamentals Section) to suggest evolution of the domain as as whole.

Partial examples of work products are sometimes presented in the text to illustrate the form and content of individual Synthesis work products. These examples contain fragments of Domain Engineering work products drawn from the Traffic Light Control Software System (TLC) domain. A TLC system controls and coordinates the operation of traffic lights at a given intersection.

1.6 CONVENTIONS AND NOTATION

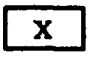
This guidebook uses the following typographic conventions:

Serif font	General presentation of information.
Capitalized Serif font	Names of Synthesis work products and activities.
Italicized serif font	Mathematical expressions and publication titles.
Boldfaced serif font	Section headings and emphasis.
Boldfaced italicized serif font	Run-in headings in bulleted lists and low-level titles in the process sections of guidebooks.
`Typewriter` font	Syntax of code.
{ }	Alternative items (one or more).

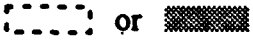
[ ] ..... Optional items (zero or one).

| ..... Separator for a list of alternatives.

In this guidebook, figures that depict a process diagram use the following symbols:

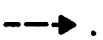
 ..... Activity or step named X.

..... Product or work product named Y.

 ..... A logical grouping of activities or steps.

Italicized serif font ..... An activity outside the context of the particular figure.

..... A relationship between activities or steps and the work product(s) that are inputs to or results of those activities or steps.

 ..... A relationship between activities showing additional interaction or information communicated between those activities.

Work product examples in this guidebook use a metaprogramming notation to represent variability in a product. Variability in a product means that a product will have different content, depending on certain critical decisions. A *metaprogramming notation* allows you to describe how a product's content is determined by those decisions. A simple example of this is the use of a macro processor to defer a decision about the size of a data structure. Instead of making the decision on the size of the structure when the code is written (by embedding a constant), you can defer the decision by setting parameters for (parameterizing) the code and supplying the required value at compile time. A metaprogramming notation is an extension of this idea.

Boldfaced, bracketed text is used for metaprogramming notation in this guidebook:

<boldfaced_identifier> ..... A deferred decision (e.g., **<size>**). Such identifiers may be separated by dots to indicate elements of composite decisions (e.g., **<stack.type>** and **<stack.size>**). This identifier is replaced with the actual value of the decision whenever an instance of a work product is created.

<if predicate **then>** body1 [**<else>** body2] **<endif>** ..... A conditional inclusion. If the predicate evaluates to true, then body1 is included in the work product. If an **else** clause is included and the predicate evaluates to false, then body2 is included in the work product. The predicate is informal and defined in terms of decisions. Also referred to as a conditional term.

<forall ident **in** list**>** body **<endfor>** An iterative (repeating) inclusion. The list is an identifier for a decision that is multivalued. This construct includes

one copy of body in the work product for each value of the decision. For each copy of body included, the corresponding decision value replaces all occurrences of identifier ident in that copy. Also referred to as an iterative term.

A body is any text that may be a part of some work product. A body may also contain nested metaprogramming constructs; if so, those constructs must be evaluated to determine the content of the body.

This page intentionally left blank.

OV.2. FUNDAMENTALS OF SYNTHESIS

Synthesis was conceived in recognition of past experience that suggested a need for a major reconception of the software process (Campbell, Faulk, and Weiss 1990) to produce improvements in software productivity, product quality, manageability, and customer responsiveness. Davis, Bersoff, and Comer (1988) present a comparison of several models of the software development life cycle, which is a good framework for understanding this need. As the foundation for an initial understanding of Synthesis, this section describes the key principles underlying the Synthesis approach, discusses the context in which Synthesis applies, and provides an overview of how Synthesis works, both in general and at the stages of reuse program implementation for which example Synthesis processes have been defined.

2.1 KEY PRINCIPLES

The Synthesis concept depends on four basic principles: program families, iterative processes, specifications, and abstraction-based reuse. An understanding of these principles will help you to understand Synthesis.

2.1.1 Program Families

A program family is a set of programs that are sufficiently similar that it is worthwhile to understand the common properties of the set before considering the special properties of individual instances. The concept of program families was first proposed by Dijkstra (1972) and later elaborated by Parnas (1976). Both papers argue that developers should construct software programs, not as unique artifacts, but as instances of a family of similar programs. A primary distinction of this view is in how the creation of program versions are viewed: not as successive modifications to previous versions, but as rederivations from a common abstraction. Each member of a family can be characterized entirely in terms of how it differs from the common abstraction (i.e., the variations in the family). In Synthesis, this concept of program families is generalized to a concept of product families that encompass all the work products of software development.

2.1.2 Iterative Processes

An iterative process is a process in which work products are considered complete only after repetition of producing and using activities. Each iteration is short with goals set to make progress toward an end product without unnecessary exposure to risk of failure if short-term goals are not met. In a noniterative process, an activity continues without interruption until its resulting work products are believed complete. Only after such proclaimed completion is there any attempt to use those work products in performing other activities. Although planning does not provide for rework of the results, feedback from using activities inevitably requires replanning to allow for corrections; schedule and quality are

usually degraded as a result. Since many errors in software work products are difficult to discover without feedback from detailed use, an iterative process systematically produces better quality results than a process that depends on producing correct work products without such repetition. A strong discipline of work product versioning and configuration management is crucial to a successful iterative process (Humphrey 1989).

2.1.3 Specifications

A specification is a complete, precise description of the verifiable properties required of a work product or set of work products (e.g., a complete software product). For a system, a specification can serve as an abstract model that aids understanding and analysis. For expressive power, the notation in which a specification is written usually assumes an understanding of specialized terminology and constructs. Normally, a specification is either a description of requirements (i.e., the problem to be solved) or a description of a design (i.e., the form and/or content of a solution). Additionally in Synthesis, a specification may describe a product, or work product, (i.e., a problem and its solution) by resolving the variations that characterize a family of such (work) products.

This concept of specifications derives from work in both requirements specification and automatic programming. A key objective in both areas is the creation of a notation for describing a required system without unduly constraining the details of the solution. The Naval Research Laboratory Software Cost Reduction project developed an approach to precise, semi-formal specifications of software requirements for avionics systems (Heninger et al. 1978). Winograd (1979) argues the need for a new view of programming based on a descriptive (i.e., nonprescriptive) language. Balzer and Goldman (1979) describe criteria for designing and judging the quality of a specification language.

2.1.4 Abstraction-Based Reuse

The direct result of a software development project is a set of work products that describe and implement a software solution to a customer-defined problem. Abstraction-based reuse (Campbell 1989) provides a means for representing a product family as a set of adaptable work products and for deriving instances of each work product to produce a particular system product. An adaptable work product concisely represents a family of similar work products that vary in well-defined ways. This supports the localizing of potential changes so that the cost of modification or reuse is minimized for likely changes.

The Ada generic package is a notation for representing a family of similar Ada code packages. Most word processing packages provide a "form letter" capability that can be used to represent a family of documents. Metaprogramming tools, such as TRF*, provide a flexible notation for representing families of text-based work products. Each of these mechanisms provides a facility for producing work products from a representation of a work product family.

2.2 CONTEXT FOR SYNTHESIS

In initiating a Synthesis practice, you should first consider the context in which you currently work. This context is determined by three concerns: business objectives, system engineering practices, and the objectives of a software engineering process. These concerns constrain the type of Synthesis process that an organization can adopt. Within this context, you can create a capacity for rapid, systematic delivery of similar, yet varied, systems.

* TRF is a metaprogramming tool developed by Template Software, Inc.

2.2.1 Business Objectives

Synthesis is characterized by its focus on a family of systems for a business area rather than on individual systems. This focus arises from evidence that, within a class of systems, an understanding of similarities provides significant leverage for constructing a great variety of high-quality systems cheaply and reliably.

With Synthesis, you conceive a domain not on an objective basis, but as a realization of the declared business objectives of your specific organization. Your business objectives determine the types of systems you build and who your customers are. A primary consideration in setting these objectives is the expertise already available within your organization, particularly as a result of experience in building systems in the past. Other considerations are your expectations of future customer needs and changing technology.

The environment most conducive to the effective use of Synthesis is one in which you give emphasis to long-term business objectives. A positive climate for investment in the future, possibly at the cost of deferred—but potentially greater—total return is a major advantage for realizing success with Synthesis. It is important to consider these larger business concerns when addressing the needs of a particular customer to avoid arbitrarily sacrificing longer-term interests to short-term pressures. On the other hand, Synthesis lends itself to a management philosophy of incremental commitment, both for early pay-back after a minimum initial investment and for accommodating the changing needs of a single customer or the differing needs of many.

2.2.2 System Engineering Practices

Construction of modern, complex systems is a major undertaking that often requires the coordination of many large groups of engineers with expertise in diverse fields. Some of these groups may be organizations in separate companies, working jointly or as subcontractors, to deliver a required system. Engineers follow a discipline of system engineering, in part to partition the problem and apply appropriate expertise to solve each facet of the problem most effectively. This partitioning into subsystems often follows the lines of major hardware components, but it may also serve the purpose of decomposing the system into more manageable software assemblies.

Such partitioning is compatible with Synthesis, particularly when you are able to follow a systematic approach that results in similar partitionings of similar systems. Each software partition (or subsystem) then corresponds to a member of a family of similar subsystems (i.e., software systems) that can be the responsibility of a cohesive business organization.

2.2.3 Objectives of a Software Engineering Process

The broad purposes of a software engineering process may be stated simply:

- ***Analysis.*** To allow customers and developers to communicate effectively so that the problem to be solved is understood.
- ***Synthesis.*** To allow developers to produce a solution that corresponds precisely to the problem as communicated.
- ***Evaluation.*** To allow developers and customers to validate the solution as having satisfied the actual problem.

- *Management.* To allow developers to organize and coordinate their work as a team for efficient and effective performance of the process.

A conventional approach to software development achieves these purposes, but not efficiently. Synthesis is motivated by the need for a systematic approach to tailoring the software engineering process to meet the specific needs of each organization. The key to achieving this end is to consider long-term business needs when developing software. Looking first at the similarities among systems that your organization builds provides a basis for you to become more productive by eliminating redundant effort. Your efforts to build a particular software system can be focused more efficiently on the aspects of that system that are distinctive. Synthesis prescribes the work that is necessary to achieve this potential.

2.3 AN OVERVIEW OF SYNTHESIS

The purpose of a Synthesis process is to help you better utilize your expertise about a set of similar problems and associated solutions pertinent to your business area. By viewing similar problems as a family, common characteristics provide leverage in building any particular system. Similarities support a form of standardization that enables systematic adaptation to meet the specific needs of a particular customer. The results of standardized decision making by project engineers guide appropriate adaptations of standardized, reusable work products.

The primary distinguishing features of a Synthesis process are:

- Formalization of a domain as a family of systems that share many common features, but that also vary in well-defined ways
- System building reduced to resolution of requirements and engineering decisions, representing the variations characteristic of a domain
- Reuse of software artifacts through mechanical adaptation of components to satisfy requirements and engineering decisions
- Model-based analyses of described systems to help understand the implications of system-building decisions and to evaluate alternatives

The degree to which a Synthesis process actually exhibits these features depends on the needs and abilities of the adopting organization. As a general rule, these are goals that guide the formulation of a Synthesis process but they may be moderated to formulate a process that suits each organization's circumstances.

Regardless of your specific circumstances, a Synthesis process is one which is designed to support two independent but interrelated objectives:

- To produce and deliver software systems
- To increase the productivity, product quality, manageability, and responsiveness of software production and delivery

To address these separate concerns most effectively, a Synthesis process consists of two integrated subprocesses: Application Engineering and Domain Engineering. Figure OV.2-1 shows how these processes relate to each other.

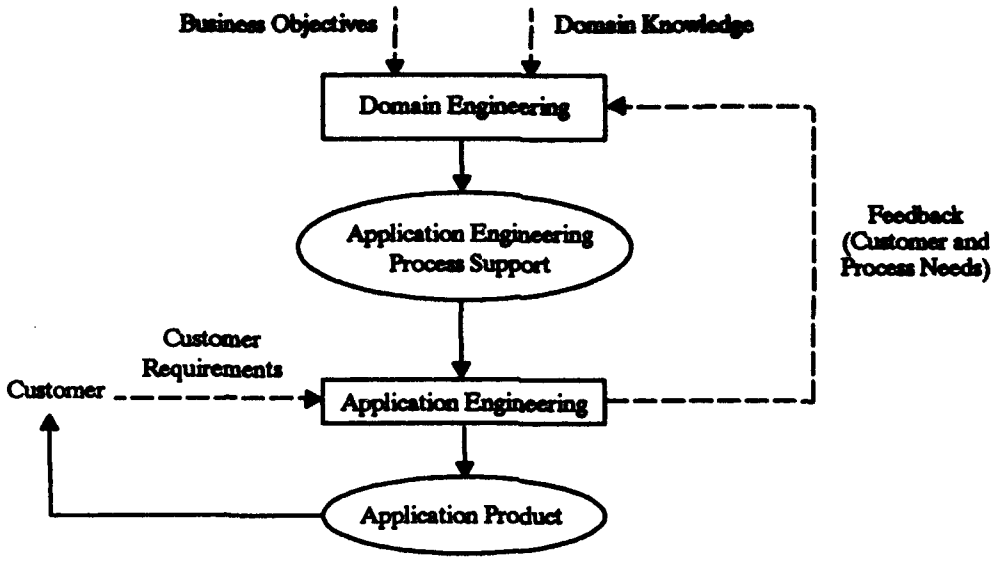


Figure OV.2-1. A Synthesis Process

Application Engineering (described further in Section AE of Parts Opp and Lev) is a standardized process by which projects produce and deliver applications to customers. In terms of objectives, this is the equivalent of conventional, "one-of-a-kind" software development. Both conventional and Synthesis approaches start with customer requirements and produce a set of software work products that are meant to satisfy those requirements. However, in a Synthesis approach, through a process of Domain Engineering (described further in Section DE of Parts Opp and Lev), you can institute a simpler, more efficient process of software development and support it with standardized, reusable work products.

Whether Application Engineering is a series of analysis, design, implementation, and testing activities or creating and validating a model to generate a required application, it focuses on requirements and engineering decisions that are sufficient to describe a particular system, given a family of such systems. Work products, including code and documentation, are mechanically derived from these decisions using adaptable forms of those work products provided by Domain Engineering.

Domain Engineering supports Application Engineering in two ways. First, it creates a set of adaptable work products that correspond to the work products that an application engineering project must produce. By identifying key decisions that are deferred until a particular system is needed and parameterizing a work product to show how it varies as a result of those decisions, Domain Engineering creates work products that are adaptable to subsequent Application Engineering decisions. Second, Domain Engineering describes a standard Application Engineering process that supports the decision making and the work-product creation appropriate to projects in the business area. The process definition institutes standard procedures and practices which Domain Engineering may augment with appropriate automated support.

Domain Engineering and Application Engineering are iterative processes to attain quality products and to support evolving needs. Application Engineering must accommodate uncertain and changing customer requirements. Domain Engineering must accommodate changing markets, as well as the evolving product and process needs of client application engineering projects.

2.3.1 Defining a Tailored Process

To adopt a reuse process (such as a Synthesis process), you need to identify one that suits your organization's needs and abilities for reuse. A key element of reuse adoption is to understand the nature and extent of your organization's current needs and capacity for reuse and develop a plan for improvements in your reuse capability. The Consortium's Reuse Capability Model (RCM), described in the *Reuse Adoption Guidebook* (Software Productivity Consortium 1992c), is a mechanism for this purpose.

The RCM provides a framework for deriving a definition of a reuse process that matches your organization's circumstances. Every Synthesis process embodies the same basic principles. However, each can differ based on the degree to which the adopting organization chooses to commit to the various success factors for improved reuse capability that the RCM defines. Success factors are grouped into four categories, reflecting different perspectives on reuse that are important to an organization:

- ***Management.*** Factors pointing to opportunities for improving management's role in facilitating reuse.

- ***Application Development.*** Factors pointing to opportunities for improving the utilization of reusable assets in the development of end-products.

- ***Asset Development.*** Factors pointing to opportunities for improving how assets are acquired or developed for reuse.

- ***Process and Technology.*** Factors pointing to opportunities for improving the effectiveness of the software development process and its use of appropriate technology.

As noted in describing the distinguishing features of a Synthesis process (at the start of Section 2.3), the form of any specific process depends not just on those features but on the needs and abilities of the adopting organization as well. An organization with unlimited abilities could adopt a Synthesis process that exhibited those features clearly; a more limited organization would adopt a process that did not require all of the abilities that these features require. A Synthesis process as a whole is affected by management factors and process and technology factors of the RCM. The Application Engineering process is further affected by application development factors. The Domain Engineering process is further affected by asset development factors.

By considering the needs and abilities of your organization with respect to each of the RCM success factors, you can design a process suited to that organization. As a guide to incrementally improving your organization's reuse capability and the implementing process, the RCM includes an implementation model consisting of four stages through which an organization's reuse practice can evolve from an initial, ad hoc reliance on the initiative of individual engineers:

- ***Opportunistic.*** Project plans allow for reuse; reusable assets are identified to support the separate creation of individual work products; effort is focused on the immediate needs of current projects.

- ***Integrated.*** Reuse is an integrated element of the standard development process; assets are developed to enable multiple use based on current needs.

- ***Leveraged.*** Assets are developed for reuse considering both current and likely future needs across a product line. Projects are viewed as agents of a business-area organization and work to gain maximum leverage from and enhance the supported domain.

- ***Anticipating.*** The potential for reuse is a primary consideration in how problems are viewed. New business opportunities are sought and evaluated in terms of how well they exploit available reusable assets. Each project is initiated with an understanding of how its problem corresponds to problems previously solved.

An organization may progress incrementally through these stages of implementation for lower risk or it may commit directly to the adoption of a more advanced stage to get corresponding benefits sooner.

Part Opp of this guidebook describes a Synthesis process that is oriented to the opportunistic stage of reuse implementation. Part Lev describes a Synthesis process that is oriented to the leveraged stage. Other Synthesis processeses could be defined at the integrated or anticipating stage. Depending on your organization's circumstances concerning reuse, you are likely to prefer one of these processes over the others. Any such process definition, however, is best viewed as a prototype from which you should derive a process that is tailored to your specific circumstances. The remainder of this section characterizes the RCM and Synthesis processes in relation to each of the RCM's four implementation stages.

The two Synthesis processes described in this guidebook are appropriate to organizations that are targeting either the opportunistic or the leveraged stage of implementation. Each process has been designed to reflect the assumptions and limitations appropriate to the targeted stage of instituting a reuse program. For example, at the opportunistic stage, application engineers find and evaluate reusable components specifically for each work product. At the leveraged stage, reuse by application engineers is indirect in terms of requirements-level variations and leads implicitly to integrated reuse across the work products that constitute the product. This difference is traceable to success factors identified respectively with these stages in the RCM. Sections 2.3.2 through 2.3.5 describe the success factors that guide the design of a Synthesis process appropriate to a particular RCM stage of implementation.

2.3.2 An Opportunistic Synthesis Process

Your organization can adopt an opportunistic Synthesis process if it is targeting a new state of practice similar to the following:

- ***Management.*** In planning an application engineering project, project management makes allowance for reuse of existing work products. Project engineers are expected to reuse existing work products or components appropriately without special support or coordination. Domain Engineering is managed separately with the goal of increasing opportunities for reuse.

- ***Application Development.*** Application Engineering is work product oriented, as in conventional practice. When creating an assigned work product, it is the engineer's responsibility to recognize and exploit opportunities for reuse of existing relevant components, particularly those provided by Domain Engineering.

- ***Asset Development.*** Domain Engineering analyzes previously developed work products to find components to combine and/or refine in ways that increase opportunities for their reuse. The

known needs of active or impending application engineering projects guide this analysis and implementation.

- ***Process and Technology.*** The Application Engineering process is not substantially changed to accommodate reuse. A Domain Engineering process is an adaptation of that process. Existing off-the-shelf technology, such as Ada generics and word processing packages, may be used to create adaptable work products or components (i.e., families).

At the opportunistic stage of implementation, the Application Engineering process is organized and managed in a conventional manner. The emphasis is still on the development of "one-of-a-kind" systems and the phased completion and review of corresponding deliverable work products. For example, DOD-STD-2167A (Department of Defense 1988) has traditionally been interpreted as a "waterfall" model, leading through a series of phases, such as software requirements analysis, preliminary design, and detailed design. An opportunistic reuse process has the following implications for an organization:

- Reuse is a responsibility of individual engineers as they produce assigned work products. Domain Engineering attempts to provide families of work products (as a whole or in parts) that consolidate past experience in building similar systems in the domain. It is left to each engineer to decide when there is sufficient opportunity for reuse.

- Domain engineering and application engineering projects are managed independently. The immediate needs of projects to satisfy customer requirements override any longer term objectives for the domain. When conflicts arise, Domain Engineering gives priority to supporting immediate project needs. Whenever possible, those needs are satisfied in a way that comes closest to matching long-term objectives.

An opportunistic reuse process is one in which reuse efforts are oriented entirely to supporting the current needs of a business group's active (and imminent) application engineering projects. An opportunistic Synthesis process is directed toward increasing the opportunities for effective reuse without causing significant changes in the application development process already familiar to managers and engineers.

In opportunistic Synthesis, a project follows its normal process (similar to that of Figure OV.2-2, which was derived following Evolutionary Spiral Process [ESP] guidance), changed only by the addition of general, low-level guidelines for individual engineers to find and exploit relevant reusable assets. Domain Engineering (Figure OV.2-3) is an attempt to identify, organize, and improve assets from previously built systems (a key element of domain knowledge) so that those assets will be useful in satisfying engineers' current needs. Because Application Engineering, at this stage of reuse program implementation, emphasizes the direct creation of documents and code, Domain Management focuses Domain Engineering efforts on opportunities for standardizing the form and content of particular work products. The domain engineer identifies sets of similar work products, defines each set's common and varying features, and uses those features to represent the set as a family from which application engineers can extract instances.

2.3.3 An Integrated Synthesis Process

Your organization can adopt an integrated Synthesis process if it is targeting a new state of practice similar to the following:

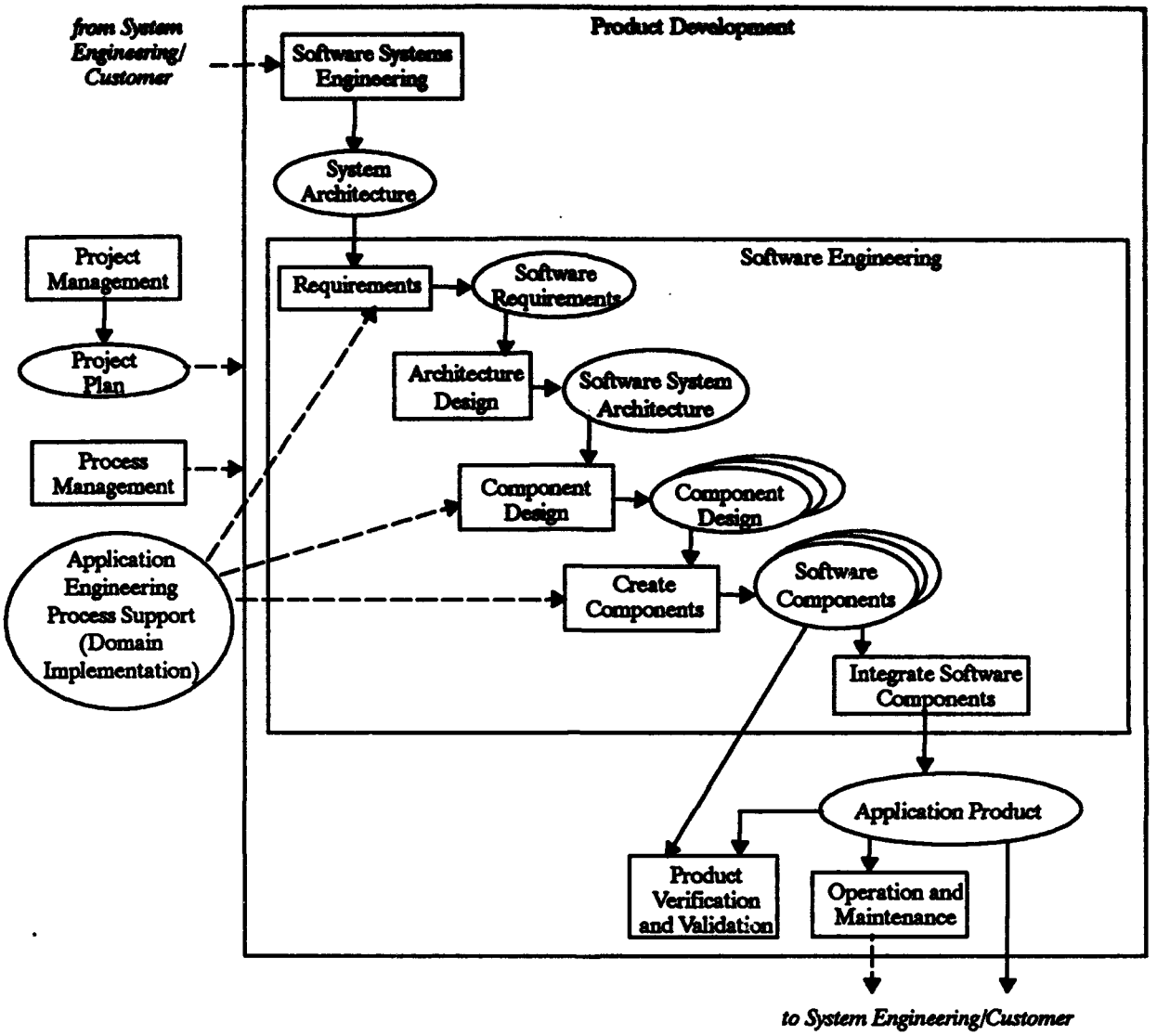


Figure OV.2-2. A Simple Example of an Application Engineering Process for Opportunistic Reuse

- ***Management.*** Management recognizes that application engineering projects share a common base of domain expertise. Domain and application project planning are integrated to avoid duplicate effort and ensure timely domain support for critical application project needs. The domain supports project needs and objectives as long as those needs and objectives fit with the accepted domain scope and domain resources suffice.

- ***Application Development.*** An Application Engineering process is work-product oriented but application engineers expect to be able to produce a reasonable approximation of the deliverable product from an Application Model and Adaptable Components and create the final product without major structural rework. A unified core Application Model, with extensions specific to each work product, drives production of all required work products.

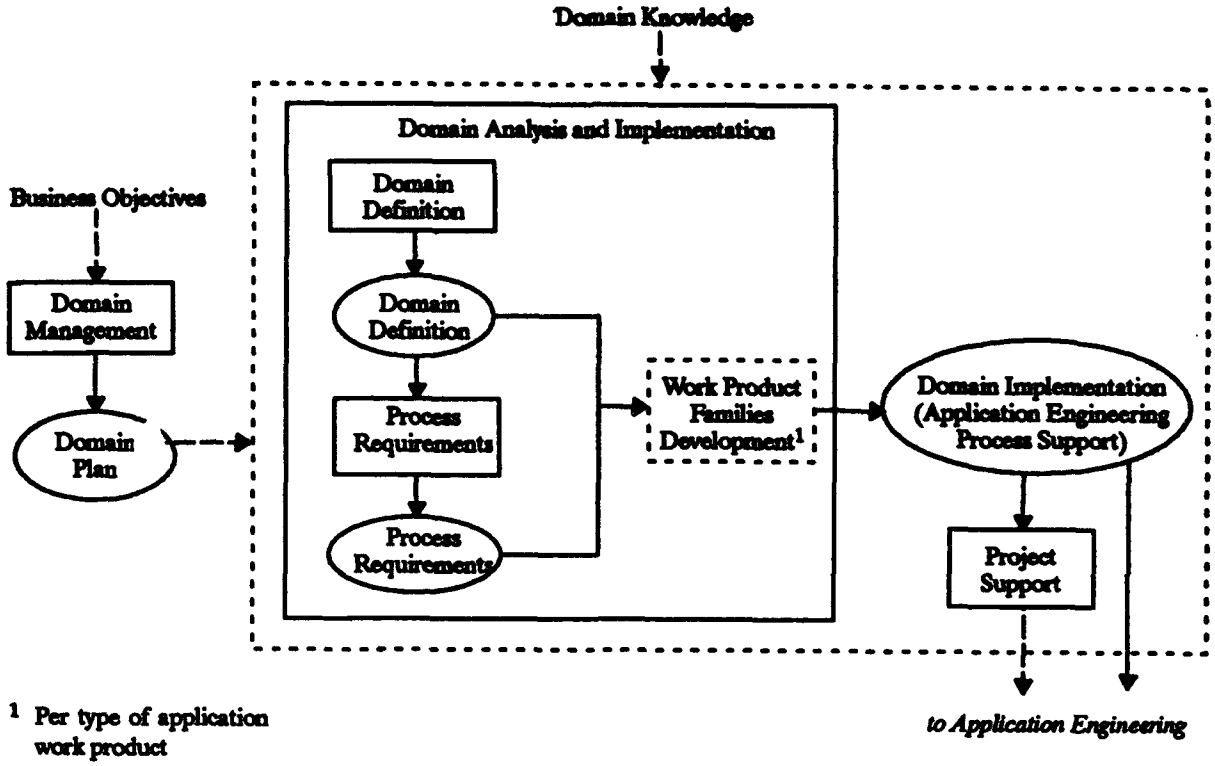


Figure OV.2-3. A Domain Engineering Process for Opportunistic Reuse (Simplified)

- ***Asset Development.*** Every required application engineering work product is represented as a family. Whenever there are similar variations in related work products, there is a common source of variability to which traceability is maintained. Both the Application Engineering product family and the associated process are tailored to the needs of active projects. Problems due to Domain Engineering are tracked and used to motivate enhancements.

- ***Process and Technology.*** The Domain Engineering and Application Engineering processes are integrated with explicit domain support for active application projects and specific feedback on needs from those projects to Domain Engineering. Application Engineering relies on Domain Engineering technology except with explicit waivers.

At the integrated stage of implementation, the advantages of leveraged effort compete with the immediate concerns of individual projects. Solution approaches that can build on existing assets are preferred over unique, single-use solutions. The long-term needs of the business organization influence decisions on the near-term use of resources. Although application projects may still have to perform one-of-a-kind effort, it is easier to distinguish essential from arbitrary product variations.

Application Engineering and Domain Engineering processes typical of an integrated Synthesis practice have not yet been defined.

2.3.4 A Leveraged Synthesis Process

Your organization can adopt a leveraged Synthesis process if it is targeting a new state of practice similar to the following:

- ***Management.*** Management views a business organization as a vehicle for building systems in a chosen problem domain. Management of the domain and its client application engineering projects is unified. Projects are initiated in a particular organization when a problem fits properly within its domain. Domain Engineering objectives consider project objectives but are motivated predominantly by the need to achieve strategic business objectives.

- ***Application Development.*** Application engineers specify a problem and its solution in the form of an Application Model. Changes in the problem or changes in its understanding or alternative solutions lead to a modified Application Model. Deliverable work products are generated directly from an Application Model and Adaptable Components. Normally, Domain Engineering handles discrepancies in a product; occasionally (e.g., when project constraints conflict with domain objectives), Application Engineering directly modifies generated work products.

- ***Asset Development.*** Development focuses on supporting a unified product family that comprises families of deliverable and supporting work products. Domain Engineering gives application engineering projects an ability to derive a product entirely from a model representing requirements and engineering decisions and reusable work product families.

- ***Process and Technology.*** Domain Engineering creates a standard, reuse-oriented Application Engineering process for projects in the domain. Key aspects of that process are supported by specially built automation.

At the leveraged stage of implementation, the Application Engineering process is organized and managed in a standardized way based on the nature of the business as determined by Domain Engineering. The emphasis is on producing and delivering systems that meet the needs of customers; however, only systems that fit the strategic charter of the organization are initiated. This process has the following implications for an organization:

- The core expertise of the business is focused on effective domain engineering. Application engineering projects are agents of the domain and leverage the core expertise appropriately. Management of the domain is unified to achieve a proper balance between strategic business objectives and current project needs.

- Most of the Application Engineering process focuses on capturing and refining an understanding of a customer's requirements in an Application Model and comparing standard alternative solutions that satisfy the model. Work product creation is reuse-based and largely automated, as is support for much of the rest of the process.

A leveraged Synthesis process is one in which strategic business needs, tempered by past project experience and current project needs, define a domain that motivates reuse as a vehicle for achieving long-term organizational objectives. A separate application engineering project is instituted to serve each customer having a problem judged to be within the domain.

In a leveraged Synthesis process, projects follow a standardized, reuse-driven process (e.g., like the one shown in Figure OV.2-4) that results from an analysis by Domain Engineering of how projects can operate most effectively doing family-oriented reuse (Figure OV.2-5). At the leveraged stage of implementation, Domain Management focuses Domain Engineering efforts on the adaptable standardization of the process and products of Application Engineering to improve the life-cycle productivity of

the total software development enterprise. The deliverable work products of each Application Engineering project are mechanically derived from an Application Model and domain-specific adaptable specifications, designs, and implementations, resulting in a tailored, integrated software product. At this stage, the concept of a reuse library is subsumed into a broader framework of process support.

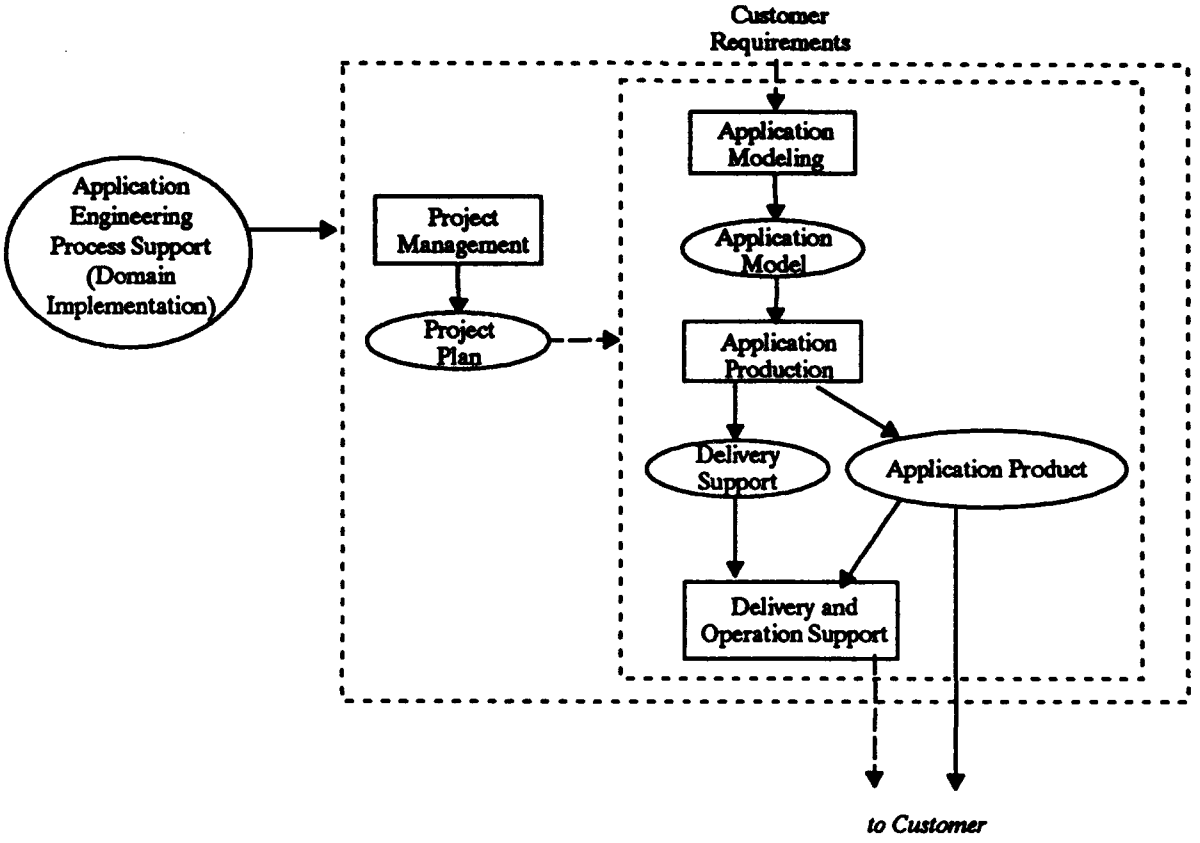


Figure OV.2-4. A Prototypical Application Engineering Process for Leveraged Reuse

2.3.5 An Anticipating Synthesis Process

Your organization can adopt an anticipating Synthesis process if it is targeting a new state of practice similar to the following:

- *Management.* Domain and application projects are managed as vehicles to serving a perceived customer market. Management's objectives try to anticipate and stimulate customer expectations to match the organization's capabilities and assets. Application projects are initiated because of a recognition that they fit with and can leverage the organization's expertise. Domain and application project resouces are strategically coordinated to seek optimum value from the investment.

- *Application Development.* The Application Product, including deliverable versions of all work products, are derived entirely from an Application Model leveraged with domain assets. The process is highly iterative, customer-involved, and systematic, including the use of predictive models of system properties that aid the engineer in rapidly delivering an acceptable product.

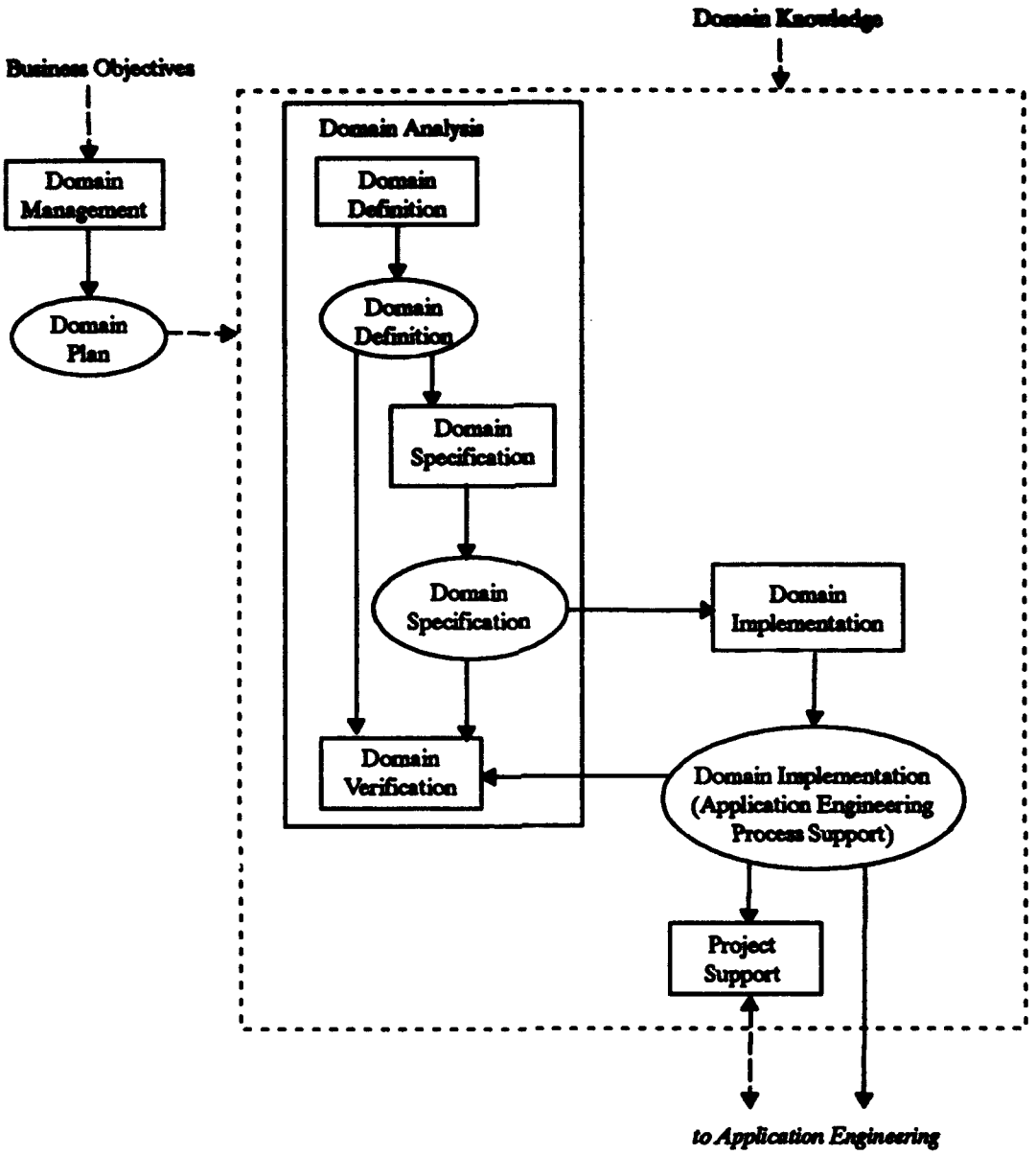


Figure OV.2-5. A Domain Engineering Process for Leveraged Reuse

Any inadequacies that cannot be remedied effectively through the Application Model are referred to Domain Engineering for improvement.

- ***Asset Development.*** Domain Engineering provides assets and automation sufficient to build and deliver complete products that match the needs of Application Engineering projects. Previously undiscovered needs are rapidly introduced within a flexible and consistent framework of existing capabilities. Efforts to identify new needs and improve responsiveness to existing needs stimulate continuing evolution of the domain.

- ***Process and Technology.*** Both the Domain Engineering and Application Engineering processes are systematically adapted to suit the evolving needs of the organization and its market. Each

of the processes is optimally automated based on an analysis of cost and benefit. The processes and supporting technologies are integrated for rapid response to changing circumstances.

At the anticipating stage of implementation, the emphasis is on anticipating and creating a market for an organization's product line. The focus of investment is on predicting future market needs and creating assets that will serve those needs. Application projects are sought that not only exploit existing capabilities but also provide opportunities for enhancing those capabilities. Application projects are seen as a means to apply the knowledge and expertise embodied in a domain to problems that can benefit from this knowledge and expertise.

Application Engineering and Domain Engineering processes typical of an anticipating Synthesis practice have not yet been defined.

PART OPP: OPPORTUNISTIC SYNTHESIS

This page intentionally left blank.

OV. OVERVIEW OF AN OPPORTUNISTIC SYNTHESIS PROCESS

This part of the guidebook presents an opportunistic Synthesis process. This is a process suitable for an organization that has achieved the goals associated with the opportunistic stage of reuse capability, as defined by the RCM. To help you understand whether an opportunistic process suits your organization, this section discusses the assumptions that underlie the process and the nature of software development when using the process.

1. UNDERLYING ASSUMPTIONS

The RCM defines four stages of reuse capability implementation and characterizes each stage by a set of goals. The opportunistic Synthesis process described in this part of the guidebook was designed to fit the goals associated with the opportunistic stage as described in the Fundamentals section of the overview to this guidebook (Section OV.2). To adopt this process, an organization will have targeted those goals as a minimum, rather than the more ambitious goals associated with other stages. Your organization may choose to adopt the opportunistic process even if it has the potential to attain goals associated with more advanced stages of reuse capability implementation. However, this process does not depend upon nor require your organization to attain any of the more ambitious goals associated with more advanced stages.

The goals associated with the opportunistic stage introduce requirements for a process to be used by organizations targeting that stage. The Synthesis process described in this part of the guidebook is one example of a process that an organization targeting the opportunistic stage could adopt. It differs from other such processes because it reflects assumptions about an organization's circumstances that extend those implied by the RCM goals for the opportunistic stage. Section 2.3 in Part Syn of the guidebook describes characteristics common to all Synthesis processes, particularly that the process comprises iteratively cooperating Domain Engineering and Application Engineering subprocesses. Additional assumptions that distinguish the opportunistic Synthesis process are as follows:

- ***Managers and engineers are working in a familiar domain.*** Application engineers are building a product similar to one they have built before, and domain engineers are analyzing products of a sort with which they have experience. The "theory" underlying the product is familiar to them, at least intuitively; based on previous experiences, they have some idea of appropriate design structures (e.g., process communication models) and testing strategies. Also, they know the types of work products that they will produce as part of their normal Application Engineering routine (requirements documents, design documents, etc.); the form and structure of each work product (e.g., DOD-STD-2167A); and the process by which projects produce work products.

In other words, domain and application engineers understand existing problems and solutions in the domain. Generally speaking, domain engineers do not have to create new ways to describe problems and solutions. They only need to document them, codifying their intuitions, common sense, and day-to-day practices. They can assume application engineers understand problems similarly and know how to make appropriate use of existing solutions.

- ***The organization has developed and has access to similar systems.*** In the near-term, the organization expects to develop systems similar to those that it has built in the past. Knowledge, experience, and existing work products are, therefore, relevant to current and planned systems. Domain engineers have access to application engineering work products of the types they plan to make available for reuse. These work products include code, documentation, test plans, and anything else that might have reuse value.

  This assumption improves the viability of a domain, independent of reuse stage. Domain engineers can rely entirely on their knowledge and experience, but detailed analyses of previously-developed work products can provide increased insight and ground the result in reality. Existing work products provide raw material from which a reuse library can be derived, with more confidence and less effort than original development provides.

- ***The role of Domain Engineering is to increase opportunities for reuse during Application Engineering.*** This assumption has two meanings. First, domain engineers obtain, codify, and organize domain knowledge and existing work products. Domain engineers focus on providing accurately-documented, existing work products to application engineers.

  Second, domain engineers sometimes reengineer existing work products to satisfy new needs. Strictly speaking, reengineering does not occur in a pure opportunistic process, where domain engineers only take advantage of existing opportunities—i.e., existing work products. The pure opportunistic process limits reuse to needs of past application engineering projects. This approach is undesirable if domain engineers perceive greater opportunity by satisfying new needs, they should provide a solution to those needs.

- ***Application engineers want to use a familiar process.*** Application engineers are already familiar with the types of work products that their customers require for the domain, and with the process for producing them. They want to use this process on the current project. Application engineers, therefore, have expectations about the types of work products they will produce and the order in which these work products are created. Reuse must not affect either the form or the order. In other words, it must not influence the overall Application Engineering process.

- ***Reuse focuses on resolving variations among members of a work product family.*** This means that sufficient flexibility in reuse depends on accommodating variations. Domain engineers standardize a work product family by identifying commonalities and variabilities that characterize its members. Commonalities provide the basis for potential leverage, in that they express general Application Engineering needs. Variations correspond to decisions that application engineers need to make to express their needs precisely.

- ***Domain Engineering supports only one application engineering project at a time.*** Domain engineers scope a domain, and focus their efforts on creating reusable products, based on the needs of a single application engineering project. Domain engineers support a current project that is similar to previous projects and a predictor of future projects. The organization expects to reuse the resulting assets in both the current and future projects.

This assumption provides domain engineers with a strong focus. Their examination of existing material and the goals they set for reuse are tied to one project, not to an entire organization or to anticipated projects (a consideration at later stages of reuse capability). Also, this assumption helps justify having the current project directly fund Domain Engineering work in the early stages of reuse program implementation.

2. SOFTWARE DEVELOPMENT WITH AN OPPORTUNISTIC SYNTHESIS PROCESS

This section is a general overview of the opportunistic Synthesis process presented in this part of the guidebook. It gives an overall feel for software development and management as practiced by an organization with opportunistic reuse capabilities, without elaborating every activity or every possible variation or interpretation of the process. As Figure OV.2-1 in Part Syn depicts, Domain Engineering and Application Engineering activities, and the interactions between them, are defining aspects of any Synthesis process. This section describes this opportunistic Synthesis process from three perspectives:

- The organization controlling Application Engineering (application software development to satisfy a particular customer) and Domain Engineering (the effort invested in reuse) within a domain
- An individual application engineering project
- The domain engineering project for a domain

This section concludes with a brief scenario of how Domain Engineering and an Application Engineering project interact.

2.1. Organizational Perspective

In opportunistic reuse, an organization's strategy is to leverage reusable assets as best it can with minimal investment. Reuse at the opportunistic stage is not a key business-area strategy but rather an opportunity exploited on a project-by-project basis. The business organization may allocate limited initial funding, but in general would like to operate in a "pay-as-you-go" mode: each increment of investment in reuse, whether from organization or project funds, should result in a payoff for a specific project (against which direct costs are charged and justified).

This business/management orientation motivates how Synthesis is interpreted for the opportunistic context. As always with Synthesis, Domain Engineering exists to support Application Engineering. However, in the opportunistic process, Domain Engineering focuses on current needs. The current needs are defined by the application engineering project targeted for support by Domain Engineering. However, the organization can expect Domain Engineering products to be useful on other application engineering projects that are building applications in the same domain.

The first time an organization practices Synthesis, it may have to expend preliminary effort, beyond current project needs, to get up to speed. Once a domain exists, there are reusable assets that future projects can use as well. Each time an organization begins a new project, it considers focusing Domain Engineering on that project. The domain engineering project revises existing domain work products to reflect any differing needs of this new application engineering project. As the project proceeds, Domain Engineering adds and revises reusable assets to correspond to the current concept of the commonalities and variabilities among systems in the domain and their associated work products.

2.2. Application Engineering Perspective

At the opportunistic stage of reuse program implementation, the current project shares a framework of process and work products with past projects. This framework links the reuse program's success to the individual application engineer's ability to take advantage of reuse opportunities as he builds each work product.

Application Engineering follows its organization's normal process, probably similar to the waterfall-like process depicted in Figure OV-1. However, whenever Domain Engineering has provided reusable assets associated with a particular type of work product, the application engineer has the ability (and responsibility) to evaluate and exploit opportunities for reuse in creating that work product. Such reuse can leverage existing work products of the same type, either in their structure, in the content of portions, or in their entirety. In any case, the application engineer will likely have to tailor the resulting draft work product for it to meet fully the precise needs of the current project. Whenever the engineer is unable to create the entire work product through reuse, he still has the ability to work conventionally to create the appropriate final work product.

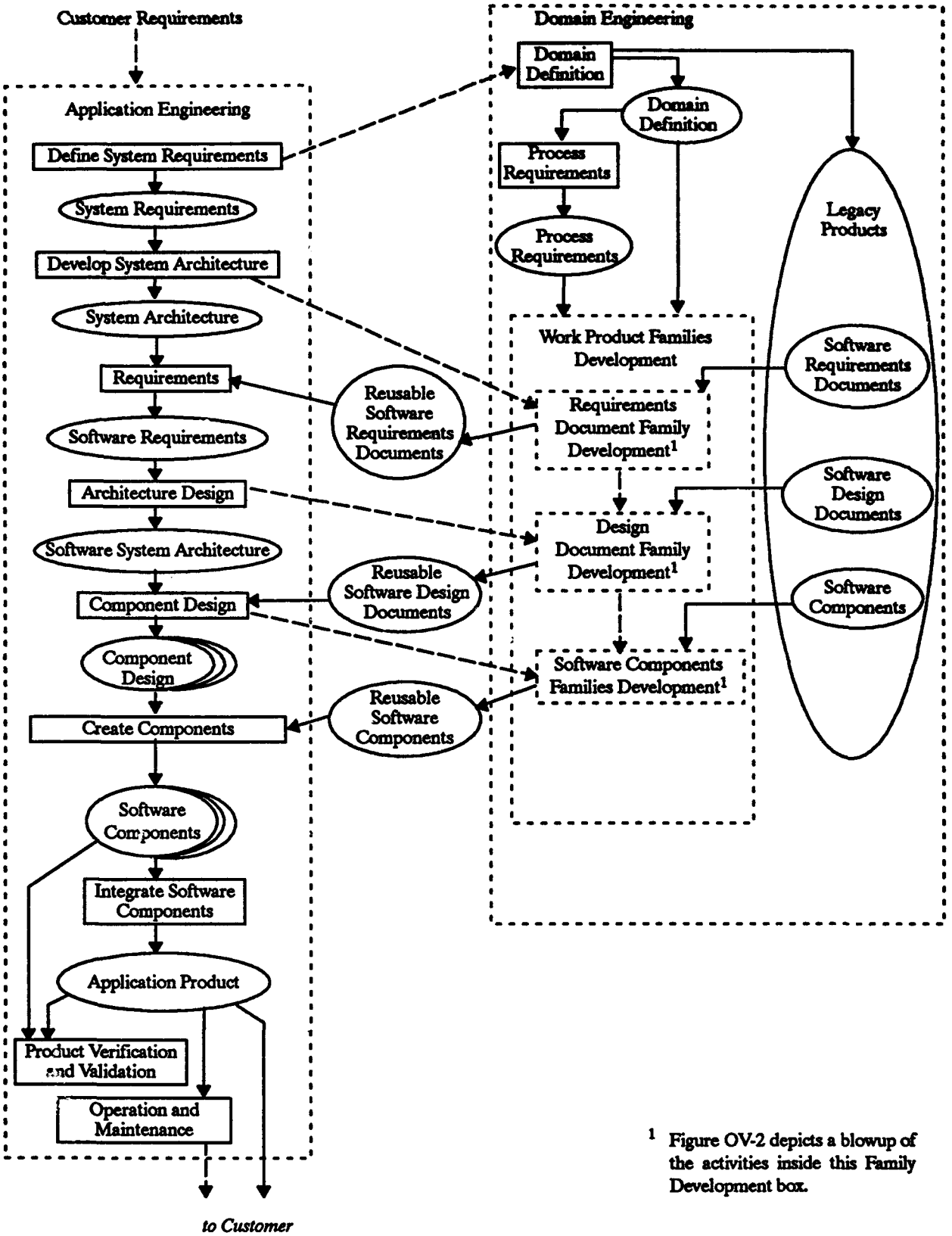
2.3. Domain Engineering Perspective

In opportunistic reuse, the goal of Domain Engineering is to support reuse by an application engineering project in a way that minimizes an application engineer's effort to discover whether reuse is feasible. Domain Engineering adopts a narrow focus on each work product that application engineers have to create. By analyzing the commonalities and variabilities of a work product type that has been produced by previous projects, domain engineers can focus the attention of the Application Engineering to those work products and aspects of each that are most likely to yield effective reuse.

To have sufficient context, Domain Engineering works on supporting reuse for a work product just prior to the planned Application Engineering activity for producing it. Domain Engineering uses work products and other information from preceding phases of Application Engineering to determine which reusable assets are most likely to fulfill the needs of the next phase. Ideally, the past is a perfect predictor of the future, and assets of past projects fit perfectly with the current project. In practice, application engineers will need assistance in making the correct match and adjustments between need and available reusable assets.

Where the predicted needs for the current project only partly match existing domain assets, domain management must decide whether original work is justified. When the current project has a need that is different from previous projects, the need may be unique to that project or it may indicate an important new need of all future projects. As a rule, until a need can be clearly characterized as a common need, it remains outside the scope of further Domain Engineering consideration. In opportunistic reuse, there must be an expectation of future recurring uses to justify a reuse-oriented investment. Application engineers must handle whatever custom development is necessary to address any needs perceived as unique to their project. However, when a similar need arises for a subsequent project, there is then a foundation of existing application work products from which Domain Engineering can create reusable assets and enhance the domain.

Even when common needs clearly exist, Domain Engineering is not obligated to provide reusable assets for every work product of Application Engineering. The domain's resources must be intelligently allocated to work products that offer the most potential benefit from reuse. Domain engineers work with particular work products based on the following criteria:



[1] Figure OV-2 depicts a blowup of the activities inside this Family Development box.

Figure OV-1. Example Application Engineering and Domain Engineering Interaction

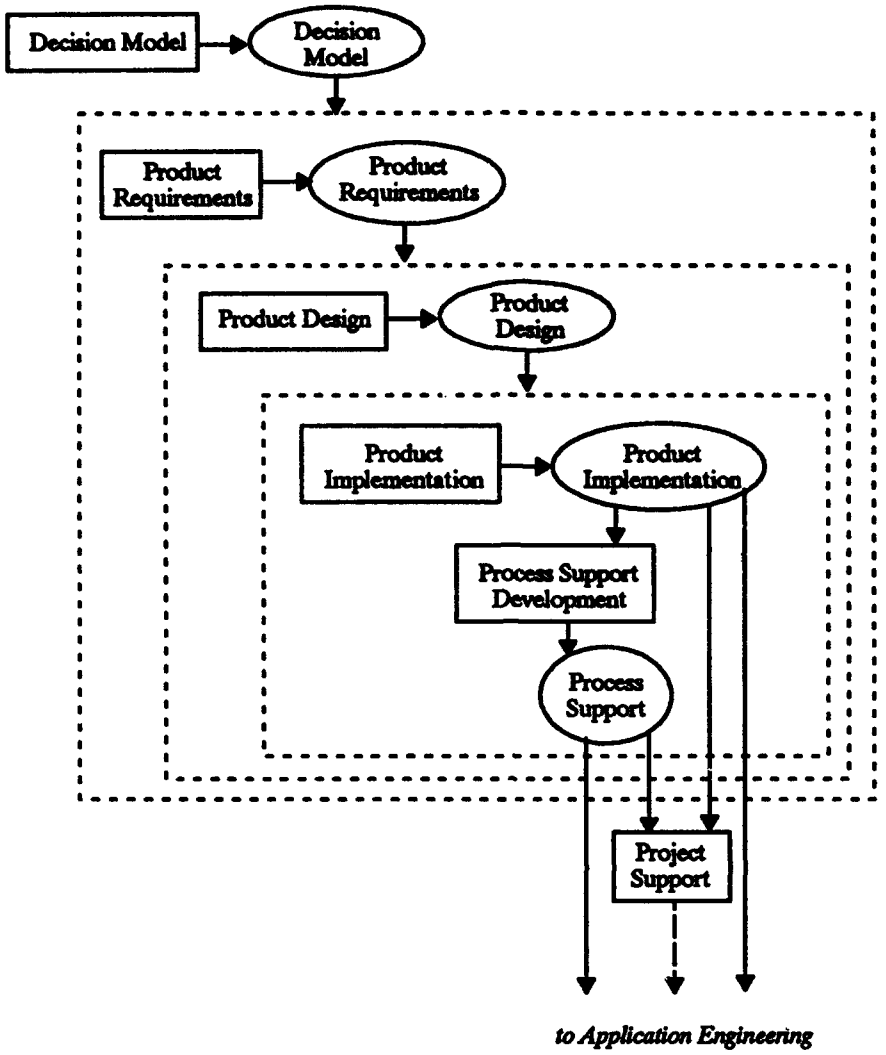


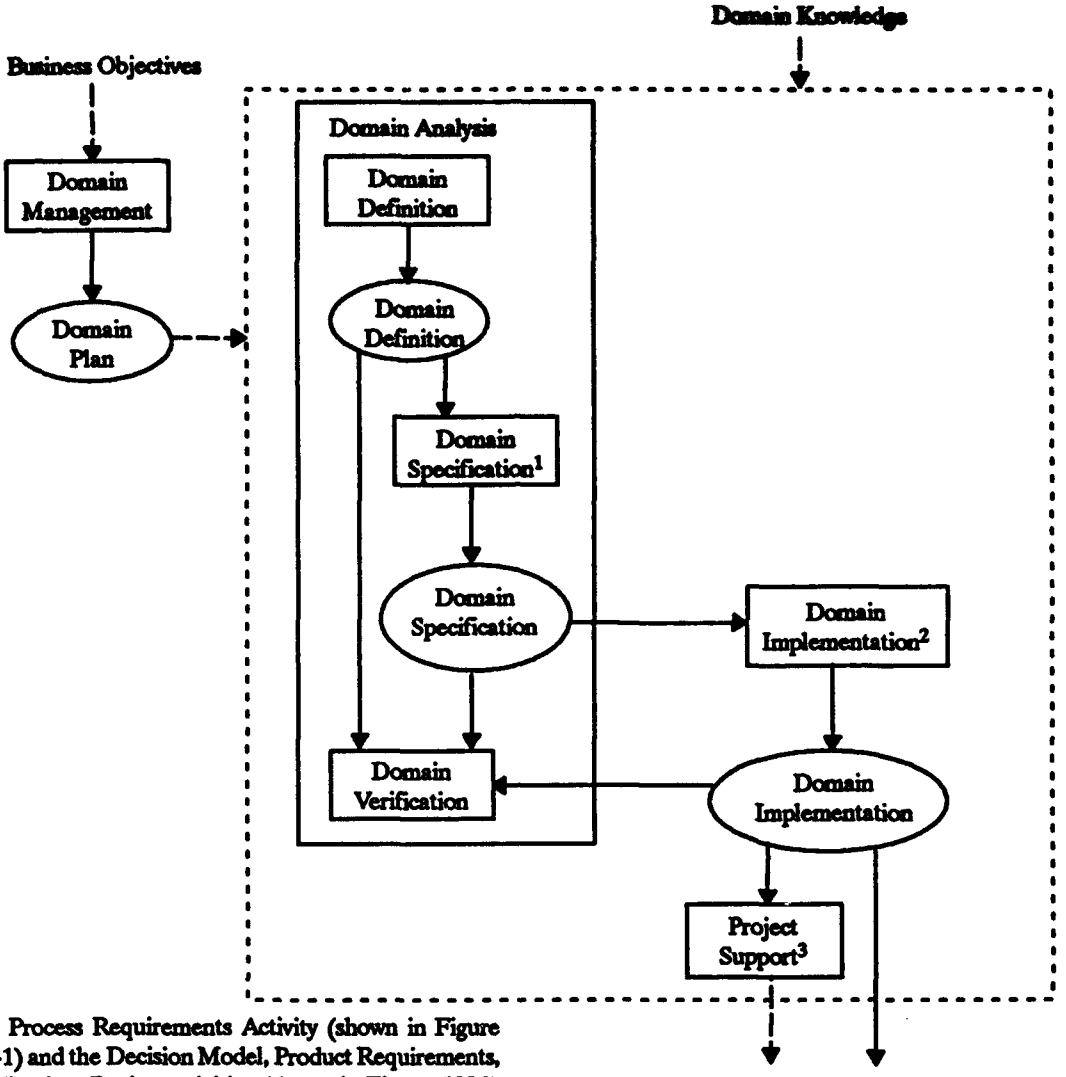
Figure OV-2. Blowup of a Work Product Family Development Activity Group

- Domain Engineering can extract the needed work products from previous applications (perhaps with some modification to put them into a more reusable form).
- Domain Engineering can provide the needed work products within the project's schedule for this work.
- The cost to recover and reuse an asset will, at most, equal the cost for addressing the need via custom development.

Figure OV-3 shows the relationship between this work product development approach (Figures OV-1 and OV-2) and Domain Engineering as discussed in remainder of this guidebook.

2.4. An Example Scenario

Consider the hypothetical process in Figure OV-1. Domain Engineering supports the reuse of three types of work products: requirements documents, design documents, and computer software units.



[1] The Process Requirements Activity (shown in Figure OV-1) and the Decision Model, Product Requirements, and Product Design activities (shown in Figure OV-2) are part of the Domain Specification Activity.

[2] The Product Implementation and Process Support activities (shown in Figure OV-2) are part of the Domain Implementation Activity.

[3] The Project Support Activity as shown here represents the combination of all work product Project Support activities depicted in Figure OV-2.

Figure OV-3. Relationship Between Work Product Family Development and Domain Engineering

Application Engineering begins with software systems engineering, establishing the system requirements and system design. During software systems engineering, Domain Engineering creates a Domain Definition, a general description of the domain scope, and Process Requirements, a description of the expected process and work products of Application Engineering. The Domain Definition identifies existing systems that are in the domain and available to Domain Engineering as sources

of raw material. Normally, a new project continues a line of business that the organization has been pursuing; this presumption is the basis for intuitive expectations for reuse opportunities.

As the project progresses through the Application Engineering process producing work products, the application engineers look for opportunities to reuse existing work products, in whole or in part. In order to make reuse attempts for particular work products more effective, Domain Engineering attempts to organize, improve, combine, and document existing work products of that type. In this example, Domain Engineering has determined that software requirements documents, software design documents, and software components offer the best opportunities for reuse by the current project. As a result, domain engineers focus their efforts on enhancing the reusability of those work products and helping application engineers recognize and exploit them as opportunities arise.

Application Engineering benefits from opportunistic Synthesis because Domain Engineering identifies and provides potentially reusable existing assets based on perceived needs of the current project. This focus makes the likelihood of reuse greater than if Domain Engineering populated the library with arbitrary components. Because the organization expects more business in the domain, it anticipates that this effort will benefit future projects as well.

DE. DOMAIN ENGINEERING OVERVIEW

1. GETTING STARTED

Domain Engineering is an activity of a Synthesis process that creates and supports work products which support the Application Engineering process in a business area, particularly with respect to the needs of a specific project (hereafter termed the targeted project). Domain Engineering is a comprehensive iterative life-cycle process with management, analysis, implementation, and support activities for a product family. A product family is represented by a collection of work product families.

1.1 Objectives

The objectives of Domain Engineering are to:

- Organize and direct resources to facilitate opportunities for reuse of existing application engineering work products by the targeted project within an organization
- Define the nature, extent, and substance of a set of work product families that complements those opportunities for reuse

1.2 Required Information

Domain Engineering requires the following information:

- Domain knowledge (including existing products)
- Business objectives

1.3 Required Knowledge and Experience

Domain Engineering requires domain and software knowledge and experience in:

- The needs that motivate systems in the domain (i.e., application engineering work products) and associated work products
- The environments in which the systems in the domain will operate
- How the systems in the domain are built
- How application engineering projects in the domain are managed

2. PRODUCT DESCRIPTION

Domain Engineering creates four work products: Domain Plan, Domain Definition, Domain Specification, and Domain Implementation. Domain engineers evolve these products in subsequent iterations of Domain Engineering to support future projects, consistent with organizational business objectives.

2.1 Domain Plan

Purpose	A Domain Plan (see Section DE.1) establishes the scope of domain development and defines the tasks and resource allocations for domain development increments.
Verification Criteria	The expected needs of planned projects in the business area are sufficient to compensate for projected costs and risks of domain development.

2.2 Domain Definition

Purpose	A Domain Definition (see Section DE.2.1) defines the informal scope and orientation that characterize a viable domain.
Verification Criteria	The Domain Definition captures sufficient information to allow domain engineers to describe any existing or potential system in the domain (in particular, the system being built by the targeted project).

2.3 Domain Specification

Purpose	A Domain Specification (see Section DE.2.2) defines a set of work product families that provides increased opportunities for reuse to the targeted project within its Application Engineering process.
Verification Criteria	The Domain Specification precisely expresses the domain as captured in the Domain Definition.

2.4 Domain Implementation

Purpose	A Domain Implementation (see Section DE.3) is an implementation (with documentation and automated support) of a set of work product families, as prescribed by the Domain Specification.
Verification Criteria	The Domain Implementation provides each work product family described in the Domain Specification.

3. PROCESS DESCRIPTION

Domain Engineering is an interaction among the four steps shown in Figure DE-1.

3.1 Procedure

Follow these steps for the Domain Engineering Activity.

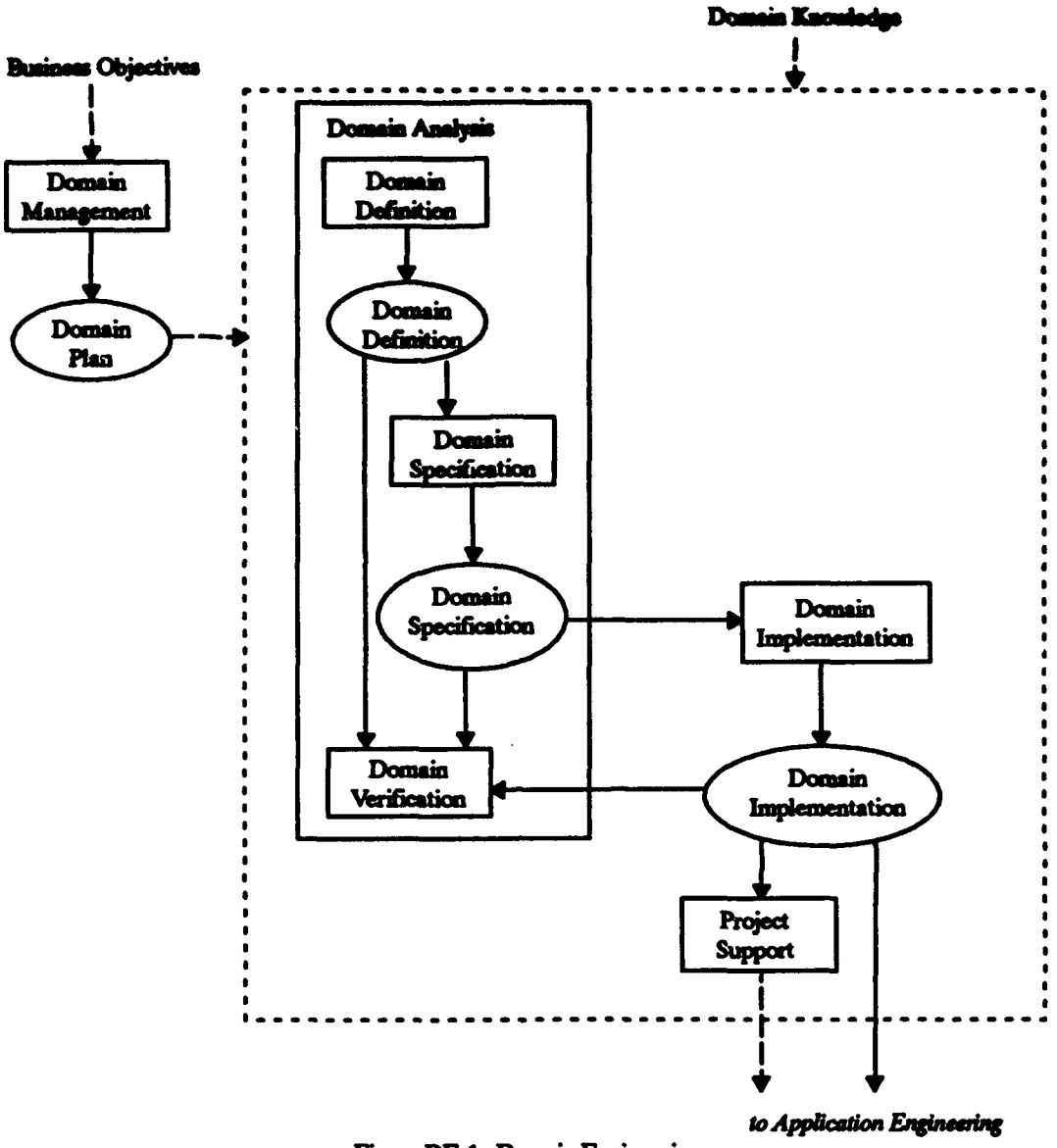


Figure DE-1. Domain Engineering

Step: Domain Management Activity

Action	Plan, monitor, and control the use of domain resources to provide reusable work product families for a domain of interest to projects within a business-area organization.
Result	Domain Plan
Heuristics	Define near-term objectives that support the application engineering project targeted by Domain Engineering. Organize and manage domain resources to achieve those objectives.

Step: Domain Analysis Activity

Action	Scope and specify a domain based on an analysis of needs of a targeted project in an organization.

Input Domain Plan

Result

- Domain Definition
- Domain Specification

Heuristics

- Create an informal definition of the domain. Characterize its scope, the aspects common to all systems in the domain, and the features that vary across systems in the domain. Also characterize the work products commonly produced as part of Application Engineering. Explicitly state what is not part of the domain. Provide a glossary of common terms. Assess the viability of supporting each of the aspects you have characterized.
- Precisely specify problems within the scope of the domain. Describe both common problems and variations in those problems. Specify solutions to the problems in the domain so that the solutions vary in the same way as the problems. Identify important work products (i.e., those susceptible to reuse). Specify the Application Engineering process commonly used in the domain, showing when a given work product is developed. Determine the work products for which (reusable) work product families will be developed.

Step: Domain Implementation Activity

Action Implement the domain as defined by the Domain Specification.

Input

- Domain Definition
- Domain Specification
- Domain Plan

Result Domain Implementation

Heuristics

- Implement the work product families and process described in the Domain Specification. Incorporate variations into the implementation of the solutions. Structure the implementation of the solutions as a set of work product families, each of which supports reuse for a particular work product that application engineers might want to develop.
- Describe conventions, procedures, and standards for using these work products. Document how application engineers can perform reuse based on these descriptions. If time and resources permit, automate routine time-consuming tasks.

Step: Project Support Activity

Action Support a project in performing the Application Engineering process.

Input Domain Implementation

Heuristics Deliver, install, and support the Domain Implementation for use by the targeted project.

3.2 Risk Management

Risk The products of Domain Engineering will not lead to standardized domain reuse practices on projects.

Implication The Domain Engineering investment will not produce projected benefits for the targeted project or the business organization.

Mitigation

- Staff the Domain Engineering work with experienced project managers and engineers. Ensure that all work is actively reviewed by other experienced managers and engineers and is adequately reviewed by all participants of application engineering projects. Include engineers familiar with the needs of the targeted project.
- Evaluate the effectiveness of the Domain Engineering process and work products relative to past project experiences. Ensure that the characteristics of that experience or the resulting systems are not in conflict with the process and work products.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

None

4.2 Feedback From Product Consumers

Contingency The work product families are inadequate to support the needs of the targeted project.

Source Application Engineering

Response

- Determine that expressed needs are outside of or otherwise conflict with chosen domain objectives or cannot be viably satisfied given available domain resources.
- Evolve the domain to satisfy current needs.

This page intentionally left blank.

DE.1. DOMAIN MANAGEMENT ACTIVITY

1. GETTING STARTED

Domain Management is an activity of Domain Engineering for managing business-area resources to increase opportunities for cost-effective reuse of previously developed assets. Domain Management plans, assigns resources, and directs Domain Engineering to serve the needs of a targeted application engineering project.

Domain Engineering develops and evolves a domain through a series of increments. The Domain Plan lays out both a master plan for evolution through projected increments (evolution plan) and, as each increment is initiated, a detailed plan for each increment (increment plan). The evolution plan identifies the systems that provide the basis for a domain (i.e., the source of raw material) and the application engineering projects that are targeted for support. An increment plan determines how domain engineering resources are applied to create reusable assets that can be exploited effectively in the targeted application engineering project.

Domain Management monitors domain engineering performance to assess progress, ensure proper adherence to plans, and guide needed revisions to the evolution and increment plans based on feedback from Application Engineering use of domain assets. A key concern of Domain Management is coordinating Domain Engineering activities to support the needs and priorities of targeted application engineering projects in satisfying customers' needs. Domain Management attempts to ensure effective use of allocated resources by coordinating its planning to match the needs of a targeted application engineering project.

1.1 Objectives

The objective of Domain Management is to manage business-area resources to create cost-effective reuse opportunities for a targeted application engineering project. Management establishes domain objectives for the organization to guide the creation and revision of an increment plan for a series of domain increments. An increment begins when a project is first targeted for support or when the needs or status of the targeted project changes significantly.

For each increment of development, Domain Management develops a plan to deliver capabilities that match the needs of targeted application engineering projects. Application engineering projects are planned independently, but with an awareness of domain capabilities, to meet particular customer needs.

1.2 Required Information

The Domain Management Activity requires the following information:

- Business objectives, specifically the priorities of executive management for business area development
- Domain Definition: Domain Status

1.3 Required Knowledge and Experience

The Domain Management Activity requires domain and business-area knowledge and experience in:

- The nature of projects in the business area
- All aspects of strategic business-area management in the organization,
- All aspects of application project management in the organization

2. PRODUCT DESCRIPTION

Name Domain Plan

Purpose Define near-term objectives for a business area and organize and manage domain resources to achieve those objectives.

Content A Domain Plan consists of three parts:

- ***Domain Evolution Plan.*** The Domain Evolution Plan establishes a purpose and scope for near-term domain development.
- ***Practices and Procedures.*** Practices and Procedures prescribe the preferred practices and procedures that are to guide the proper performance of domain development.
- ***Domain Increment Plans.*** A Domain Increment Plan specifies how to organize and manage Domain Engineering resources to facilitate reuse of previously developed assets by the targeted application engineering project.

Both the Domain Evolution Plan and the Domain Increment Plans include the following:

- ***Risk Analysis.*** Identification of uncertainties in meeting allocated business (for the Domain Evolution Plan) or domain (for a Domain Increment Plan) objectives, assessment of the risks of failure, and identification of mitigation strategies.
- ***Objectives.*** The scope and focus of support to be provided for the domain or the increment of domain development, reflecting the needs and priorities of targeted application engineering projects. Scope is indicated by an identification of previously built systems upon which the domain will be based; focus is indicated by the choice of targeted

project. Objectives are divided into risk objectives and product objectives. Risk objectives attempt to mitigate risks identified in the risk analysis. Product objectives establish goals and success criteria for creating specific work products.

- *Schedule.* The allocation of domain resources to development increments that satisfy domain objectives or to activities within an increment that satisfy increment objectives. The schedule establishes specific milestones and success criteria for domain development increments or for the activities of a development increment.

- *Issues.* A description of issues that arise in performing the plan.

Form and Structure

To the extent possible, the form of a Domain Evolution Plan should be the form your organization currently uses: the Domain Evolution Plan should follow the form used for business planning; Practices and Procedures should follow the form used for standardizing the practices of application projects; and the Domain Increment Plans should follow the form used for application project planning.

Verification Criteria

The verification criteria for the Domain Evolution Plan are:

- The expected needs (i.e., opportunities for reuse) of planned projects in the business area are sufficient to compensate for projected costs of domain development.

- The Domain Evolution Plan gives the domain a viable near-term purpose that is consistent with the needs of targeted projects.

- Each Domain Increment Plan institutes a plan that seems likely to satisfy the expected needs of the targeted project.

3. PROCESS DESCRIPTION

The Domain Management Activity consists of three steps as shown in Figure DE.1-1.

The Domain Evolution step begins upon creation of a domain and continues until the domain is no longer judged to be viable in serving the needs of an application project within the business area. The Domain Evolution Plan prescribes a series of Domain Development increments. Each increment is planned and performed iteratively until its objectives in the Domain Evolution Plan are met. The Domain Evolution Plan is subject to revision after the completion of each increment to reflect progress or changing needs of targeted application engineering projects and their customers. The step to Institute Practices and Procedures occurs before the initiation of the first increment of Domain Development and is revisited as needed to update the Practices and Procedures to ensure an effective and efficient approach to Domain Engineering.

The plan is iterated whenever a targeted project is initiated, terminated, or has a significant change in its own plan. Plan iteration requires you to reconsider the scope and focus of support you are providing to projects.

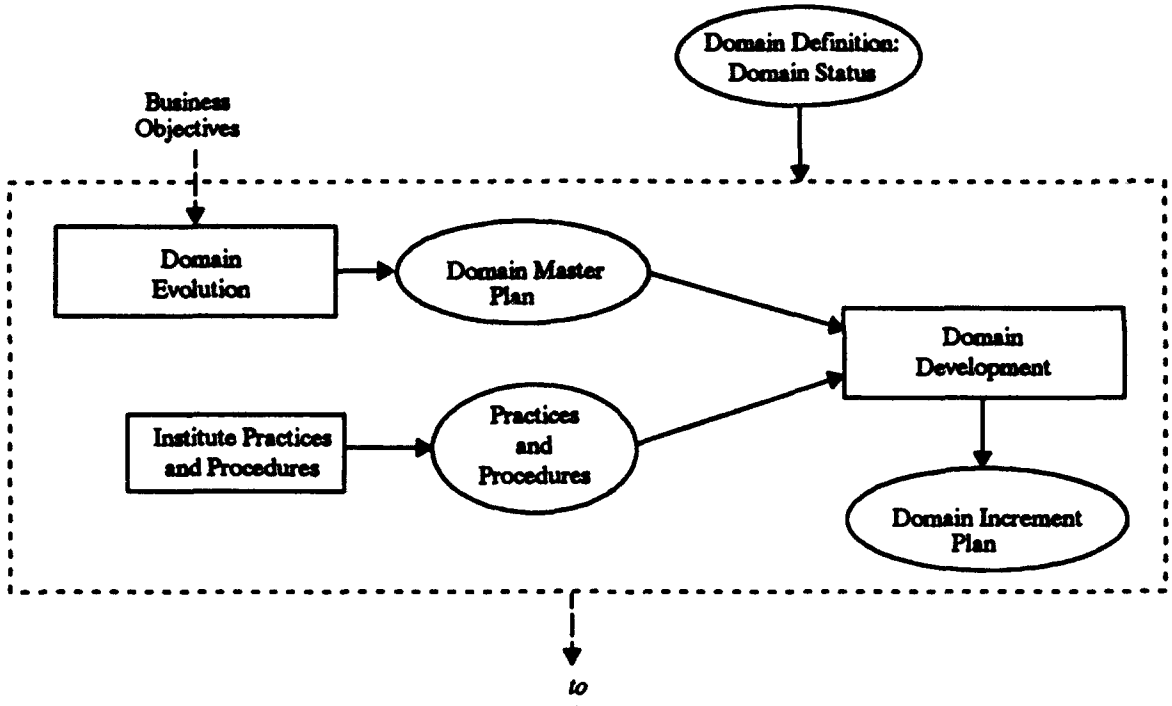


Figure DE.1-1. Domain Management Process

3.1 PROCEDURE

Follow these steps for the Domain Management Activity.

Step: Domain Evolution

Action	Create a plan for Domain Evolution.
Input	• Business objectives • Domain Definition: Domain Status
Result	Domain Plan: Domain Evolution Plan
Heuristics	• Develop a prioritized set of near-term domain objectives that will guide you in domain development. Develop a preliminary statement of domain objectives that identifies targeted projects and their needs. Refine the objectives to reflect the views of project managers, particularly their perceptions of their projects' risks and assets of greatest benefit to their projects. (Refer to the heuristics for the Domain Development step for suggestions on a risk-based management process that you can also apply in performing this step.) • Identify any previously built systems that are to be taken as characteristic instances of the domain and from which reusable assets are likely to be extracted.

- Try stating objectives in terms of commonalities and variations that characterize systems in the domain or that arise in the practice of Application Engineering.

Step: Institute Practices and Procedures

Action Develop and document standard practices and procedures to be followed in the activities of Domain Engineering.

Input None

Result Domain Plan: Practices and Procedures

Heuristics

- Prescribed practices and procedures should encompass administrative, software development (e.g., requirements and design methods, coding and documentation standards), project management and control, and quality assurance (e.g., testing, walkthrough, and review procedures).
- Configuration management procedures are a key element for controlling iterative domain development. Each iteration of domain development is represented by one version of each domain engineering work product that you produce. Feedback on the use of a product version leads to the creation of a new version in a later iteration of development.
- Consider how consistency and quality standards will be achieved in domain practices.

Step: Domain Development

Action Create a plan for developing a domain increment.

Input

- Domain Evolution Plan
- Practices and Procedures
- Domain Definition: Domain Status

Result Domain Increment Plan

Heuristics

- A Domain Development increment consists of repeated cycles through a process comprised of four steps as shown in Figure DE.1-2. This guidance assumes you are experienced in project management. Refer to the descriptions of the process model and activities for the ESP (Software Productivity Consortium 1992b) for a detailed project management method that you can follow to tailor and elaborate this process.
- The Domain Evolution Plan identifies the objectives to be met by the increment. Identify and rank the domain development risks faced by the organization in meeting these objectives.

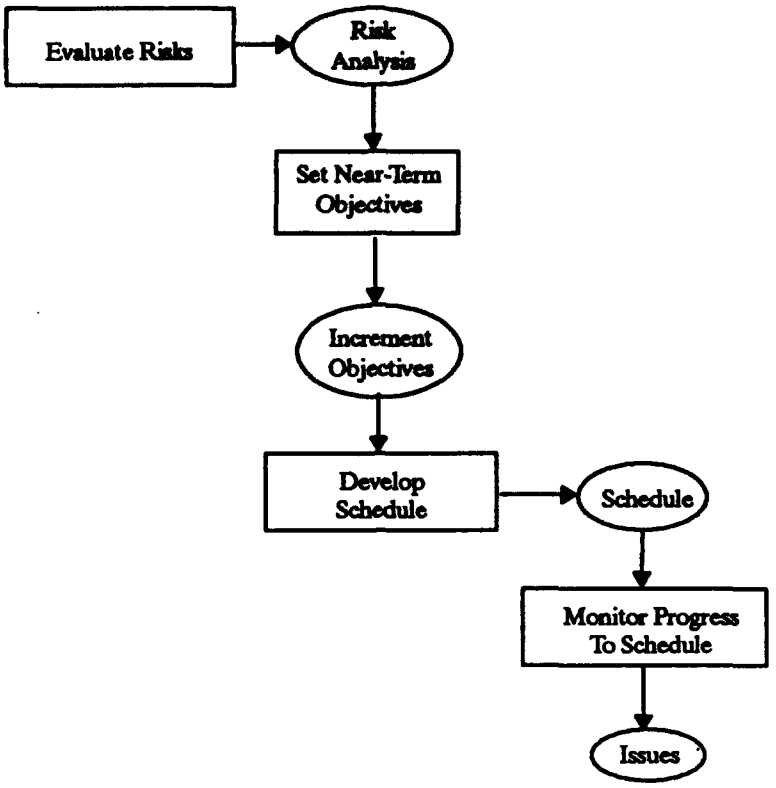


Figure DE.1-2. A Risk-Based Process for Increment Management

  - A recurring class of risks that domain development must face is the near-term ability of targeted application engineering projects to create required products. Actions to mitigate these risks take priority over other domain objectives.

  - Another important class of risks relates to problems discovered in previous iterations of domain development.

- Develop a prioritized set of near-term increment objectives that address the risks identified in the risk analysis. These objectives may be revised, as Domain Engineering iterates, to meet the Domain Evolution Plan objectives for the increment.

  - A domain-development approach must address the short-term needs of targeted projects.

  - Each objective should have associated success criteria that are used to determine whether the objective has been met. The success criteria should be written in such a way that they are directly measurable (if possible).

  - Objectives are often stated in terms of variations that characterize systems in the domain as represented by a set of work products or variations to be allowed in the practice of Application Engineering.

  - Set objectives for this increment of domain development that respond to both the risk and the intended result of domain objectives. Increment objectives should address the particular needs of targeted projects.

- Develop a schedule that allocates resources to tasks.

  - Create specific goals and completion criteria for each task. Each task is characterized by a Domain Engineering activity to be performed and completion criteria appropriate to that activity. The full set of tasks must address the near-term objectives within the resource budget provided. If the resource budget does not allow all the objectives to be addressed, the objectives with the highest priority should be addressed.

  - Plan for short iterations so that mistakes made in front-end tasks may be caught and corrected quickly in a subsequent iteration. Iterations at the beginning of the life cycle of a domain should be particularly short (three to four months) to compensate for the likely learning curve in domain concepts.

- Monitor domain engineering work progress to planned milestones and completion criteria.

  - The schedule establishes milestones and completion criteria for activities that are used to evaluate progress. Whenever new issues are identified or progress differs from that planned, evaluate whether to document your concerns for future planning or to revise the current plan for immediate action.

  - Document the source, implications, and possible and actual resolutions of each issue.

3.2 Risk Management

Risk	Domain plans will not be met within schedule with allocated resources.
Implication	Domain capabilities will fall short of plans.
Mitigation	• Review plans with experienced engineers to ensure that planned development is technically viable. • Reevaluate domain objectives to provide for shorter iterations that achieve essential capabilities sooner; defer work on less important objectives.
Risk	Domain engineers resist using standardized practices and procedures.
Implication	Inefficient operation and employee dissatisfaction will reduce productivity.

Mitigation

- Involve domain engineers in developing practices and procedures.
- Provide education and apprenticeships.
- Conduct pilot projects that emphasize learning new skills over product delivery.

Risk Domain engineers fail to recognize when to terminate an iteration.

Implication

- There will be excessive detail in products without adequate foundation or potential benefit.
- Schedules will slip.

Mitigation Review objectives and completion criteria to make sure they are specific and understood by the domain engineers.

Risk Project needs will not be met by planned development.

Implication Provided reusable assets will not have sufficient value to targeted projects to justify costs of the domain.

Mitigation Review objectives and plans with project managers to ensure that the needs of their projects and the projects' customers are properly understood.

Risk Domain plans will not be met within schedule with allocated resources.

Implication Domain capabilities will fall short of plans.

Mitigation

- Review plans with experienced engineers to ensure that planned development is technically viable.
- Reevaluate objectives and project needs to focus support on key needs of the targeted project first; defer work on less urgent objectives.
- Revise the Domain Increment Plan to add or defer activities, or to reallocate time and resources among planned activities, as priorities dictate.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition and/or Domain Specification fails to provide the needed capabilities required by the Domain Plan.

Source Domain Analysis Activity

Response Describe ways in which the Domain Definition and/or Domain Specification fail to provide the necessary capabilities. Modify schedule to allow completion of indicated Domain Definition or Domain Specification revisions.

4.2 Feedback From Product Consumers

Contingency	Project needs are not being met by the domain.
Source	Project Support Activity
Response	• Revise the Domain Plan to accommodate new needs.
	• Determine that the needs of a project are outside the proper boundaries of the domain.
Contingency	Practices and procedures are either ineffective or inefficient.
Source	• Domain Analysis Activity
	• Domain Implementation Activity
	• Project Support Activity
Response	Revise practices and procedures to reflect domain experience.
Contingency	The Domain Plan is too ambitious for available resources or expertise.
Source	• Domain Analysis Activity
	• Domain Implementation Activity
Response	• Allocate additional resources or time to domain development.
	• Refine the Domain Plan to reduce the scope.

This page intentionally left blank.

DE.2. DOMAIN ANALYSIS ACTIVITY

1. GETTING STARTED

Domain Analysis is an activity of Domain Engineering for studying and formalizing a business area as a domain. The purpose of formalizing a domain is to leverage knowledge of how recurring and varying problems from previous projects affect the form and content of application engineering work products for the targeted project. The scope of a domain is a decision based on existing systems and ongoing projects within the organization. Domain Analysis specifies a standardized Application Engineering process and work product families to support Application Engineering and verifies that a corresponding Domain Implementation meets that specification.

1.1 Objectives

The objectives of Domain Analysis are to:

- Determine scope and viability of a domain based on existing systems, planned domain development and evolution, and needs of the targeted project
- Establish, manage, and evolve a set of work product families representing domain knowledge perceived relevant to the targeted project
- Describe an Application Engineering process and work product families appropriate to the domain and relevant to the targeted project

1.2 Required Information

Domain Analysis requires the following information:

- Domain Plan: Domain Objectives
- Domain Implementation

1.3 Required Knowledge and Experience

Domain Analysis requires domain and software knowledge and experience in:

- Past and ongoing projects in the organization
- The needs that motivate systems in the domain

- The environments in which these systems operate
- How these systems are built

2. PRODUCT DESCRIPTION

Domain Analysis creates two work products: Domain Definition and Domain Specification.

2.1 Domain Definition

Purpose	A Domain Definition (see Section DE.2.1) is an informal description of the systems and related application engineering work products in a business area that form a domain. A Domain Definition characterizes how existing systems and systems being developed in ongoing projects in the domain are similar and how they differ.
Verification Criteria	The Domain Definition captures sufficient information to allow domain engineers to describe accurately any existing or potential system (in particular, the system being built by the targeted project).

2.2 Domain Specification

Purpose	A Domain Specification (see Section DE.2.2) characterizes work product families of the domain that are relevant to the targeted project and an Application Engineering process for constructing members of the respective work product families.
Verification Criteria	The Domain Specification precisely expresses the domain as captured in the Domain Definition.

3. PROCESS DESCRIPTION

The Domain Analysis Activity consists of the three steps shown in Figure DE.2-1.

3.1 Procedure

Follow these steps for the Domain Analysis Activity.

Step: Domain Definition Activity

Action	Characterize the domain to satisfy domain objectives relative to the targeted project's needs (see Section DE.2.1).
Input	Domain Plan
Result	Domain Definition
Heuristics	• Characterize the domain by defining its scope (i.e., classes of systems, characteristics, or functions included and excluded from the domain) and

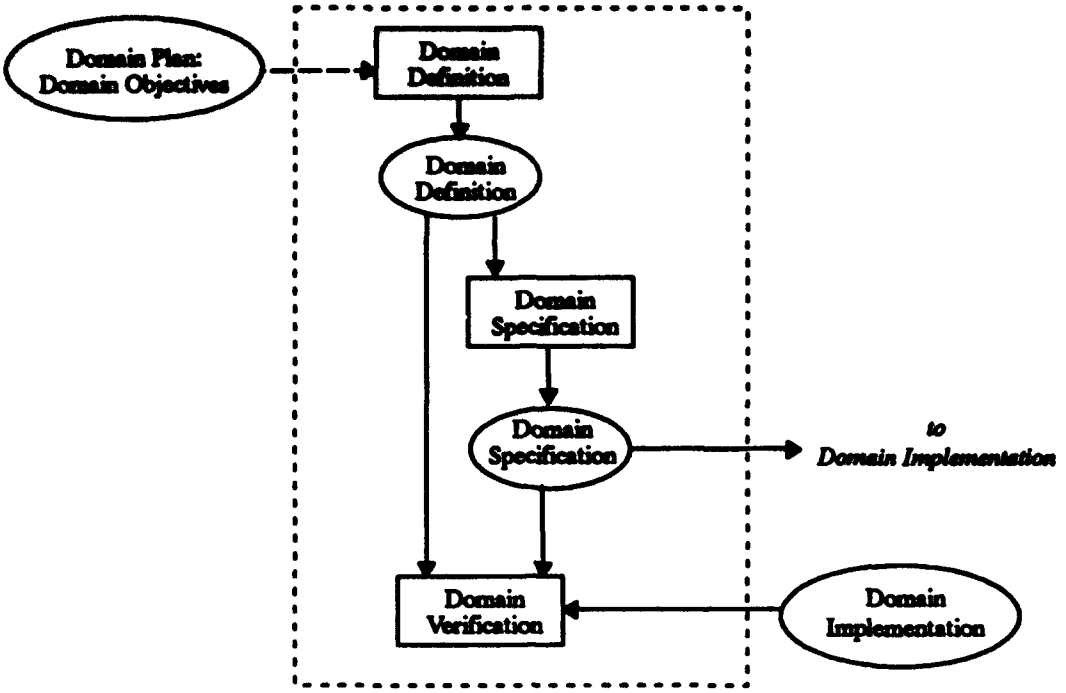


Figure DE.2-1. Domain Analysis Process

how included systems are distinguished from one another. These definitions are a basis for judging the qualitative and economic characteristics of the domain to determine whether the domain as defined will be economically viable.

- Consider how the work products for the system needed by the targeted project are similar and distinguishable from work products of existing systems.
- Use this definition as a basis for judging the qualitative and economic characteristics of the domain to determine whether the domain, as defined, will be economically viable for the targeted project. If analysis of the Domain Definition fails a test of economic viability for the targeted project, reevaluate the scope of the domain in terms of domain objectives relative to the targeted project's needs.

Step: Domain Specification Activity

Action	Specify Application Engineering Process Support (see Section DE.2.2).
Input	Domain Definition
Result	Domain Specification
Heuristics	• Identify and specify work product families appropriate for the domain and susceptible to reuse. You need to ensure that these families are useful when application engineers develop application engineering work products for the targeted project.

- Create a specification of standard support for Application Engineering in the domain. This support describes the process followed to identify work products that provide the focus of reuse efforts.
- Identify, for each work product family, the decisions that an application engineer must make to describe fully the variations in the different work product families. These decisions should accommodate aspects appropriate for the work product family (such as functional [e.g., behavioral] and nonfunctional aspects [e.g., size, timing, fault tolerance, hardware architecture, hardware/software configuration]) so that the application engineer can express his needs.
- Create standardized designs for the work product families. The designs must satisfy both the common and variable aspects of the work product family as perceived relevant to the targeted project. A standardized design includes both design structures that define various views of the work product structure and components from which a work product is constructed that might satisfy the application engineer's needs for the targeted project.

Step: Domain Verification Activity

Action Verify the correctness, consistency, and completeness of domain engineering work products (see Section DE.2.3 for motivation).

Input
- Domain Definition
- Domain Specification
- Domain Implementation

Result None

Heuristics
- Verify the consistency and completeness of the Domain Definition.
- Verify that the representation of the Application Engineering process in the Domain Specification is consistent and complete with respect to its representation in the Domain Definition.
- Verify that the representation of the application engineering product in the Domain Specification is consistent and complete with respect to its representation in the Domain Definition.
- Verify that the Product Implementation is consistent and complete with respect to the Domain Specification.
- Verify that the representation of the Application Engineering process in the Process Support is consistent and complete with respect to its representation in the Domain Specification.

3.2 Risk Management

Risk The cost of an increment of Domain Analysis is projected to exceed the budget.

Implication Insufficient resources exist to complete a planned iteration of Domain Engineering.

Mitigation

- Reduce the current scope.
- Seek a change in domain objectives or an increase in the budget for the increment from Domain Management.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Definition and a Domain Specification that satisfy the Domain Plan as closely as possible.

Contingency The Domain Implementation does not satisfy the Domain Specification.

Source Domain Implementation Activity

Response Clarify the intent of the Domain Specification.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Implementation Activity

Response

- Revise the Domain Specification.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency	The Domain Definition and/or Domain Specification fails to provide the capabilities required by the Domain Plan.
Source	Domain Management Activity
Response	Evolve the Domain Definition and the Domain Specification to be consistent with the Domain Plan.
Contingency	The Domain Specification is incomplete, ambiguous, or inconsistent.
Source	Domain Implementation Activity
Response	Revise the Domain Specification to correct the inadequacies.
Contingency	The standardized Application Engineering supporting work products are inefficient or lead to less-than-ideal results for the targeted project.
Source	Project Support Activity
Response	• Determine that the benefits of work product standardization outweigh the interests of the particular project. • Evolve the definition of the Application Engineering supporting work products to reflect the project's experience or to be adapted to the particular conditions of concern.

DE.2.1. DOMAIN DEFINITION ACTIVITY

1. GETTING STARTED

Domain Definition is an activity of Domain Analysis for creating a Domain Definition. A Domain Definition is an informal description of the systems and related application engineering work products in the business area that form a domain. A Domain Definition characterizes how systems in the domain, as represented by a set of work products, are similar and how they differ.

1.1 Objectives

The objectives of the Domain Definition Activity are to:

- Establish a conceptual basis and bounds for more detailed Domain Analysis
- Determine whether planned development of the domain is viable relative to the needs of the targeted project and the organization's reuse objectives
- Establish criteria by which management and engineers can judge whether a proposed system is properly within the domain

1.2 Required Information

The Domain Definition Activity requires the Domain Plan.

1.3 Required Knowledge and Experience

The Domain Definition should be developed by experts with a variety of backgrounds in the domain understudy. They need broad domain knowledge. Such knowledge includes what systems have been built, relevant existing work products (requirements specification documents, design documents, etc.), and the nature of systems likely to be requested and built, especially by the targeted project.

2. PRODUCT DESCRIPTION

Name	Domain Definition
Purpose	A Domain Definition establishes the scope of a domain and a justification of its economic viability. It provides a basis for determining, informally, whether a system is properly within that scope. The Domain Definition does not answer detailed questions of scope, but clearly includes and excludes broad classes of systems and their associated

work products. Assumptions of commonality and exclusion identify the common features of systems and work products in the domain, thereby establishing a family. Assumptions of variability identify how systems and work products in the family are distinguished from one another. Justification provides a basis for judging technical and economic feasibility of the domain to evaluate whether there is sufficient confidence in the viability of supporting work-product reuse by the targeted project for a system in the domain.

Content

A Domain Definition consists of the following components:

- ***Domain Synopsis.*** An informal statement of the scope of the domain.
- ***Domain Glossary.*** Definitions of significant terminology used by experts in discussing needs and solutions in the domain.
- ***Domain Assumptions.*** A description of what is common, variable, and excluded among systems in the domain, as reflected in their associated work products.
- ***Domain Status.*** An assessment of the current maturity and viability of the domain relative to its expected use by the targeted project.
- ***Legacy Products.*** A representative collection of work products from existing systems in the product line which may be a suitable source of information and raw material for developing the domain.

Verification Criteria

Every characteristic ascribed to all systems in the domain by the Domain Synopsis must be stated as a Commonality Assumption.

2.1 Domain Synopsis

Purpose

The Domain Synopsis is an informal statement of the scope of the domain. It characterizes systems included in the domain and work products your organization produces during software development.

Content

A Domain Synopsis includes an informal characterization of the systems and work products that make up the domain.

Form and Structure

A Domain Synopsis is a simple narrative using terms defined in the Domain Glossary. Example DE.2.1-1 illustrates a fragment of a Domain Synopsis for the TLC domain. This fragment depicts typical information contained in a Domain Synopsis and the use of terms from the Domain Glossary.

Verification Criteria

- The Domain Synopsis must give an intuitive feel for the definitive characteristics of systems in the domain, as reflected in their associated work products. It should, in itself, adequately describe any existing system (in particular, the system being built by the targeted project).
- A term that could have different meanings to different readers may be used in the Domain Synopsis only if it is defined in the Domain Glossary.

The Traffic Light Control Software System (TLC) domain is a family of embedded computer systems to control the operation of traffic lights at a given intersection. The TLC domain is limited to controlling traffic at intersections of two roads with at most one road dead-ending at the intersection. Systems in the TLC domain control traffic at the intersection by changing the indicators of each traffic light (called a traffic light sequence). Each traffic light sequence is coordinated with the other traffic lights in the intersection to prevent accidents while vehicles traverse the intersection. Usually, a TLC system generates its traffic light sequence based on a clock-generated cycle which may last from one (1) to three hundred (300) seconds. The traffic light sequence generated from the clock may be modified based on signals from optional input devices.

One optional input device is a trip mechanism buried under the roadway. A trip mechanism may be associated with either a left-turn lane, a right-turn lane or a thru-traffic lane. Input from a trip mechanism associated with a left-turn lane controls whether the left-turn indicator of the traffic light is turned on during a traffic light sequence. A trip mechanism associated with a right-turn signal may act in an analogous manner for a right-turn signal. Inputs from any trip mechanisms may alter the traffic light sequence. A trip mechanism is not required for the proper operation of the turn lane.

Another optional input device is the pedestrian crosswalk push button. This input controls whether the walk/don't walk indicator is on during a traffic light sequence. It may also modify the traffic light sequence. A push button is not required for the proper operation of the pedestrian lane.

....

Example DE.2.1-1. Fragment of TLC Domain Synopsis

2.2 Domain Glossary

Purpose

The Domain Glossary is a compendium of precise definitions for all significant terminology used by experts for discussing problems and solutions in a domain. This domain terminology is organized into a taxonomy of terms.

Content

A Domain Glossary has two parts:

- A set of standard terms and their definitions
- A list of references to external sources which define and elaborate on relevant topics and terminology

Form and Structure

A reference to an external source is written using an accepted documentation style for a reference (e.g., author-date). Standard terms are defined in alphabetical order using the following forms:

Term 1 definition (source)

Term 2 (1) first definition (source); (2) second definition (source)

The source of the term's definition (source) is listed after the definition. Example DE.2.1-2 illustrates a fragment of a Domain Glossary for the TLC domain. This fragment depicts typical terminology needed to discuss systems, problems, and solutions in the TLC domain.

Verification Criteria

- The Domain Glossary must contain precise definitions of all significant terminology used by domain experts for discussing the requirements or engineering of systems in the domain or their associated work products.

Term	Definition
Crosswalk	A specially paved or marked path for pedestrians crossing a street or road.[1]
Crosswalk Push Button	A monitoring device which allows a pedestrian to signal to the system his presence at a crosswalk.
Trip Mechanism	A traffic monitoring device used to determine whether a vehicle is present in a lane.
Traffic Control	A synchronized set of traffic light sequences specified as a function of time and traffic monitoring device inputs.
Traffic Light	A set of indicators placed at the intersection of streets to regulate traffic.
Traffic Light Cycle	One iteration through a traffic light sequence, i.e., from the display of the red indicator through to the next display of the red indicator.
Traffic Light Sequence	The order in which a set of traffic light indicators are displayed during a traffic cycle. Typically, this ordering is red, green, amber, but other orderings are possible.
Traffic Monitoring Device	A device that monitors the flow of traffic.
....	....

[1] Webster's Third New International Dictionary of the English Language Unabridged.

Example DE.2.1-2. Fragment of TLC Domain Glossary

- Any term used in a definition that could have different meanings to different readers must also be defined.
- All independently-used terms that are generalizations, specializations, or components of defined terms must also be explicitly defined.
- Terms defined in the Domain Glossary must be sufficient for a domain expert to give an accurate description of any existing system and the system being built by the targeted project.

2.3 Domain Assumptions

Purpose Domain Assumptions describe what is common to all systems or their associated work products in the domain and in what significant ways those systems and work products vary and can be distinguished. These assumptions determine, informally, whether a system, as represented by a set of work products, is within the scope of the domain.

Content There are three types of assumptions:

- ***Commonality Assumptions.*** A set of assumptions about the characteristics that are common to all systems in the domain and their associated work products (commonalities).

- ***Variability Assumptions.*** A set of assumptions about the characteristics that distinguish systems in the domain and their associated work products (variabilities).

- ***Exclusionary Assumptions.*** A set of assumptions about the characteristics of systems and their associated work products that are outside the scope of the domain (exclusions).

Every assumption is composed of a description and justification.

Assumptions may also be elaborated with associated, subordinate assumptions. For example, a commonality assumption may have specific variabilities associated with it. Similarly, a particular resolution of a variability assumption can be thought of as characterizing a subfamily of the product family. The subfamily then may have additional, more specific commonalities and variabilities that further distinguish the members of the subfamily.

Form and Structure

An assumption description and justification are informal text. Assumptions which elaborate another assumption should be presented in adjacent, indented text. Examples DE.2.1-3 and DE.2.1-4 illustrate fragments of some commonality and variability assumptions for the TLC domain. The justification provides rationale on why the domain engineers believe the assumption to be valid.

Verification Criteria

- Commonality and variability assumptions must capture all important aspects that are common to all systems in the domain, as represented by a set of work products, and the significant ways in which these systems and their work products can vary. Exclusionary assumptions must not exclude needed capabilities.

- Systems and their work products must only vary as implied by the variability assumptions.

- A commonality assumption must apply equally well, without qualification, to any system in the domain, as represented by a given type of work product. Systems, as represented by their associated work products, must not violate a stated commonality, either by excluding an included feature in the commonality assumptions or by including an excluded feature in the exclusionary assumptions.

- All reviewers must agree that domain experts will consider Domain Assumptions to be consistent and unambiguous, relative to the definitions in the Domain Glossary. A term that could have different meanings to different readers may be used in a Domain Assumption only if it is defined in the Domain Glossary.

....

- A TLC system controls the traffic light sequences at an intersection.

  Justification The purpose of a TLC system is to control when traffic light sequences change, and the interactions of the traffic lights at an intersection.

....

- A TLC system coordinates all traffic lights at an intersection.

  Justification Safe transit of intersection requires that streams of traffic not cross, e.g., east bound traffic and north bound traffic cannot have green indicators concurrently.

....

- The entire week is divided into traffic cycles. The traffic lights at the intersection are synchronized based on these traffic cycles.

  Justification Traffic patterns and loads vary over the course of a day and of a week. The timing of the traffic cycles must be varied to deal with the variations in the traffic load.

....

- A TLC system must process signals from a trip mechanism or push button within a specified time.

  Justification Smooth traffic flow depends upon the TLC system detecting and responding to requests to a traffic light sequence in a timely manner.

....

Example DE.2.1-3. Fragment of TLC Commonality Assumptions

2.4 Domain Status

Purpose Domain Status describes the current technical maturity of the domain that the organization has achieved relative to planned evolution, and assesses the viability of evolution. Of particular concern are unsupported variability assumptions (i.e., default commonalities).

Content The Domain Status is an informal characterization of the degree to which Domain Objectives are satisfied by past development. It includes an analysis of how well the needs of the targeted project are being met by the domain.

Form and Structure The maturity of a domain can be expressed as limitations in satisfying variability assumptions. Risks can be mitigated by imposing limits on variability assumptions.

2.5 Legacy Products

Purpose Legacy Products provide access to work products from existing systems that may be useful sources of information and raw material for developing the domain.

- An intersection may be formed from two through roads (an X intersection), or from one through road and a dead-ending road (a T intersection).

  Justification The number of roads that can meet at an intersection can vary from intersection to intersection. At least two roads must cross to have an intersection, but intersections of three or more roads are common. The marketing department believes that the vast majority of intersections are of the T and X variety. It has decided to concentrate on this large subset of traffic intersections.

....

- The number of lanes of traffic on any road approaching or leaving the intersection may vary. The minimum number of lanes of traffic is one (1). The maximum number is six (6).

  Justification Any road bringing traffic into the intersection must have at least one lane of traffic into the intersection. Any road leaving the intersection must have at least one lane of traffic from the intersection. Engineering has determined that the hardware components of the system cannot control more than six lanes of traffic in a timely fashion.

....

- Any road approaching the intersection may have a trip mechanism buried under the traffic lanes to alert the system to the presence of vehicular traffic. There may be no trip mechanism or there may be one (1) per traffic lane.

  Justification The traffic light sequence may take into account the presence or absence of vehicular traffic at the intersection.

....

- The duration of a traffic control schedule will vary.

  Justification Traffic patterns and volumes vary from intersection to intersection. Effective traffic control at these intersections requires schedules that take these variations into account. Traffic control schedules must also vary to account for differences in system configurations.

....

Example DE.2.1-4. Fragment of TLC Variability Assumptions

Content

Legacy Products consists of a representative collection of work products (or portions thereof) from existing systems in the product line to be supported by the domain.

Form and Structure

- Work products may be physically stored, on paper or in electronic media, or may only be identified by reference when sufficiently accessible in this way (e.g., in an organization's local library or in an accessible repository set up for another, existing domain).

- Work products are kept in Legacy Products in the form in which they were produced. Other, consuming activities of Domain Engineering will copy and excerpt or adapt these work products, as needed, in order to create reusable assets.

- The work products comprising the Legacy Products are organized in a suitable manner to provide access by other Domain Engineering activities to a particular system's work products or to individual work products of a particular type.

Verification Criteria Each work product in Legacy Products must come from an existing system that was determined to be in the domain.

3. PROCESS DESCRIPTION

The Domain Definition Activity consists of the five steps shown in Figure DE.2.1-1.

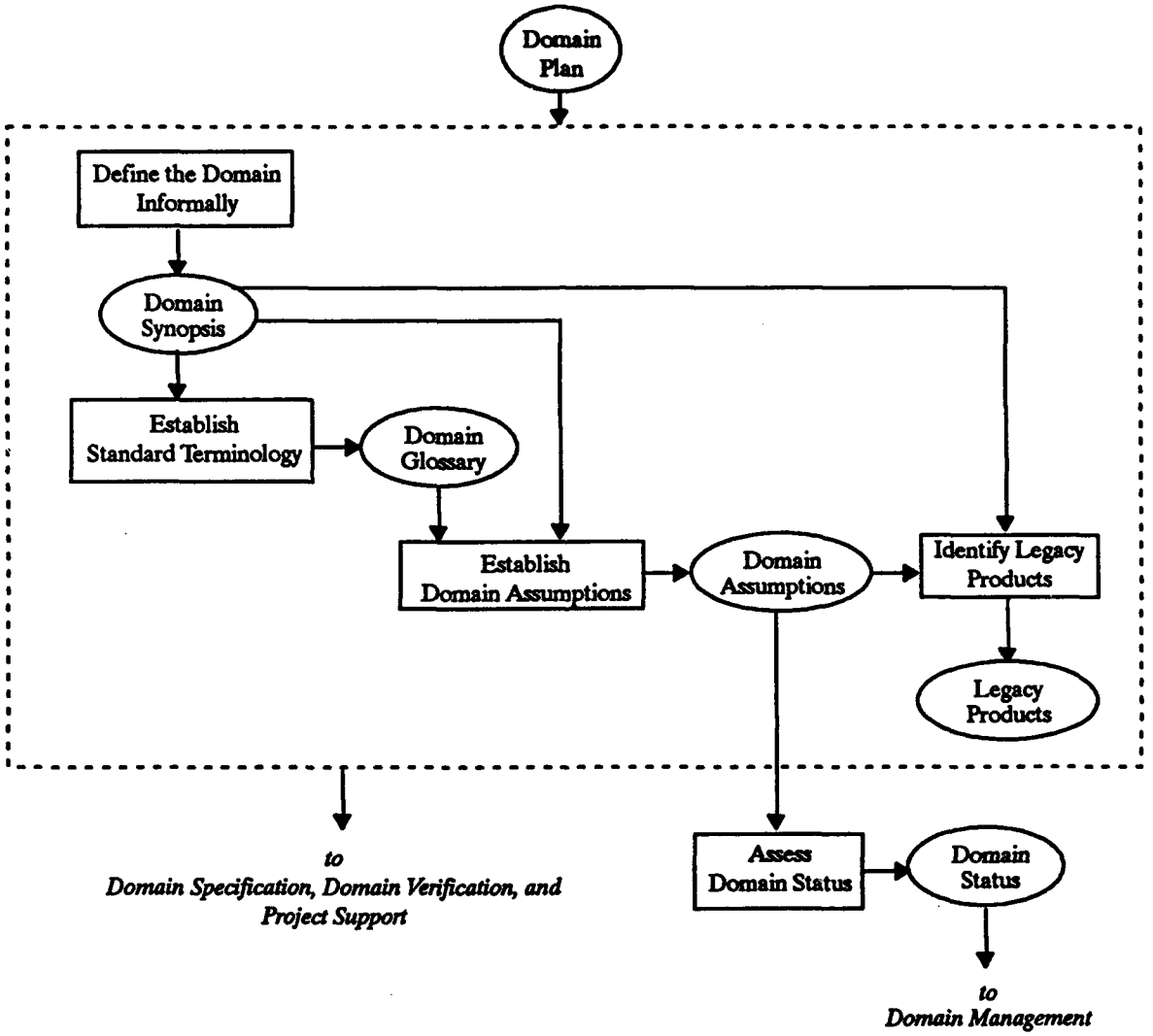


Figure DE.2.1-1. Domain Definition Process

3.1 PROCEDURE

Follow these steps for the Domain Definition Activity.

Step: Define the Domain Informally

Action Create a description of the domain, characterizing key technical objectives of included systems.

Input Domain Plan: Domain Objectives

Result Domain Definition: Domain Synopsis

Heuristics

- Start with a one-sentence description of the family of systems that constitutes the domain.
- Refine the Domain Synopsis to two pages, at most, of intuitive and not overly-restrictive text. Impart, concisely, an informal but complete sense of the domain in the first paragraph. Try to focus on the essential nature, scope, and variety of systems in the domain.
- Characterize the type of problem that systems in the domain solve, and the external environment (i.e., devices, systems, and users) with which systems interact. Describe the observable behavior that systems exhibit in solving the problem. You might also establish significant constraints concerning how the systems operate in terms of performance, reliability, or distribution concerns.
- Cover the primary functions performed by every system in the domain and any important functions performed by only some systems. Maintain a black-box perspective when describing functional aspects of the system.
- Be sure your descriptions of problems and functions characterize the system to be produced by the targeted project.
- Use terms defined in the Domain Glossary to keep the Domain Synopsis short.
- If the domain (e.g., process control systems) is based on formal theories that provide experts with a common language of communication about problems, refer to those theories in the Domain Synopsis.

Step: Establish Standard Terminology

Action Create definitions of all significant terms used by domain experts in discussing the requirements or engineering of systems in the domain or their associated work products.

Input Domain Definition: Domain Synopsis

Result Domain Definition: Domain Glossary

Heuristics

- Maintain term definitions in alphabetical order for ease of reference. Provide cross-references to related terms.

- Use definitions from standard glossaries where possible. Make note of such sources in each definition for future traceability.
- Make definitions as precise as possible.
- Make sure that all terminology used in the Domain Synopsis is defined in the Glossary.

Step: Establish Domain Assumptions

Action Create lists of the assumptions that allow you to think of the envisioned set of systems as a family and the assumptions that allow you then to distinguish among them and their associated work products.

Input
- Domain Definition: Domain Synopsis
- Domain Definition: Domain Glossary

Result Domain Definition: Domain Assumptions

Heuristics
- State only those assumptions that affect the system software and associated delivered products (e.g., documentation, test support).
- Initially, concentrate on assumptions related to system functionality. Expand your focus to design issues and to assumptions about related work products.
- Assumptions often come from knowledge or analyses of work products of Legacy Products. If some assumptions relate only to a particular type of work product, then group those assumptions apart from the generally more applicable assumptions.
- To create a preliminary set of assumptions:
  - Create a commonality assumption for each characteristic specified in the Domain Synopsis that is shared by all systems in the domain.
  - Create a variability assumption for each characteristic specified in the Domain Synopsis that is not shared by all systems in the domain.
  - For each term in the Domain Glossary, determine whether the term indicates a commonality or a variability among systems in the domain. Create an assumption accordingly.
- Make variability assumptions precise by indicating the type of decision the application engineer must make to resolve the variability. It is not sufficient to note only that some characteristic varies. You must establish how much flexibility the application engineer needs to characterize different systems adequately.

- Elaborate commonality assumptions to uncover specific variabilities assumptions associated with them. This will more precisely characterize a subset of the product family.
- Elaborate variability assumptions to find more specific commonality and subsequent variability assumptions that further distinguish members of the subfamily.
- Compare the characteristics of existing systems to facilitate the identification of common features and variations.
- Distinguish between system-generation-time and run-time variations when developing assumptions about variable aspects of the domain. Treat a run-time variation that is characteristic of all systems in the domain as a commonality.
- Use exclusionary assumptions to clarify a domain's boundary. Do not enumerate every type of system or function that is outside the domain. Rather, exclude explicitly those functions or characteristics that a domain expert might incorrectly assume to be part of a system when reading the Domain Synopsis. Thus, you can answer the question of whether a particular system belongs within the domain more directly by checking the exclusions. Exclusions often result from a viability analysis of the domain.
- State unresolved issues of functionality, design, etc., from the targeted project as variability assumptions. This will help you identify the full range of existing work products that might meet an anticipated Application Engineering need.

Step: Assess Domain Status

Action Evaluate the technical maturity of the domain in terms of targeted project needs, Domain Objectives, and plans for domain development.

Input
- Domain Plan
- Domain Definition: Domain Assumptions

Result Domain Definition: Domain Status

Heuristics Domain Status must result in an endorsement of and commitment to a specific domain scope (set of assumptions). This endorsement comes from targeted project needs, as tempered by the intuitions of experienced personnel, and resource constraints (e.g., what products are available). In other words, you should assess which Domain Assumptions are currently satisfied versus which should be satisfied. Of those which should be satisfied, assess which can be satisfied using existing resources and which can be satisfied using resources presumed to be available at some future date. This information should help you determine whether the features associated with each assumption are implementable, and the corresponding risk.

Step: Identify Legacy Products

Action	Identify existing systems in the product line that are considered representative of the domain and whose work products may prove useful as sources of information and raw materials in developing the domain.
Input	• Domain Synopsis • Domain Assumptions
Result	Domain Definition: Legacy Products
Heuristics	• Use the Domain Synopsis as a guide to select existing systems that are within the domain (or subsystems that would be parts of such systems). • Based on Domain Assumptions, identify work products (or fragments, if appropriate) from these systems that reasonably satisfy some or all of the Domain Assumptions. • Create a brief description of the selected systems and work products as a guide to their use as a source of information and raw materials by other Domain Engineering activities.

3.2 Risk Management

Risk	There is a lack of critical expertise.
Implication	The Domain Definition cannot be completed or there is unacceptably low confidence in the results.
Mitigation	• Commit time and resources to acquiring the expertise. • Restrict Domain Assumptions sufficiently to reduce the need for expertise.
Risk	The scope of the domain may be too narrow, precluding useful variations.
Implication	• Opportunities for additional projects are lost. • Application engineering projects miss opportunities for reuse.
Mitigation	• Review the Domain Definition with management and experienced engineers to identify additional variations. • Include unresolved issues from the targeted project.
Risk	The scope of the domain may be too broad.
Implication	Resources are misapplied to solve an unnecessarily general problem; the result will be parts that require so much hand-tailoring that the cost of reuse becomes higher than that of developing parts from scratch.

Mitigation

- Review the Domain Definition with management and experienced engineers to identify under-constrained Commonalities.
- Focus variations on the needs of the targeted project.

Risk Domain Assumptions are too precise or too vague.

Implication Flexibility is reduced unnecessarily, or key decisions are left to the discretion of domain engineers.

Mitigation Review the Domain Definition with management and experienced engineers to identify over- or under-constrained Domain Assumptions.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Definition that satisfies Domain Objectives as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency The Domain Definition fails to provide the capabilities required by the Domain Plan.

Source Domain Management Activity

Response Evolve the Domain Definition to be consistent with the Domain Plan.

Contingency The Domain Definition is incomplete, ambiguous, inconsistent, or incorrect.

Source

- Domain Specification Activity
- Domain Verification Activity

Response Revise the Domain Definition to correct the inadequacies.

This page intentionally left blank.

DE.2.2. DOMAIN SPECIFICATION ACTIVITY

1. GETTING STARTED

The Domain Specification Activity is an activity of Domain Analysis for creating a Domain Specification. A Domain Specification is a description of the Application Engineering process for a domain and a definition of a collection of work product families that support reuse within that process. The Application Engineering process used by projects in the domain determines the types of work products whose creation could be supported by reuse. For targeted application engineering projects, those work products that offer the greatest opportunities for reuse are designated for formulation as a family.

A description of a work product family consists of a description of how members of the family vary, an abstract of the content of its members, and a specification of the design (i.e., composition and structure) of its members. The descriptions of content and design must allow for the diversity among members indicated by the described variation in the family.

1.1 Objectives

The objectives of the Domain Specification Activity are to:

- Create a precise specification of the work product families that Domain Engineering provides to application engineering projects
- Identify a collection of work product families that will provide targeted application engineering projects with optimal reuse opportunities within their existing process

1.2 Required Information

The Domain Specification Activity requires the Domain Definition.

1.3 Required Knowledge and Experience

The Domain Specification Activity requires domain and software knowledge and experience in:

- Past and current systems in the domain
- The process that projects in the organization use to construct application engineering work projects
- Creating work products upon which reuse in the domain is to be based; expertise in what motivates differences in their form and content

- The concepts and structures that are convenient forms by which to communicate about the distinguishing features of work products in the domain

2. PRODUCT DESCRIPTION

Name Domain Specification

Purpose A Domain Specification is a description of the Application Engineering process for a domain and a set of work product families that support reuse in that domain. Domain Specification identifies a collection of work product families which the targeted project can reuse (through adaptation and tailoring) to produce individual work products that meet the needs of its particular customer.

Content A Domain Specification consists of one of each of the following components:

- ***Decision Model.*** A Decision Model identifies, for each work product family, the application engineering requirements and engineering decisions that determine how members of the work product family can vary (see Section DE.2.2.1). A Decision Model has one component for each supported work product family.

- ***Product Requirements.*** Product Requirements describe, for each work product family, the abstracted content of the members of the family (see Section DE.2.2.2). Product Requirements have one component for each supported work product family.

- ***Process Requirements.*** Process Requirements describe the established process of Application Engineering within a domain (see Section DE.2.2.3). It identifies the types of work products produced, and the types of work products for which families may be provided.

- ***Product Design.*** A Product Design determines, for each work product family, the structure and composition of solutions provided by members of the work product family (see Section DE.2.2.4). A Product Design has one component for each supported work product family.

Verification Criteria Problems and solutions typical both of existing application engineering work projects and of the needs of the targeted project are adequately addressed within the perspective of the Domain Specification.

3. PROCESS DESCRIPTION

The Domain Specification Activity consists of the five steps shown in Figure DE.2.2-1.

3.1 PROCEDURE

Follow these steps for the Domain Specification Activity.

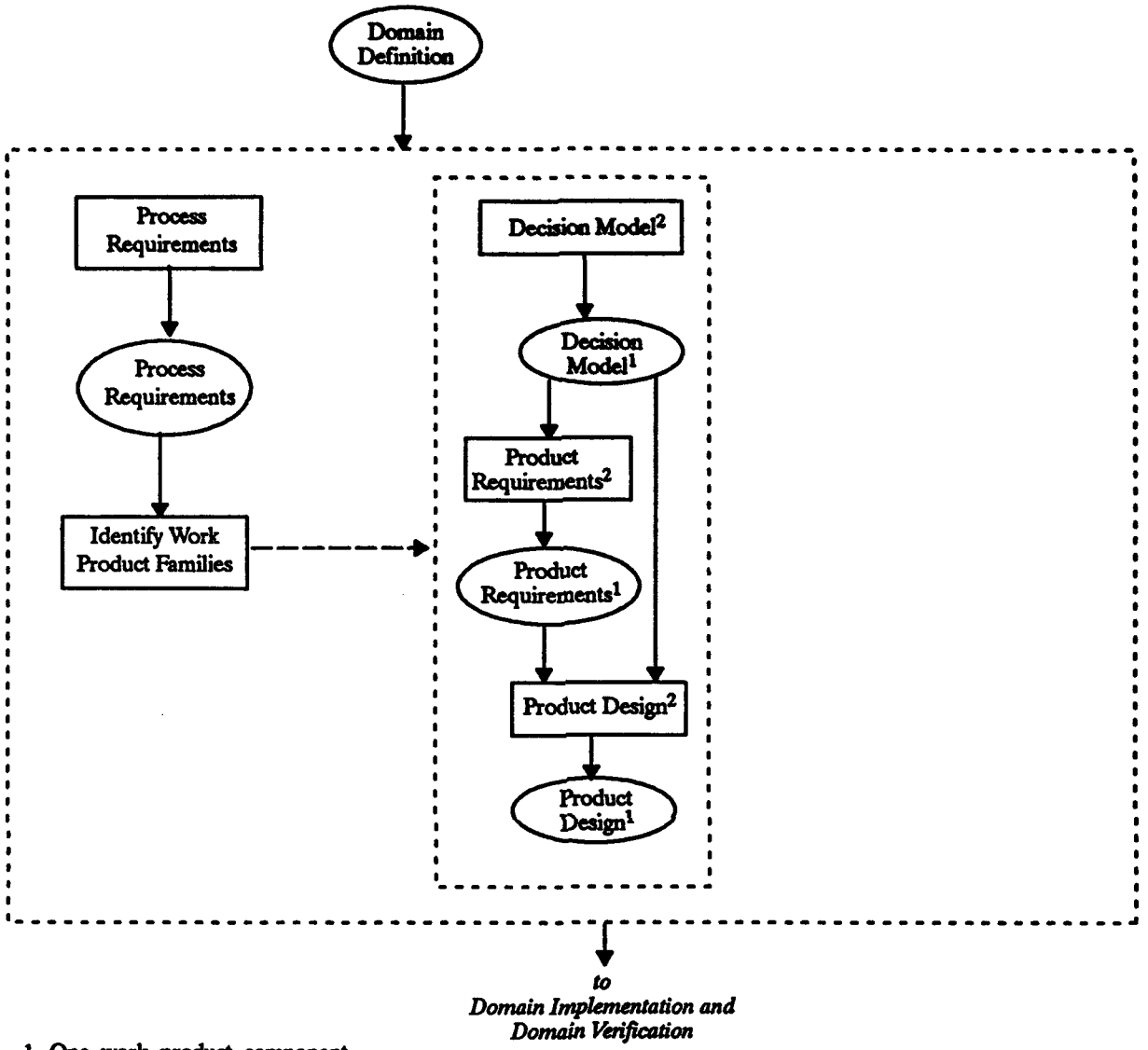


Figure DE.2.2-1. Domain Specification Process

Step: Process Requirements Activity

Action	Describe the process and work products of Application Engineering.
Input	Domain Definition
Result	Process Requirements
Heuristics	• Identify the types of work products, resulting from the Application Engineering process, that are normally used by projects in the domain.

Make allowance for any deviations known to be required by the targeted project.

- Determine how the process of work product development used by individual application engineers is to be modified to exploit reuse opportunities. Section AE provides a description of a typical process modified for reuse.

Step: Identify Work Product Families

Action Identify work product families targeted as the focus for increasing opportunities for reuse.

Input Process Requirements

Result None

Heuristics

- Identify those work product families that offer the best opportunities for reuse by the targeted application engineering project.
- A survey of existing work products is the primary basis for determining which types of work products should be supported with families. Focus the survey on what is available and what the targeted project will need.

Step: Decision Model Activity

Action Define the set of requirements and engineering decisions that an application engineer must resolve to select an instance from a designated work product family.

Input Domain Definition

Result Decision Model

Heuristics

- Perform this step for each designated work product family. Define the decisions that lead to expected differences among the members of the family.
- A Decision Model for a work product family should reflect the decisions that application engineers had to make when creating such work products in previous projects.
- The Decision Model for a work product family should reflect the variability assumptions from the Domain Definition that are relevant to this particular work product family.
- Ensure that supported decisions are sufficient to distinguish each existing work product from other members of the family.
- Identify logical relationships among the decisions that characterize a work product family and use them to structure the Decision Model. Such relationships can reduce the number and complexity of separate decisions that application engineers have to make.

Step: Product Requirements Activity

Action — Describe the content, in abstracted form, of the members of a designated work product family.

Input

- Domain Definition
- Decision Model

Result — Product Requirements

Heuristics

- Perform this step for each supported work product family. Describe the content of family members as briefly as possible without omitting key information.
- To the degree that application engineering decisions change work product content, describe how content varies with respect to those decisions. This description will provide a partial basis for explaining the meaning of decisions to application engineers.

Step: Product Design Activity

Action — Define the design (i.e., composition and structure) of the members of a designated work product family.

Input

- Decision Model
- Product Requirements
- Domain Definition: Legacy Products

Result — Product Design

Heuristics

- Perform this step for each supported work product family. Create a design for the work product family. An annotated outline is one model of a Product Design for a document work product family. An information hiding structure and process structure from the Ada-based Design for Real-Time Systems (ADARTS℠) (Software Productivity Consortium 1993) design method are models of a Product Design for a software work product family..
- A key element of domain knowledge is how existing instances of the designated work product family are designed. When feasible, derive the initial Product Design for a family by extracting the design essentials of existing instances. Ensure that the composition and structure of existing instances are appropriately reflected in the design.
- To the degree that Application Engineering decisions change work product composition and structure, describe how composition and structure vary with respect to those decisions. This description will provide a

further, but still partial, basis for explaining the meaning of decisions to application engineers.

3.2 Risk Management

Risk — The Domain Specification does not accommodate work products that meets the needs of the targeted application engineering project.

Implication — The domain will not provide sufficient opportunities for reuse by the targeted project.

Mitigation — Compare previously developed application engineering work projects that should be within supported families with expected needs of the targeted project. Check that likely differences are accommodated.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency — The Domain Definition is incomplete, ambiguous, or inconsistent.

Source — Domain Definition Activity

Response — Describe the inadequacies in the Domain Definition. Proceed with Domain Specification, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency — The Domain Plan cannot be satisfied with available technical capabilities.

Source — Domain Management Activity

Response — Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Specification that satisfies the Domain Plan as closely as possible.

Contingency — The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source — Domain Management Activity

Response — Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency — Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source	Domain Implementation Activity
Response	• Revise the Domain Specification. • Refer opportunities to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Domain Specification fails to provide the capabilities required by the Domain Plan.
Source	Domain Management Activity
Response	Evolve the Domain Specification to be consistent with the Domain Plan.
Contingency	The Domain Specification for a work product family is incomplete, ambiguous, inconsistent, or incorrect.
Source	• Domain Implementation Activity • Domain Verification Activity
Response	Refine the Domain Specification to correct any inadequacies.
Contingency	The standardized Application Engineering process for developing application engineering work projects from a supported work product family is inefficient or leads to less-than-ideal results for the targeted project.
Source	Project Support Activity
Response	Revise the Application Engineering process to reflect the problems encountered.
Contingency	Supported work product families are not useful for the targeted project.
Source	Project Support Activity
Response	• Determine that the nature of the problem and the consequent costs of upgrading the work product families outweigh expected benefits to the targeted project. • Evolve the domain engineering work product family to reflect this project's experience or to be more flexible under the particular conditions of concern.

This page intentionally left blank.

DE.2.2.1. DECISION MODEL ACTIVITY

1. GETTING STARTED

The Decision Model Activity is an activity of the Domain Specification Activity for producing a Decision Model. A Decision Model defines the set of requirements and engineering decisions that an application engineer must resolve to describe and construct a draft application engineering work product. A Decision Model is an elaboration of a domain's variability assumptions and is the abstract form (i.e., concepts and structures) of an Application Modeling Notation for a work product family. These decisions, and the logical relationships among them, determine the variety of work products in the domain. To construct a work product, these decisions must be sufficient to distinguish the work product from all other members of the family. The decisions establish how work products of application engineering, including software and documentation, can vary in form and content.

The Decision Model Activity is performed for each targeted work product family in the domain. Consequently, there will be a Decision Model (i.e., decisions and logical relationships among them) for each targeted work product family. The work product family's Decision Model is viewed as a partition of the domain's Decision Model. The focus of the interactions with other Synthesis activities occurs at the work product family's Decision Model.

1.1 Objectives

The Decision Model Activity defines a set of decisions that are adequate to distinguish among the members of an application engineering work product family and to guide adaptation of Adaptable Components that are composed to form application engineering work products.

1.2 Required Information

The Decision Model Activity requires the Domain Definition.

1.3 Required Knowledge and Experience

The Decision Model Activity requires domain and software knowledge and experience in:

- Conceptual modeling skills similar to those needed to create a database conceptual schema; see, for example, Kent (1978) and Borgida (1985)
- The issues that experienced engineers resolve when constructing systems in the domain

2. PRODUCT DESCRIPTION

Name Decision Model

Purpose

A Decision Model specifies the decisions that the Application Modeling Notation must allow an application engineer to make in describing an instance of a work product. These decisions determine the extent of variation in form and content that is possible in the work product belonging to the family.

To interpret fully the effects of decisions (i.e., to understand all properties of the family member identified by a set of decisions) requires both a Decision Model and a Product Requirements. The Decision Model specifies only the variations among members of a family. It does not specify their common properties. Product Requirements state the common properties, plus the effects of the decisions in a Decision Model.

Content

The Decision Model work product consists of three components:

- ***Decision Specifications.*** Specifications of the set of decisions that suffice to distinguish among members of a work product family.
- ***Decision Groups.*** A structuring of the decision specifications into logical groups, based on domain-related criteria.
- ***Decision Constraints.*** A set of constraints on the resolution of interdependent decisions.

Form and Structure

A Decision Model can be represented by one of the following forms:

- List of questions
- Tabular format

In the question-list format, each decision is phrased as a question and a non-empty set of valid answers. The question identifies the decision that an application engineer must make. The set states all permissible answers to that question.

In the tabular format, each horizontal row in the table expresses a decision specification. The horizontal row is divided into columns. A column identifies either the decision that an application engineer must make, the permissible answers for that decision, or a brief description of the decision.

Each decision and each decision group must have a unique identifier. Domain engineers use this identifier when they define adaptable work products. Each decision group has one list or table that is labeled with a mnemonic appropriate to the group. The group is a set of related decisions. Each entry is an independent decision that has its own distinct mnemonic label, a specification of allowed values that can resolve the decision, and a short explanation of the meaning of the decision.

If a set of related decisions is always resolved as a unit, you can define the set to be a composite decision. Composite decisions are shown in tabular form using a combination of the composite of indicator and indentation. If the application

engineer can choose to resolve one (and only one) decision from a set of alternatives, you can define the set to be an alternative decision. Alternative decisions are shown in tabular form using a combination of the alternative of indicator and indentation.

You can also use a tabular format to specify constraints on decision making. Decision constraints may be either structural or dependency. In both cases, a decision group (the Decision Group column) is specified as the focus of the decision constraint. A structural constraint is a decision constraint that limits the number of instances of a decision group in an Application Model. Valid entries include exactly-one, one-or-more, zero-or-one, zero-or-more, and one-for-each X, where X corresponds to other identified decision groups. A dependency constraint is a decision constraint that specifies how decisions made by an application engineer affect subsequent decisions.

Example DE.2.2.1-1 illustrates a fragment of a Decision Model for a work product family of the TLC domain. The figure portrays decision groups (e.g., Street, Lane_Group) and their corresponding decisions, along with appropriate constraints.

TLC_SSS: composed of	
Geometry: one of	{intersection geometry}
X: list length 4 of Street	{street characteristics for an X intersection}
T: list length 3 of Street	{street characteristics for a T intersection}
HW_Platform: composed of	
Platform: one of (TLC1, TLC2, TLC3)	{hardware platform the software will execute on}
Interface: one of (TL-A, TL-B)	{type of interface to traffic light indicators}
....	
Street: composed of	
Through_Lanes: Lane_Group	{characteristics for the thru lanes}
Left_Turn_Lanes: Lane_Group	{characteristics for the left-hand turn lanes}
Right_Turn_Lanes: Lane_Group	{characteristics for the right-hand turn lanes}
Pedestrian_Lane: PL_Group	{characteristics of a pedestrian lane}
Lane_Group: composed of	
Trip_Mechanism: one of (yes, no)	{designates whether the lanes have a trip mechanism}
Turn_Light: one of (yes, no)	{designates whether the lanes have a separate turn light}
....	
PL_Group: composed of	
Push_Button_Mechanism: one of (yes, no)	{designates whether the pedestrian lanes have a push button mechanism}
....	
Constraints	
– A Through_Lanes group must be specified for each Street.	
....	

Example DE.2.2.1-1. Fragment of TLC Decision Model for the System/Segment Work Product Family

Verification Criteria

- Every decision must be an elaboration of one or more variability assumptions or reflect variations that are characteristic of existing work product family members.
- All Domain Assumptions pertinent to the work product family must be elaborated in at least one decision.

3. PROCESS DESCRIPTION

The Decision Model Activity consists of three steps shown in Figure DE.2.2.1-1.

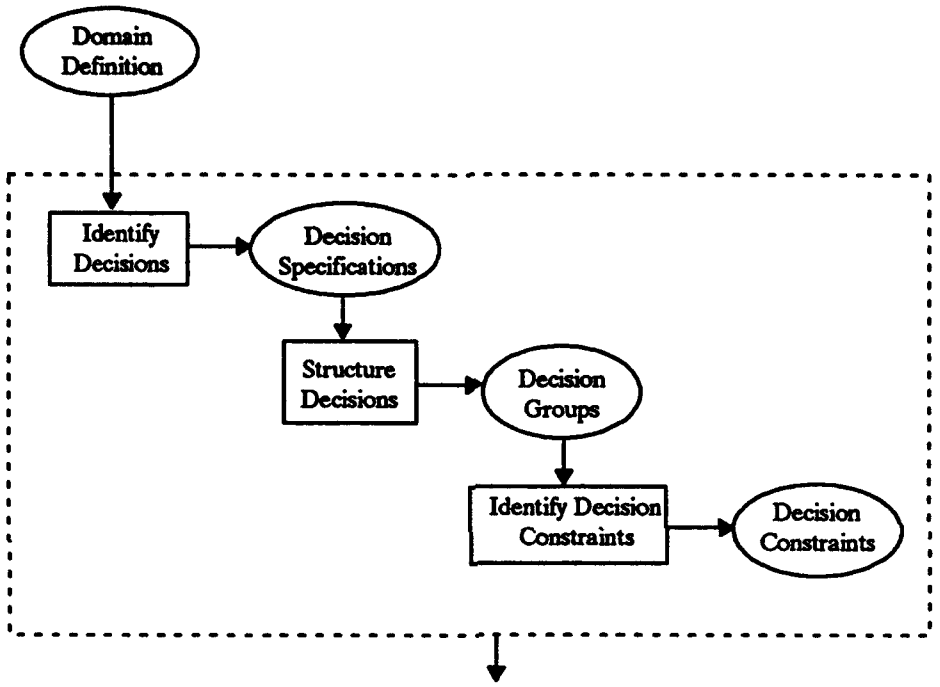


Figure DE.2.2.1-1. Decision Model Process

3.1 PROCEDURE

Follow these steps for Decision Model Activity.

Step: Identify Decisions

Action Identify the decisions that application engineers can make to resolve all of the variations for a work product of a system in the domain.

Input Variability assumptions

Result Decision specifications

Heuristics

- Derive decisions directly from variability assumptions. You will likely derive multiple decisions from a single variability assumption; each decision is an elaboration of some aspect of the basic variability.

- Derive decisions by examining relevant work products of existing systems. Even when you derive decisions from variability assumptions, you should examine these work products to help you understand how to express decision specifications in a form that is natural to the domain (e.g., data types, units).

- When you derive decisions by examining existing work products, use domain concepts to express the value space of the answers. This will help application engineers resolve decisions. In other words, if you have three work products that are members of the same family, identify values for answers that suggest differences among the work products.

- Keep in mind that the relevant decisions are those concerning system generation time, rather than run-time variation. If you followed a similar heuristic in identifying Domain Assumptions, run-time decisions should not be an issue here. Your focus now should be on how members of a work product family differ, rather than on ways in which a member varies its behavior at run-time. However, if members of a work product family have variable run-time behavior, then a valid decision may concern whether or how a particular member varies its behavior.

- If a variability assumption asserts that a certain characteristic of systems in the domain is variable without saying exactly how it varies, you must determine exactly how the characteristic can vary with respect to the work product family you are analyzing. Specify the precise type of information that will resolve a decision.

- Avoid routinely providing decisions that dictate arbitrary implementation limits (e.g., maximum number of users) unless those limits reflect a policy decision. Optimization of a system requires adequate flexibility.

- Create a separate Decision Model for each application engineering work product family. However, you will probably want to share decision specifications across the models, especially for work product families of the same types (e.g., families of code).

- When you examine existing work products, you sometimes gain a fuller understanding of differences by analyzing and comparing their structure. Structural analysis is part of the Product Architecture Activity. Verify the completeness of your Decision Model using the knowledge gained from performing the Product Architecture Activity.

- Derive at least one decision from every variability assumption that the Domain Plan says you are to support. You need not elaborate the answers for a decision if you do not fully understand its range. Instead, you can choose an answer that arbitrarily fixes the decision. The Decision Model, therefore, contains the normal set of decisions an application engineer expects to make when building a work product in the domain. Thus, he can follow a familiar decision-making process.

Step: Structure Decisions

Action Organize decisions into logically-related and interconnected groups.

Input Decision specifications

Result Decision groups

Heuristics

- Each decision group should represent a coherent and cohesive concept to domain experts. Such concepts usually have recognizable names. A concept may be independent of other concepts, or may be an aggregate concept that unifies other simpler concepts. In other words, a decision group may include both individual decisions and decision groups as elements.
- Structure the set of decisions based on the principle of separation of concerns (Dijkstra, Dahl, and Hoare, eds. 1972). For example, create a decision group for decisions that correspond to features of a single, physically-distinct entity.
- Group together mutually-dependent decisions, i.e., those that are unlikely to change independently. Domain experts often rely on a single concept that ties dependent decisions together.
- Group together decisions that repeat. For example, if you need to describe multiple types of a particular device, the engineer may make similar decisions for each type. You can group these decisions to create a single concept as a focus for decisions.
- Group together decisions if they are derived either from a corresponding single variability assumption or from separate assumptions that were grouped in the Domain Definition. A single assumption that motivates several decisions often represents a single concept, while assumption groupings often suggest how domain experts organize their thoughts about such systems.
- The principles of database schema normalization form a valid model for this step. As is the case with normalization, the goal here is to identify and organize a set of concepts without redundancy or inconsistency.
- Define explicit logical connections between the decision groups. These define the relationships between the decision groups.

Step: Identify Decision Constraints

Action Define structural and dependency constraints that limit how decisions are resolved.

Input Decision groups

Result Decision Model

Heuristics

- Define a structural constraint for each decision group; specify limits on when the group can validly occur in an Application Model.
- Define a dependency constraint whenever one decision narrows the resolution that the application engineer can provide for another decision.
- You may sometimes create decision groups where the cross-product of the decision specifications implies family members that do not exist. You should examine existing work products and specify constraints that omit these members from the Decision Model.

3.2 Risk Management

Risk The Decision Model is inadequate for descriptions of existing application engineering work products.

Implication The domain will not provide effective support for the targeted project.

Mitigation Try to describe one or more existing work products in terms of the Decision Model. Review these descriptions with experienced engineers from the targeted project to identify erroneous assumptions or unacceptable limitations.

Risk The decision space is too large or complex.

Implication Effort required to develop the Decision Model and subsequent adaptable work products will exceed a reasonable level.

Mitigation

- Focus on a set of well-understood decisions and make the assumption, explicitly, that the other decisions have fixed values (i.e., temporarily constrain them to be commonalities). Plan to relax these assumptions in subsequent iterations, or, in extreme cases, suggest that the Domain Definition Activity consider narrowing the domain scope.
- Reorganize the decision space to achieve a more effective separation of concerns.
- Introduce an indexing scheme into the decision groups (or use a more sophisticated one).

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity

Response Describe the inadequacies in the Domain Definition and suggest appropriate refinements. Proceed with Decision Model, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency	**The Domain Plan cannot be satisfied with available technical capabilities.**
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Decision Model that satisfies the Domain Plan as closely as possible.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency	The Decision Model is incomplete, ambiguous, or inconsistent.
Source	• Product Requirements Activity • Process Requirements Activity • Product Design Activity • Product Implementation Activity
Response	Refine the Decision Model to correct inadequacies.

DE.2.2.2. PRODUCT REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Product Requirements Activity is an activity of the Domain Specification Activity for creating Product Requirements. A requirements specification describes needs that are satisfied by creating a work product. Similarly, Product Requirements is a requirements specification that is adaptable to the decisions supported by the work product family's Decision Model. The Product Requirements describes the set of problems solved by the members of a work product family. By applying the decisions that characterize a particular work product (i.e., its Application Model) to the Product Requirements, a standardized description of that work product is produced. A Product Requirements gives meaning to an Application Model as a description of a member of a work product family. The Product Requirements Activity is performed for each work product family in the domain.

1.1 Objectives

The objective of the Product Requirements Activity is to define the requirements for a work product family described in Process Requirements. The specification must be adaptable to decisions allowed by the work product family's Decision Model.

1.2 Required Information

The Product Requirements Activity requires the following information:

- Domain Definition
- Decision Model

1.3 Required Knowledge and Experience

The Product Requirements Activity requires domain and software knowledge and experience in:

- The nature, purpose, and use of work products for existing applications
- The issues that application engineers must resolve in creating work products in the domain
- The principles and use of an appropriate specification method (e.g., informal, structured, semi-formal, or formal)

2. PRODUCT DESCRIPTION

Name Product Requirements

Purpose

Product Requirements specify the requirements of members of a work product family. Product Requirements also define the meaning of an Application Model created in accordance with the corresponding work product family's Decision Model. You can use the Product Requirements to understand (and explain to application engineers) the implications of decisions in an Application Model (which describes the problem solved by a work product).

Content

The Product Requirements is an adaptable requirements specification for a work product family. A specification contains four types of information:

- ***Concept.*** An overall characterization of purpose and objectives.
- ***Context.*** A characterization of the relevant environment and relationships within it.
- ***Content.*** A characterization of the expressed or contained substance, meaning or behavior, and scope.
- ***Constraints.*** A characterization of limits and demands on context of use or content.

As a whole, this information is sufficient to characterize each particular member of a work product family as implied by the decisions allowed by the family's Decision Model.

Form and Structure

Product Requirements may be expressed in any well-defined form, for example:

- Structured, informal text
- Assertions
- A formal or semi-formal specification

The assertions form of Product Requirements is a set of assertions that describe the (black-box) meaning of work products in the domain. Assertions may be simple or parameterized to reflect decisions defined in the Decision Model. Assertions can be structured into a hierarchy to facilitate separation of concerns.

Appropriate formal or semi-formal notations depend on the domain being analyzed, the application engineering work products mandated by Process Requirements, and the existing work products available for analysis. For example, when writing the Product Requirements for a family of code, you might want to represent the Product Requirements using a package interface specification like that of Booch (1987). To define behavior precisely, you might choose a semi-formal method as described in Heninger (1980).

For all forms, parameterization can be used to express the effects of decisions on Product Requirements. A metaprogramming notation can describe text

substitution, conditional inclusion, and iteration over repetitive decisions. Example DE.2.2.2-1 illustrates a fragment of a Product Requirements for a DOD-STD-2167A System/Segment Specification document work product family of the TLC domain. This fragment depicts a portion of the concept (e.g., the objectives) and content (e.g., topics) covered in this document. This fragment also depicts the use of parameterization (in terms of appropriate decisions from the work product family's Decision Model shown in Example DE.2.2.1-1) to express requirements that characterize particular members of the work product family. For example, the block of text describing the pedestrian lane push-button device topic is only included in the Product Requirements when there is at least one Street in the TLC system which has that device.

1. Overview

The objectives of a System/Segment Specification (SSS) within the TLC domain are as follows:

- The SSS specifies the requirements for a system or a segment of a system within the domain.
- The SSS provides a general overview of the TLC system or segment.

....

2. Key Topics and/or Functions

The basic requirements for an SSS are stated in the DID DI-CMAN-80008A. The members of the SSS work product family of the TLC domain vary from or interpret this specification as indicated below.

....

<if there is least one Street S that has S.Pedestrian_Lane.Push_Button_Mechanism = yes then>
The SSS shall identify the push-button device as an external to which the TLC system must interface. It shall define in detail the specific interface to the push button from the TLC system based upon the **<TLC_SSS.HW_Platform.Interface>** interface.
<endif>

....

<if TLC_SSS.HW_Platform.Platform is TLC1 then>
The SSS shall contain the subparagraph titled *Toxic products and formulations* (numbered 3.3.1.1) specifying the requirements for the control of the toxic products used in the manufacture of the **<Platform>** platforms.
<endif>

....

<if TLC_SSS.HW_Platform.Platform is TLC2 or TLC3 then>
The SSS shall omit the subparagraph titled *Toxic products and formulations* (numbered 3.3.1.1) with the statement "This subparagraph is not applicable to this system."
<endif>

....

Example DE.2.2.2-1. Fragment of TLC Product Requirements for the System/Segment Specification Work Product Family

Verification Criteria

- All implicit requirements must be an elaboration of one or more commonality assumptions.
- The Product Requirements must elaborate all commonality assumptions that are applicable to the work product family.
- If decisions that characterize a particular work product are applied to the Product Requirements, the result should be a correct description of that work product.
- One adaptation of the Product Requirements must describe a work product that is relevant to the targeted project.

3. PROCESS DESCRIPTION

The Product Requirements Activity consists of four steps shown in Figure DE.2.2.2-1.

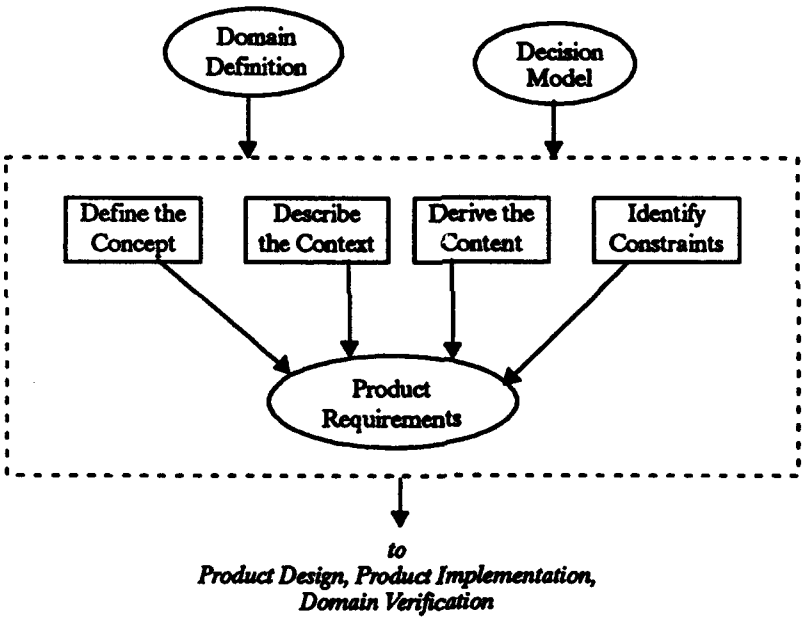


Figure DE.2.2.2-1. Product Requirements Process

3.1 PROCEDURE

Follow these steps for the Product Requirements Activity.

Step: Define the Concept

Action Describe overall purpose and objectives for the work product family.

Input
- Domain Definition
- Decision Model

Result Product Requirements: Concept

Heuristics

- Select a requirements method that best supports an abstract description of the work product family. For documents, a brief textual description may suffice.
- Describe the work product's objectives and provide an overview of its content.
- Examine commonality assumptions to identify additional aspects of concepts that apply to all members of the work product family.
- Examine variability assumptions to derive additional aspects of concepts that distinguish particular members of the work product family. Capture these requirements by parameterizing concept descriptions in terms of the appropriate decisions from the work product family's Decision Model.
- Examine Legacy Products from the Domain Definition to derive additional concept requirements that apply to all or some members of the work product family. Describe variations in concepts in terms of decisions in the work product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Describe the Context

Action Describe the relevant environment and relationships for the work product family.

Input

- Domain Definition
- Decision Model

Result Product Requirements: Context

Heuristics

- Describe the work product's audience, its expected benefits, and its relation to other work products.
- Examine commonality assumptions to derive additional context requirements that apply to all members of the work product family.
- Examine variability assumptions to derive additional context requirements that characterize a particular member of the work product family. Capture these requirements by parameterizing context descriptions in terms of the appropriate decisions from the work product family's Decision Model.
- Examine Legacy Products to derive additional context requirements that apply to all or some members of the work product family. Describe variations in context in terms of decisions in the work product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Derive the Content

Action Describe the subject matter covered in the work product family.

Input
- Domain Definition
- Decision Model

Result Product Requirements: Content

Heuristics
- Describe the topics covered by the work product.
- Examine commonality assumptions to derive requirements that apply to all members of the work product family.
- Examine variability assumptions to derive additional requirements that characterize particular members of the work product family. Capture these requirements in the Product Requirements by parameterizing content descriptions in terms of the appropriate decisions from the work product family's Decision Model.
- Examine Legacy Products of the Domain Definition to identify and extract additional common and varying requirements for content. Describe variations in content in terms of decisions in the work product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Identify Constraints

Action Describe limits and demands on members of the work product family.

Input
- Domain Definition
- Decision Model

Result Product Requirements: Constraints

Heuristics
- Describe any formatting guidelines or other restrictions on the work product.
- Examine commonality assumptions to derive additional constraints that apply to all members of the work product family.
- Examine variability assumptions to derive additional constraints that characterize particular members of the work product family. Capture these requirements by parameterizing constraint descriptions in terms of the appropriate decisions from the work product family's Decision Model.
- Examine Legacy Products to derive additional constraints that apply to all or some members of the work product family. Describe variations in

constraints in terms of decisions in the work product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

3.2 Risk Management

Risk	Product Requirements do not capture all Domain Assumptions accurately.
Implication	A derived requirements specification will not accurately describe the problem that the corresponding work product family member solves.
Mitigation	Create an Application Model for one or more existing work products and derive their respective requirements specifications. Review the specification with customers, experienced engineers, and domain experts to identify any inaccuracies.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Definition is incomplete, ambiguous, or inconsistent.
Source	Domain Definition Activity
Response	Describe the inadequacies in the Domain Definition. Proceed with Product Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete Product Requirements that satisfy the Domain Plan as closely as possible.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures that will make them more effective.
Contingency	The Decision Model is incomplete, ambiguous, or inconsistent.
Source	Decision Model Activity
Response	Describe the inadequacies in the Decision Model. Proceed with Product Requirements, and document any assumptions made regarding the inadequate portions of the Decision Model.

4.2 Feedback From Product Consumers

Contingency	Product Requirements fail to describe a work product family that is consistent with the Domain Definition.
Source	Domain Management Activity
Response	Modify the Product Requirements to be consistent with the Domain Definition.
Contingency	Product Requirements are incomplete, ambiguous, or inconsistent.
Source	• Product Design Activity • Product Implementation Activity
Response	Refine the Product Requirements to correct inadequacies.

DE.2.2.3. PROCESS REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Process Requirements Activity is an activity of the Domain Specification Activity for creating Process Requirements. Process Requirements is a description of the Application Engineering process currently in use by projects in your organization (including the targeted project). The description of the process identifies the activities, work products, and common methods or practices used by those projects to produce software work products. In the Process Requirements Activity, you determine which types of work products are produced, what steps engineers take to produce them, and how those steps can be modified to allow for reuse.

In an opportunistic Synthesis process such as this one, the goal of the Process Requirements Activity is primarily to document the actual process in use by targeted projects. Any changes to that process intended to facilitate reuse are minimized, with an ideal of not changing major activities, relationships, or work products. Changes are limited to the way in which individual work products are produced, with the intent of aiding the efficient discovery and exploitation of opportunities for cost-effective reuse.

1.1 Objectives

The objective of the Process Requirements Activity is to characterize the prevalent Application Engineering process used by projects in a domain and the work products that result. It is assumed that the targeted project will follow a similar process and produce work products of the same types. Based on that characterization, the Process Requirements Activity identifies which work products the targeted project can likely produce more easily if reusable assets are available.

1.2 Required Information

The Process Requirements Activity requires the Domain Definition.

1.3 Required Knowledge and Experience

The Process Requirements Activity requires domain and software knowledge and experience in:

- The conventional life-cycle process of systems in the domain and the role of customers and standards in that process
- How each of the work products of Application Engineering is produced currently and how it would be produced if reuse were a viable option

2. PRODUCT DESCRIPTION

Name Process Requirements

Purpose	**Process Requirements are an analysis of the current Application Engineering process that identifies work products which provide a focus for reuse efforts. The process described in the Process Requirements may be manual or may incorporate varying levels of automation.**
Content	The Process Requirements work product consists of: • ***Process Specification.*** A definition of the work products, activities, and contributing methods or practices that application engineers currently use. For each activity, Process Specification describes its purpose, the work products created, and interactions with other activities. • ***Work Product Creation Procedure.*** A description of how application engineers produce a work product either with or without reuse.
Form and Structure	A Process Specification can be described informally or using a process modeling notation. The form and content of each work product should be described explicitly or by reference to customer or industry standards. Section AE provides an example of how a Work Product Creation Procedure might be described.
Verification Criteria	The described Application Engineering process should accurately account for current work products and practices of application engineering projects.

3. PROCESS DESCRIPTION

The Process Requirements Activity consists of two steps shown in Figure DE.2.2.3-1.

3.1 PROCEDURE

Follow these steps for the Process Requirements Activity.

Step: Describe the Application Engineering Process

Action	Describe the process that application engineering projects follow to create an application product and its constituent work products.
Input	Domain Definition
Result	Process Specification
Heuristics	• Identify the deliverables that the Application Engineering process produces. This set of work products is determined by the needs of customers for the targeted project. The form and content of some work products may be based on corporate, customer, or industry standards, as appropriate. Work products that must satisfy a form and content standard are more likely to offer opportunities for reuse. • Identify additional (i.e., intermediate or auxiliary) work products that result from the Application Engineering process. These work products

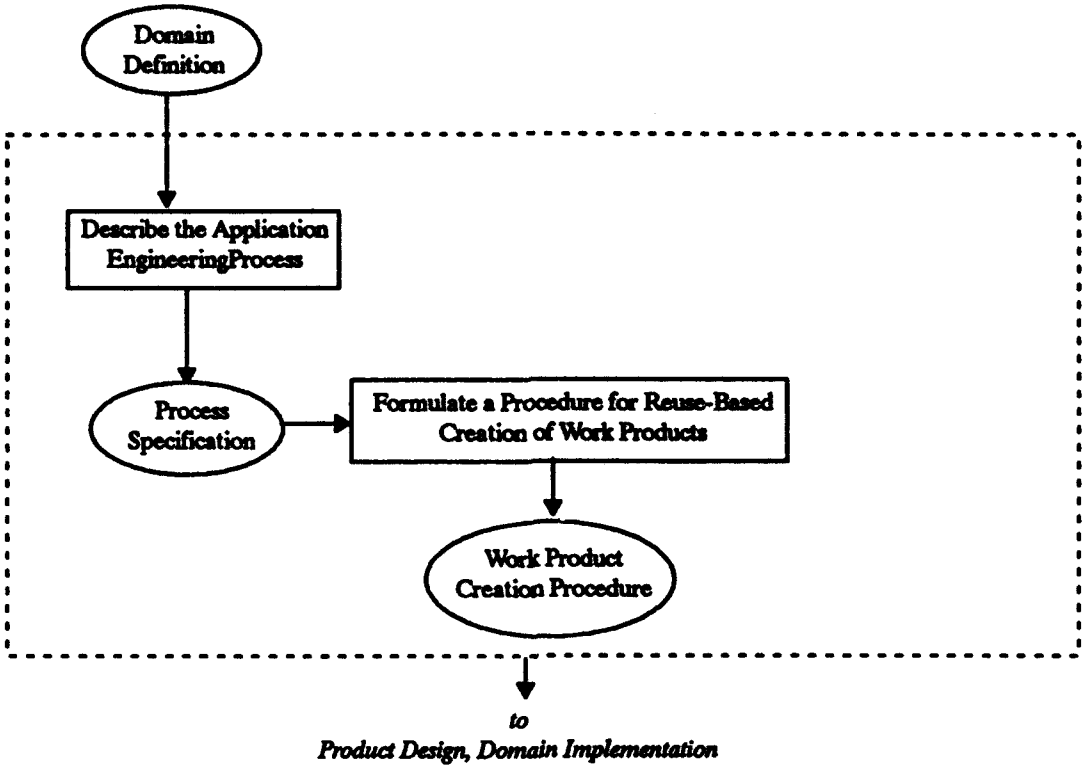


Figure DE.2.2.3-1. Process Requirements Process

retain preliminary or supporting information or support project and process management, including risk management and quantitative and qualitative analyses of deliverable work products and of the production process.

Step: Formulate a Procedure for Reuse-Based Creation of Work Products

Action Formulate a procedure that will serve as an informal guide for application engineers to create a work product, allowing for discovery and exploitation of supported reuse opportunities as appropriate.

Input Process Specification

Result Work Product Creation Procedure

Heuristics

- Start with an approximate description of the steps of work product creation as application engineers currently work. Section AE provides an example of such a work product creation procedure. Any work product can be created, with or without reuse, roughly following that procedure.

- Consider how your work product creation procedure will be affected by the availability of relevant reusable assets. Provide guidance on how reuse should affect the way activities are performed. Consider the guidance given for the procedure in Section AE as a starting point.

3.2 Risk Management

Risk	The documented Application Engineering process does not match actual practices.
Implication	Subsequently developed work product families will not support the creation of work products that projects need to produce.
Mitigation	Review the process with experienced project managers and engineers to ensure that it encompasses all activities required of a project, those work products that customers require, and those additional work products that benefit a project. Variations in project needs must be anticipated and supported.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Definition conflicts with the Domain Plan or is incomplete, ambiguous, or inconsistent.
Source	• Domain Definition Activity • Domain Management Activity
Response	Describe the inadequacies in the Domain Definition (e.g., a targeted project whose needs seem to conflict with the domain scope). Proceed with Process Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency	The Process Requirements are incomplete, ambiguous, or inconsistent.
Source	Process Support Development Activity
Response	Refine the Process Requirements to correct inadequacies.
Contingency	The documented Application Engineering process identifies types of work products that do not correspond to the needs of a particular project.

Source Project Support Activity

Response Include in the Process Requirements any activities and associated work products that offer an opportunity for reuse but have not been identified as such previously. Omit any that no longer correspond to accepted practice.

This page intentionally left blank.

DE.2.2.4. PRODUCT DESIGN ACTIVITY

1. GETTING STARTED

The Product Design Activity is an activity of the Domain Specification Activity for creating a Product Design. A Product Design specifies the design for work product family, rather than for a single work product. A design describes a work product that solves a specified problem. Similarly, a Product Design is a design that varies according to the decisions supported by the work product family's Decision Model. By applying the decisions that characterize a particular work product to the Product Design, a standardized design of that work product is produced. The Product Design Activity is performed for each work product family in the domain.

1.1 Objectives

The objective of the Product Design Activity is to create a design for a work product family. The work product family's design must satisfy its Product Requirements and must be adaptable to the decisions allowed by the family's Decision Model.

1.2 Required Information

The Product Design Activity requires the following information:

- Decision Model
- Product Requirements
- Legacy Products

1.3 Required Knowledge and Experience

The Product Design Activity requires domain and software knowledge and experience in:

- The principles and use of appropriate design method(s) used to construct existing work products in the domain
- How artifacts that represent domain knowledge (e.g., code, documentation, test plans) are designed, including an appreciation of typical engineering tradeoffs to be resolved
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name Product Design

Purpose — A Product Design specifies the design of members of a work product family.

Content — The Product Design consists of the following parts for each work product family:

- ***Product Architecture.*** A (possibly partial) specification of the internal organization of each application engineering work product that can be produced for the family (see Section DE.2.2.4.1).
- ***Component Design.*** A specification of the design of a set of Adaptable Components that can be adapted to compose draft application engineering work products useful for the targeted project (see Section DE.2.2.4.2).
- ***Generation Design.*** A specification of how a work product family's Application Model is used to select, adapt, and compose Adaptable Components to create work products that satisfy the Product Requirements and Product Architecture (see Section DE.2.2.4.3).

Verification Criteria

- All aspects of Product Requirements for a work product family are traceable into the Product Design for that family. All variations in Product Requirements for a work product family have equivalent variations in the Product Design.
- The Product Design satisfies the verification criteria appropriate to the specific design method used in creating it.

3. PROCESS DESCRIPTION

The Product Design Activity consists of the three steps shown in Figure DE.2.2.4-1.

3.1 PROCEDURE

Follow these steps for the Product Design Activity.

Step: Product Architecture Activity

Action — Create design structures that characterize the internal organization of members of the work product family.

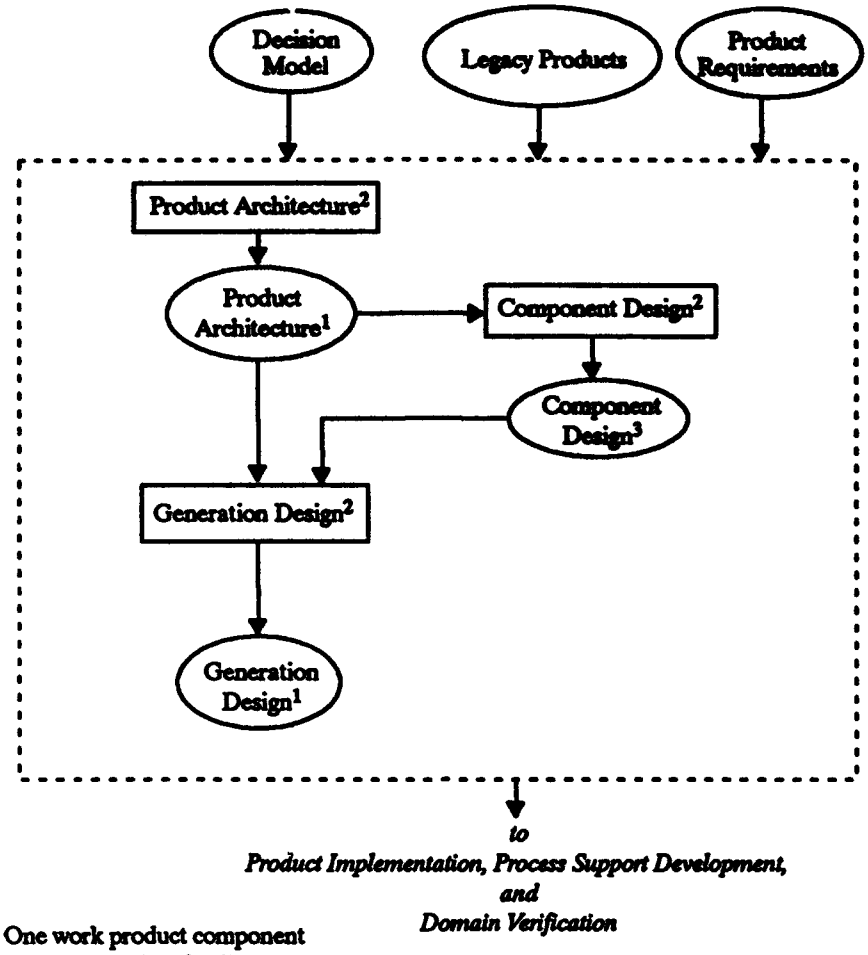
Input

- Product Requirements
- Legacy Products

Result — Product Architecture

Heuristics

- Create multiple design structures (each portraying a different perspective) for a given work product family.
- Ensure that the work product family's Product Architecture applies to all members of that family.



[1] One work product component per work product family

[2] Activities repeated for each work product family

[3] A set of Component Designs per work product family

Figure DE.2.2.4-1. Product Design Process

Step: Component Design Activity

Action	Create a Component Design for each of a set of Adaptable Components that compose a work product family as identified by the Product Architecture.
Input	• Product Requirements • Product Architecture • Legacy Products
Result	Component Design
Heuristics	Ensure that each Component Design satisfies relevant aspects of the Product Architecture and Product Requirements.

Step: Generation Design Activity

Action	Specify a precise procedure of how members of a work product family are derived from Adaptable Components based on the decisions in an Application Model.
Input	• Decision Model • Product Architecture • Component Designs
Result	Generation Design
Heuristics	• Decide how the decisions for a work product family determine the form and content of an initial draft of an application engineering work product. • Specify the design by describing how Adaptable Components are selected, adapted, and composed according to the decisions in the product family's Decision Model.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Decision Model for work product family is incomplete, ambiguous, or inconsistent.
Source	Decision Model Activity
Response	Describe the inadequacies in the Decision Model. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Decision Model.
Contingency	The Product Requirements for a work product family are incomplete, ambiguous, or inconsistent.
Source	Product Requirements Activity
Response	Describe the inadequacies in the Product Requirements. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Design that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency Suggestions are made for Product Design changes to exploit unforeseen opportunities, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source

- Product Implementation Activity
- Process Support Development Activity

Response

- Revise the Product Design.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Product Design for a work product family does not satisfy the Product Requirements.

Source Domain Verification Activity

Response Modify the Product Design to be consistent with the Product Requirements.

Contingency The Product Design for a work product family is incomplete, ambiguous, or inconsistent.

Source Product Implementation Activity

Response Refine the Product Design to correct inadequacies.

This page intentionally left blank.

DE.2.2.4.1. PRODUCT ARCHITECTURE ACTIVITY

1. GETTING STARTED

The Product Architecture Activity is an activity of the Product Design Activity for creating a Product Architecture. This activity is performed for each work product family in the domain. An architecture, for a given work product, is one or more design structures that define the internal organization of that work product from different perspectives. Similarly, a Product Architecture is a description of the internal organization of a work product family. A Product Architecture includes the architecture of each of the work product families that make up the product family.A Product Architecture varies according to the decisions supported by the work product family's Decision Model. A Product Architecture describes a standardized architecture for all members of a work product family in a domain. By applying the decisions that characterize a particular work product to the Product Architecture, a standardized architecture of that work product is produced.

For each work product family (requirements, design, code, etc.), one design structure must identify a set of Adaptable Components. Application engineers compose instances of these components to create a draft work product. Depending on which design method domain engineer's follow, they may also create other structures which provide other views of the behavior or interrelationships of components. In all cases, the structures are in an adaptable form so that Application Engineering can use them to produce any member of the indicated product family.

1.1 Objectives

The objective of the Product Architecture Activity is to define an adaptable architecture for a specific work product family. Product Architecture is the design of solutions to the problems that Product Requirements describe.

Application engineers create work products by selecting, adapting, and composing instances of Adaptable Components that Domain Engineering produces. During Product Architecture, domain engineers identify the structure of each application engineering work product family in terms of components that application engineer's may produce manually or from Adaptable Components.

1.2 Required Information

The Product Architecture Activity requires the following information:

- Product Requirements
- Legacy Products

1.3 Required Knowledge and Experience

The Product Architecture Activity requires domain and software knowledge and experience in:

- The principles and use of the design method used to create members of the work product family (e.g., ADARTS [Software Productivity Consortium 1993])
- How systems in the domain are designed and documented following chosen design and documentation methods
- The concepts and practice of metaprogramming (Campbell 1989)

2. PRODUCT DESCRIPTION

Name Product Architecture

Purpose The Product Architecture describes the internal organization of members of the application engineering work product family.

Content The Product Architecture consists of design structures for the application work product family. One of the structures must identify the components that make up each of the members of the work product family. Each structure consists of:

- A set of design elements
- A relation that associates elements

For example, a software requirements specification document is one type of application engineering work product. The domain engineers might choose sections of requirements documents as design elements, and "subsection" as the relation among these elements. This describes the structure of a software requirements specification document in enough detail to allow its composition from its constituent elements. The structures developed for a particular domain depend on the particular application work products and the design method used to produce them.

Although the Product Architecture for a particular work product may contain multiple design structures, there must be one structure that describes the decomposition of the work product into work assignments (e.g., modules, sections). The elements of this structure correspond to components that are to be implemented by Adaptable Components.

The only difference between the design structures specified in the Product Architecture and those specified in a conventional design is that the Product Architecture is parameterized and adaptable, so that it describes the family of work products in the domain.

Form and Structure The form for each structure of a Product Architecture is a textual or graphic network of elements and relations. This representation is then augmented with a suitable technology such as metaprogramming notation to parameterize the structure for adaptation to variations.

Example DE.2.2.4.1-1 illustrates a fragment of a Product Architecture for a DOD-STD-2167A System/Segment Specification document work product family of the TLC domain. The Product Architecture of this document work product family is expressed as an annotated outline where each numbered heading (e.g., 1.

Scope) corresponds to a section of the work product. Preceding the annotated outline are the decisions (e.g., ARCH_pedestrian_lanes) that parameterize the annotated outline. The content of the annotated outline will vary depending on values chosen for these decisions. For example, the content of Section 3.2.1.1.j varies for some family members. In other family members, this section is omitted.

Instantiation Parameters

ARCH_pedestrian_lanes	one of (yes, no)	{A value of yes means that the System/Segment Specification must include requirements for a pedestrian lane without push buttons. A no value means the SSS must omit these requirements.}
ARCH_pedestrian_lanes_pb	one of (yes, no)	{A value of yes means that the System/Segment Specification must include requirements for a pedestrian lane with push buttons. A no value means the SSS must omit these requirements.}

....

Instantiation Constraints

None

System/Segment Specification Product Architecture (Annotated Outline)

1. Scope

1.1 Identification

The approved identification number, title, and abbreviation of the TLC system.

1.2 System Overview

....

<if Arch_pedestrian_lanes = yes then>
3.2.1.1.1.j Pedestrian_Lanes

The purpose of the Pedestrian lanes is to allow the safe crossing of the intersection by pedestrians. This capability, Pedestrian lanes (TLC-PL), presents the functionality of the walk-don't walk indicators associated with the pedestrian lanes of an intersection.
<endif>

....

<if Arch_pedestrian_lanes_pb = yes then>
3.2.1.1.1.j Pedestrian_Lanes_With_Push_Buttons

The purpose of the Pedestrian lanes with push buttons is to allow pedestrians to safely cross an intersection. This capability, Pedestrian lanes with push buttons (TLC-PB), presents the functionality of the walk-don't walk indicators and the push buttons associated with the pedestrian lanes of an intersection.
<endif>

....

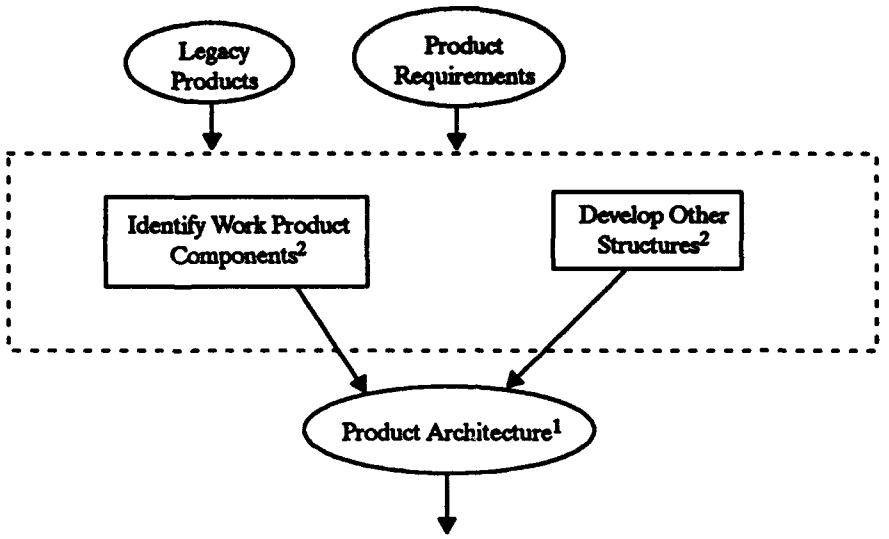
Example DE.2.2.4.1-1. Fragment of TLC Product Architecture for the System/Segment Work Product Family

Verification Criteria

- Each Product Architecture structure satisfies the verification criteria established by the specific design method used in its creation.
- The Product Architecture defines all structures for the software and other work products required by the Application Engineering process.

3. PROCESS DESCRIPTION

The Product Architecture Activity consists of two steps shown in Figure DE.2.2.4.1-1.



[1] One work product component per work product family

[2] Activities repeated for each work product family

Figure DE.2.2.4.1-1. Product Architecture Process

3.1 PROCEDURE

These steps are performed for the Product Architecture Activity.

Step: Identify Work Product Components

Action Develop a structure that describes, as a structured set of components, the internal organization of the work products in the family.

Input

- Product Requirements
- Legacy Products

Result Product Architecture: Internal Organization

Heuristics

- Examine Legacy Products for components and structural relationships. A work product usually contains structures that intuitively identify

components. A calling hierarchy is an example for code. In documents, consider basing the structure around the table of contents, especially if a standard format exists in the domain (see the following heuristic).

- Let the design methods used in the Legacy Products guide you. For example, systems developed using ADARTS include an information hiding hierarchy. Systems developed using structured design have a functional decomposition. Documents developed using DOD-STD-2167A have a predefined section organization.

- The structures that result from Product Architecture establish a de facto standard in your organization if none already exists. You should, therefore, try to use standard organizational formats in the domain, such as those defined by ADARTS or DOD-STD-2167A. Incorporate any formats that are relevant to the targeted project.

- The decisions that parameterize a structure are related to the Decision Model for the work product family of which the Product Architecture forms a solution. However, the Decision Model describes the problem space of the work product family, whereas the Product Architecture is associated with the solution space of that family. To reflect this difference, you may want to create parameters that are not in the Decision Model. During the Generation Design Activity, you will map these parameters to decisions in the Decision Model.

- One motivation for creating a Product Architecture is to break down a work product into a set of components on which subsequent Domain Engineering activities (Component Design, Component Implementation, Generation Design) can be performed independently. If the packaging of Legacy Products does not conform to the internal organization, create one that does.

Step: Develop Other Structures

Action Create any other structures required to define the Product Architecture fully.

Input

- Product Requirements
- Legacy Products

Result Product Architecture: Alternate Structures

Heuristics

- The chosen design method for software identifies other required design structures. Using ADARTS, these design structures are the Process Structure and the Dependency Structure. Hypertext-based documents would have an analogous alternative structure.

- Alternate structures impose constraints on the implementation of each component of the internal organization. The design method characterizes these constraints.

- Each structure must support a subset of the same variations. The internal organization determines how these variations affect each component. Component variations must account properly for variations in relevant parts of the alternate structures.

3.2 Risk Management

Risk — The Product Architecture will not support all features or variations in Product Requirements.

Implication — The Product Architecture is not a correct solution to the Product Requirements.

Mitigation — Review the Product Architecture with developers of the Product Requirements and experienced designers. Establish traceability of all required features to elements of the architecture. Evaluate whether variations that characterize different work products lead to proper architectural variations.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency — The Product Requirements are incomplete, ambiguous, or inconsistent.

Source — Product Requirements Activity

Response — Describe the inadequacies in the Product Requirements. Proceed with Product Architecture, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency — The Domain Plan cannot be satisfied with available technical capabilities.

Source — Domain Management Activity

Response — Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Architecture that satisfies the Domain Plan as closely as possible.

Contingency — The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source — Domain Management Activity

Response — Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency — The Product Architecture does not satisfy the Product Requirements.

Source	Domain Verification Activity
Response	Modify the Product Architecture to be consistent with the Product Requirements.
Contingency	The Product Architecture is incomplete, ambiguous, or inconsistent.
Source	• Product Implementation Activity • Generation Design Activity • Component Design Activity
Response	Refine the Product Architecture to correct inadequacies.

This page intentionally left blank.

DE.2.2.4.2. COMPONENT DESIGN ACTIVITY

1. GETTING STARTED

The Component Design Activity is an activity of the Product Design Activity for creating a Component Design. The Product Architecture identifies a set of Adaptable Components that may be used to implement a work product family. A Component Design is a design specification for one of these Adaptable Components. Application engineers, using the Generation Procedure, may adapt and compose a set of these components to implement certain work products, or portions thereof. Each component must be designed to satisfy relevant aspects of the Product Requirements and all design structures of the Product Architecture.

1.1 Objectives

The objective of the Component Design Activity is to produce a design for an Adaptable Component that satisfies applicable Product Requirements in accordance with its role in the Product Architecture.

1.2 Required Information

The Component Design Activity requires the following information:

- Product Requirements
- Product Architecture
- Legacy Products

1.3 Required Knowledge and Experience

The Component Design Activity requires domain and software knowledge and experience in:

- How components of systems in the domain are designed
- The principles and use of an appropriate design method
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name Component Design

Purpose

A Component Design is a specification for an Adaptable Component that can be used to construct a draft application work product.

Content

Each Component Design represents a family of components. A Component Design consists of two parts:

- ***Adaptation Specification.*** The Adaptation Specification for an Adaptable Component describes the ways that the component can be tailored via a set of parameters. Each parameter has a name and type to indicate its range of variations. Constraints identify invalid combinations of parameter variations.

- ***Interface Specification.*** The Interface Specification describes the desired characteristics of the implementation of the component. The exact content of the interface specification is particular to the component type and the design method used. To describe the entire family, the interface specification is parameterized with respect to the variations in the Adaptation Specification.

Form and Structure

The Adaptation and Interface Specifications each include textual and tabular information. The form of an Adaptation Specification is the same for all types of components and includes the following information:

- ***Name.*** Name of the Adaptable Component.

- ***Instantiation Parameters.*** Adaptation parameters for the component, including the name, type, and description of each parameter.

- ***Instantiation Constraints.*** Constraints on the instantiation of the Adaptable Component (e.g., constraints on the legal combination of parameter values).

The interface part of Adaptable Components is different for software and documentation. The content of the software interface is specific to the design method(s) used to create the members of the family. The following types of information are examples: definitions of interface programs (names, parameters, parameter types, returned values), definitions of exported types, descriptions of the effects of interface programs, assumptions about the environment in which the software is to be used.

The interface for a documentation component does not require the same type of detailed information. It consists of a brief statement of the content of the component. It should provide enough information to determine the importance of selecting this component as part of a Generation Procedure. Example DE.2.2.4.2-1 illustrates a fragment of a Component Design for a DOD-STD-2167A System/Segment Specification work product family of the TLC domain. This design corresponds to one of the adaptable components identified in the Product Architecture shown in Example DE.2.2.4.1-1. The Adaptation Specification defines the adaptation parameters for this adaptable

component. The Interface Specification is parameterized, where appropriate, in terms of these adaptation parameters.

Component Design – Pedestrian Lanes with Push Buttons (TLC-PL-PB-1)

Adaptation Specification

Instantiation Parameters

COMP_interface	one of (TL-A, TL-B)	{type of interface to the "walk/don't walk" light indicators}
COMP_blinking_rate	one of (yes, no)	{A yes value means the pedestrian lane indicators have a variable blinking rate capability. A no value means they do not.}

....

Instantiation Constraints

None

Interface Specification

This component contains the functional requirements for the pedestrian lanes present at the intersection. All the pedestrian lanes with push buttons at an intersection must satisfy these requirements. The indicators and the push button associated with the pedestrian lanes of this intersection conform to the <COMP_interface> standard.

<if COMP_blinking_rate is yes then>
The variable blinking rate for the pedestrian lane indicators must be set by the system.
<endif>

....

Example DE.2.2.4.2-1. Fragment of TLC Component Design for the System/Segment Specification Work Product Family

Verification Criteria

- The Component Design satisfies the verification criteria established by the specific design method(s) used for its creation.
- The Component Design satisfies all structures of the Product Architecture for the work product family.
- The value for each parameter in an Adaptation Specification either is derivable from the Decision Model for the work product family or has a fixed, default value for all instances of the component family.

3. PROCESS DESCRIPTION

The Component Design Activity consists of two steps shown in Figure DE.2.2.4.2-1.

3.1 Procedure

Follow these steps for the Component Design Activity. Domain engineers perform these steps for each Adaptable Component defined in the internal organization of the Product Architecture.

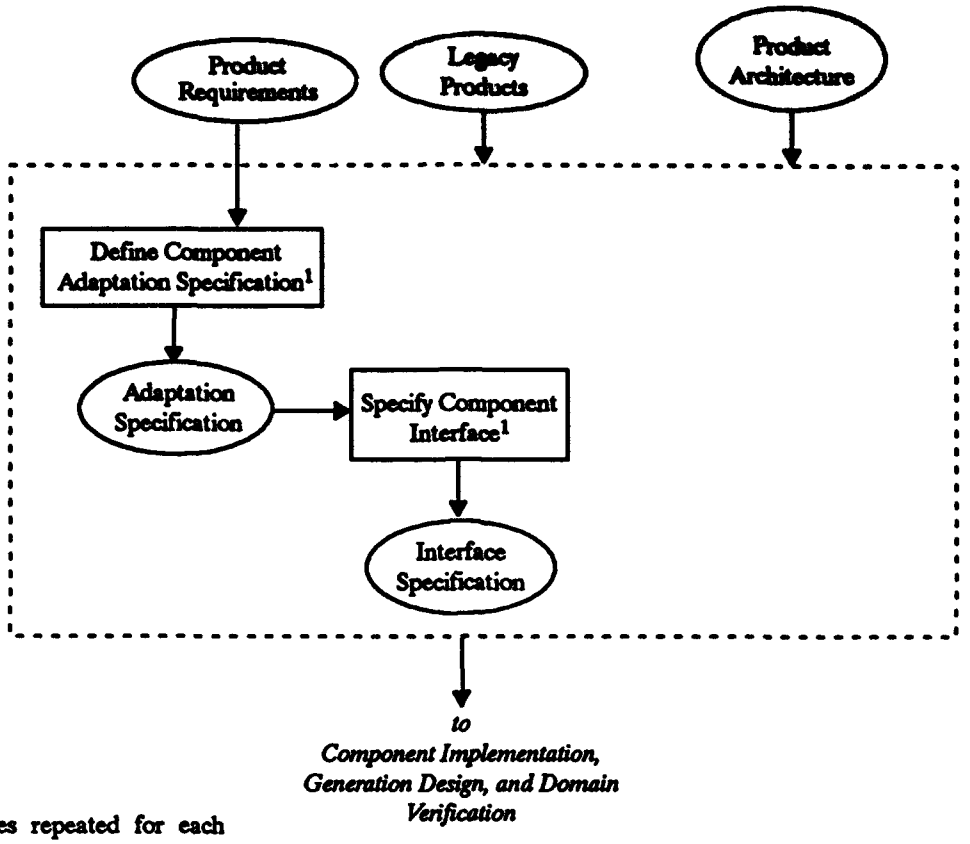


Figure DE.2.2.4.2-1. Component Design Process

Step: Define Component Adaptation Specification

Action Identify the variations that parameterize the Adaptable Component, and record constraints on legal combinations.

Input

- Product Architecture
- Product Requirements
- Legacy Products

Result Component Design: Adaptation Specification

Heuristics

- Identify components in the Product Architecture that have the same form in all instances of the domain. These components have no associated variations (i.e., they are not adaptable). You must still provide an interface specification, but you may omit the Adaptation Specification.
- Examine Legacy Products to determine variations. Concentrate on semantic, rather than syntactic, distinctions. However, unless you are willing to reengineer work products, you may need to define variations based on syntactic distinctions as well.

- Determine necessary component adaptations by analyzing the Product Requirements to see how the component must vary to satisfy relevant requirements. In practice, you should try to use both approaches.

- Decisions that parameterize components must derive from, but need not be, the decisions identified in the Decision Model. In general, there is a many-to-many relationship between Decision Model decisions and Component Design parameters. You may use whatever decisions most naturally specify variations among members of the family defined by an Adaptable Component.

Step: Specify Component Interface

Action	Specify the requisite properties for the implementation of each component.
Input	• Product Architecture • Component Design: Adaptation Specification
Result	Component Design: Interface Specification
Heuristics	The properties that you must specify depend on the type of component and the design method used. Parameterize each component interface with the decisions from the component's adaptation specification so that it describes all instances of the component.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Product Requirements are incomplete, ambiguous, or inconsistent.
Source	Product Requirements Activity
Response	Describe the inadequacies in the Product Requirements. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Component Design that satisfies the Domain Plan as closely as possible.

Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.
Contingency	The Product Architecture is incomplete, ambiguous, or inconsistent.
Source	Product Architecture Activity
Response	Describe the inadequacies in the Product Architecture. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.

4.2 Feedback From Product Consumers

Contingency	Suggestions are made for Component Design changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Component Design was completed.
Source	• Product Implementation Activity • Process Support Development Activity
Response	• Revise the Component Design. • Refer to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Component Design does not satisfy the Product Requirements.
Source	Domain Verification Activity
Response	Modify the Component Design to be consistent with the Product Requirements.
Contingency	The Component Design is incomplete, ambiguous, or inconsistent.
Source	• Component Implementation Activity • Generation Design Activity
Response	Refine the Component Design to correct inadequacies.

DE.2.2.4.3. GENERATION DESIGN ACTIVITY

1. GETTING STARTED

The Generation Design Activity is an activity of the Product Design Activity for creating a Generation Design. A Generation Design is a specification of production procedures that an application engineer uses to produce draft application engineering work products. A Generation Design defines a transformation (or mapping) from an Application Model to the equivalent application engineering work products. For each application engineering work product, a Generation Design specifies how to select and adapt Adaptable Components according to decisions in an Application Model and to compose them according to the internal organization of that work product in the Product Architecture. The Generation Design Activity is performed for each work product specified by the Product Requirements.

1.1 Objectives

The objective of the Generation Design Activity is to produce a specification for the production procedures that can be used to produce application engineering work products for a member of a work product family through reuse of Adaptable Components. The specification establishes a correspondence between an Application Model and equivalent domain engineering work products that implement the intent of the model correctly.

1.2 Required Information

The Generation Design Activity requires the following information:

- Decision Model
- Product Architecture
- Component Designs

1.3 Required Knowledge and Experience

The Generation Design Activity requires domain and software knowledge and experience in:

- How work products of systems in the domain are designed
- The principles and use of the design method used for the Product Architecture
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name

Generation Design

Purpose

A Generation Design is a specification for a production procedure for creating draft application engineering work products.

Content

A Generation Design relates the decisions from the Decision Model to the elements of a work product's internal organization defined in the Product Architecture. A Generation Design consists of three mappings:

- ***Architecture Mapping.*** The Architecture Mapping is a description of the relation between decisions in the work product family's Decision Model and the decisions of the corresponding adaptable Product Architecture. This mapping describes how values for the Product Architecture's decisions are determined from values of decisions in the Decision Model. As a result, the Architecture Mapping defines the internal organization of a work product that describes a member of the work product family based on decisions in the Decision Model (i.e., from an Application Model).

- ***Component Mapping.*** A Component Mapping is a description of the relation of each element of the organizational structure to an Adaptable Component that implements that element. This mapping defines how each component of a work product is to be produced.

- ***Decision Mapping.*** A Decision Mapping is a description of the relation between decisions in the work product family's Decision Model and the instantiation parameters in the adaptation specification of a Component Design for each work product component. This relation describes how values for the instantiation parameters are determined from values of decisions in the work product family's Decision Model.

Form and Structure

There is a Generation Design for each supported work product. The Architecture Mapping is represented as a statement for each instantiation parameter of the work product's Product Architecture. The statement contains a pairing between an instantiation parameter and an expression. The expression to determine the value to assign an instantiation parameter is described in terms of decisions in the work product family's Decision Model. The expression may involve iteration over a group of decisions or conditional testing of one or more decisions.

The Decision Mapping representation is similar to the Architecture Mapping, except that the instantiation parameters come from the adaptation specification of the Component Design for the work product.

The Component Mapping is represented as a "use" statement. If the expression bracketing the use statement is True, then the use statement describes which Adaptable Component contains the needed implementation.

The expression is usually described in terms of decisions in the work product family's Decision Model. However, if the Adaptable Component is always used, then an expression of True is sufficient to describe this situation.

Example DE.2.2.4.3-1 illustrates a fragment of a Generation Design for a work product family of the TLC domain. It depicts one way of representing the expressions discussed for Architecture, Component, and Decision Mapping. The decisions used the metaprogramming notation come directly from the Decision Model shown in Example DE.2.2.1-1. The parameters on the lefthand side of the "=" statements in the Architecture and Decision Mapping come from Examples DE.2.2.4.1-1 and DE.2.2.4.2-1, respectively.

```
Architecture Mapping
    ARCH_pedestrian_lanes =  <if there is a Street S that has a Pedestrian_Lane specified
                                 and S.Pedestrian_Lane.Push_Button_Mechanism = no then>
                             yes
                             <else>
                             no
                             <endif>

    ARCH_pedestrian_lanes_pb =  <if there is a Street S that has a Pedestrian_Lane specified
                                      and S.Pedestrian_Lane.Push_Button_Mechanism = yes then>
                                yes
                                <else>
                                no
                                <endif>

    ....

Component Mapping
    Required topic: Pedestrian_Lanes_With_Push_Buttons

        <if TLC_SSS.HW_Platform.Platform = TLC2 then>
            use component "Pedestrian lanes with Push Buttons (TLC-PL-PB-1)"
        <else>
            use component "Pedestrian lanes with Push Buttons (TLC-PL-PB-2)"
        <endif>

    ....

Decision Mapping
    Pedestrian lanes with Push Buttons (TLC-PL-PB-1)
        COMP_interface =  <if TLC_SSS.HW_Platform.Interface = TL-A then>
                          TL-A
                          <else>
                          TL-B
                          <endif>

        ....

    ....
```

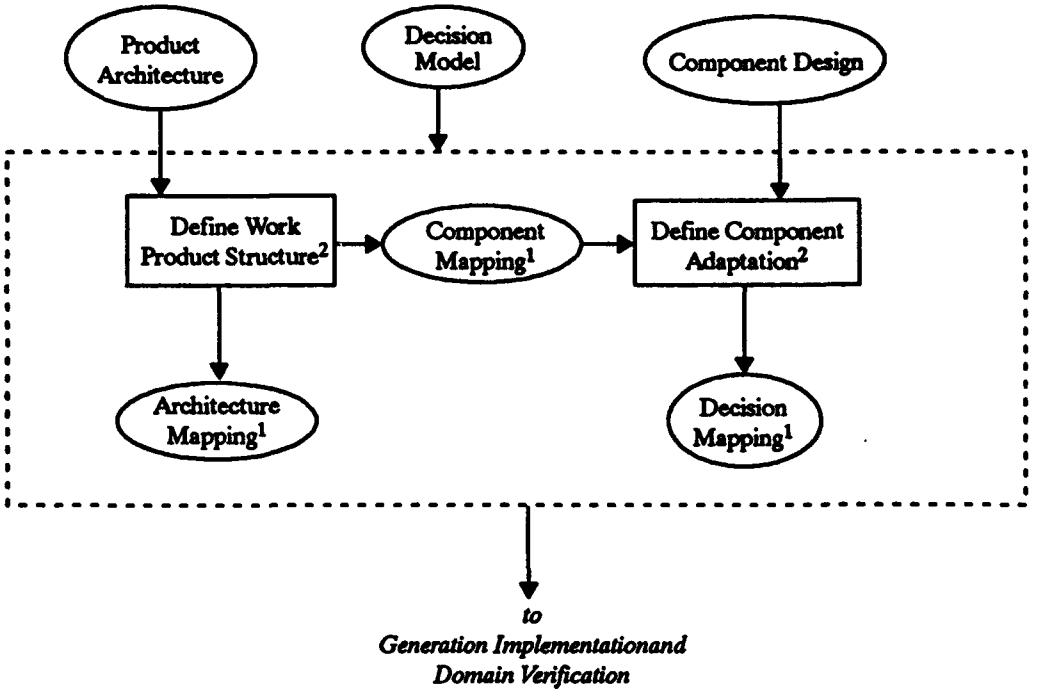
Example DE.2.2.4.3-1. Fragment of TLC Generation Design for the System/Segment Specification Work Product Family

Verification Criteria

- The Generation Design specifies mappings that will produce application work products which exhibit the internal organization specified in the Product Architecture.
- The Generation Design specifies mappings that produce application work products which satisfy the Product Requirements (i.e., the mappings are consistent with Product Requirements variation).
- All variabilities allowed by decisions are properly represented as product variations.
- The effects of variabilities among work products are mutually consistent (i.e., all mappings are consistent).

3. PROCESS DESCRIPTION

The Generation Design Activity consists of two steps shown in Figure DE.2.2.4.3-1.



[1] One work product component per work product family

[2] Activities repeated for each work product family

Figure DE.2.2.4.3-1. Generation Design Process

3.1 PROCEDURE

Step: Define Work Product Structure

Action Define how decisions in the Decision Model affect the structure of the work product.

Input

- Decision Model
- Product Architecture

Result

Generation Design: Architecture and Component Mappings

Heuristics

- It is sufficient to define the work product structure as a mapping from the Decision Model to the internal organization of the Product Architecture for the work product. The internal organization defines the components that are required to implement the work product. This mapping determines which elements of the Product Architecture are implemented for a particular Application Model.
- The Product Architecture determines (conditionally and iteratively) how components of each work product are to be derived from Adaptable Components (i.e., the component mapping is provided implicitly by the Product Architecture). The Generation Design should not modify that mapping.
- Represent this mapping in metaprogramming notation associated with components in the Product Architecture. The mapping is defined in terms of decisions in the Decision Model and determines whether (one or more of) the associated component(s) should be included in the product created for a given Application Model. This mapping is formed by analyzing the Product Architecture and noting conditions that must be true if a particular component is to be included. If a component is always included in the product, metaprogramming notation is not required.
- Several Adaptable Components might be used to implement a single Product Architecture component, depending on decisions in the Application Model. In this case, use a conditional in the Component Mapping to qualify the association between the Product Architecture and Adaptable Components, thus indicating when a particular Adaptable Component is used.

Step: Define Component Adaptation

Action

Define a mapping from the decisions in the Decision Model to adaptations of the Adaptable Components referenced by the Component Mapping.

Input

- Decision Model
- Component Design
- Generation Design: Component Mapping

Result

Generation Design: Decision Mapping

Heuristics

When a particular Component Design is to be used to implement a particular component in the Product Architecture, the variability of the Adaptable Component (i.e., its parameterization) must be realized in terms of decisions

from the Decision Model. Define the value of each parameter (by name) as a derivation from Decision Model decisions.

3.2 Risk Management

Risk	The Generation Design will not produce correctly-structured work products.
Implication	Application Production will not produce acceptable application engineering work products.
Mitigation	Derive work product structures from the Generation Design for Application Models of familiar work products and review the result with experienced engineers to determine whether the result is acceptable.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Decision Model is incomplete, ambiguous, or inconsistent.
Source	Decision Model Activity
Response	Describe the inadequacies in the Decision Model. Proceed with Product Architecture and document any assumptions made regarding the inadequate portions of the Decision Model.
Contingency	The Product Requirements for a work product family are incomplete, ambiguous, or inconsistent.
Source	Product Requirements Activity
Response	Describe the inadequacies in the Product Requirements. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Generation Design that satisfies the Domain Plan as closely as possible.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

Contingency	The Product Architecture for a work product family is incomplete, ambiguous, or inconsistent.
Source	Product Architecture Activity
Response	Describe the inadequacies in the Product Architecture. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.
Contingency	The Component Design for a work product family is incomplete, ambiguous, or inconsistent.
Source	Component Design Activity
Response	Describe the inadequacies in the Component Design. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Component Design.

4.2 Feedback From Product Consumers

Contingency	The Generation Design for a work product family does not satisfy the Product Requirements.
Source	Domain Verification Activity
Response	Modify the Generation Design to be consistent with the Product Requirements.
Contingency	The Generation Design for a work product family is incomplete, ambiguous, or inconsistent.
Source	Generation Implementation Activity
Response	Refine the Generation Design to correct inadequacies.

This page intentionally left blank.

DE.2.3. DOMAIN VERIFICATION ACTIVITY

1. GETTING STARTED

Domain Verification is an activity of Domain Engineering for ensuring the correctness, consistency, and completeness of domain engineering work products. Both formal and informal techniques may be applied to the domain engineering work products to verify these properties. Domain Verification is an independent verification activity performed separately from, and in addition to, the verification performed as part of each Domain Engineering Activity.

The Domain Verification Activity is motivated by the same concern that motivates Independent Verification and Validation (IV&V) in a conventional software development process; namely, that engineers involved in developing a work product cannot objectively judge the quality of that work product. Independent validation of the domain engineering product, from the perspective of client projects, is conducted in the Domain Validation step of the Project Support Activity.

Domain Verification establishes the correctness, consistency, and completeness of domain engineering work products. These terms have a precise meaning in the context of this activity. The concept of correctness is that of relative correctness. Similarly, the concept of completeness is that of relative completeness. A work product is said to be correct (complete) with respect to some criteria or to a more abstract representation of the entity the work product describes. For example, the Product Implementation for a work product family can be said to be correct (complete) with respect to its Product Requirements. Consistency, on the other hand, is a term that applies to a collection of related work products (at the same level of abstraction) that form a whole. Two products are consistent when they exhibit the intended interrelationships. For example, the Product Architecture, Component Design, and Generation Design work products for a work product family are strongly interrelated, and, therefore, mutual consistency is an important property for these work products.

1.1 Objectives

The objective of Domain Verification is to independently evaluate the quality of domain engineering work products.

1.2 Required Information

Domain Verification requires the following information:

- Domain Definition
- Domain Specification

- Domain Implementation
- Domain Plan: Practices and Procedures

1.3 Required Knowledge and Experience

The Domain Verification Activity requires domain and software knowledge and experience in:

- Appropriate software verification techniques
- How to systematically plan and perform software verification

2. PRODUCT DESCRIPTION

There are no Synthesis work products produced during Domain Verification.

3. PROCESS DESCRIPTION

The Domain Verification Activity consists of three steps shown in Figure DE.2.3-1.

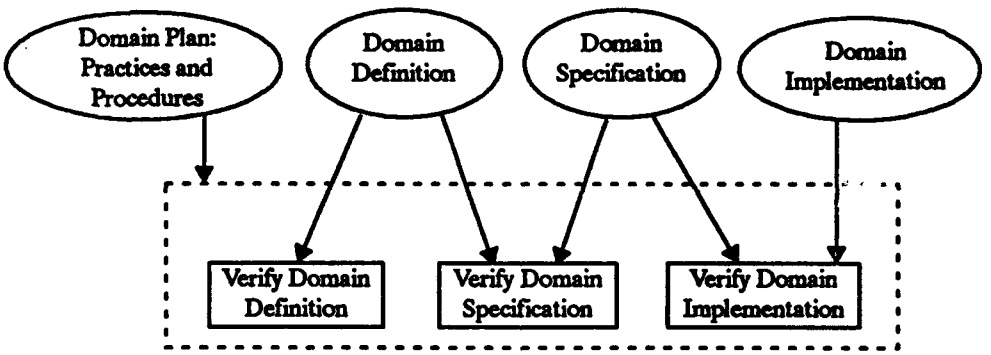


Figure DE.2.3-1. Domain Verification Process

3.1 Procedure

Step: Verify Domain Definition

Action	Verify the correctness, consistency, and completeness of the Domain Definition.
Input	• Domain Definition • Domain Plan: Practices and Procedures
Result	None
Heuristics	• Verify that the parts of the Domain Definition are correct and complete with respect to the guidance provided in their respective activity descriptions.

- Verify that the parts of the Domain Definition are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.
- Verify that the Domain Synopsis, Domain Glossary, and Domain Assumptions are mutually consistent.
- Use the verification criteria, established for the Domain Definition in its activity description, as guidance in verifying the Domain Definition.
- Use static analysis techniques (e.g., formal inspections, reviews, analysis tools) to verify the Domain Definition. These techniques are appropriate because the Domain Definition is typically represented in document form.

Step: Verify Domain Specification

Action

Verify the correctness, consistency, and completeness of each work product family in the Domain Specification.

Input

- Domain Definition
- Domain Specification
- Domain Plan: Practices and Procedures

Result

None

Heuristics

- Perform this step for each work product family in the Domain Specification.
- Verify that the parts of the Domain Specification are correct and complete with respect to the guidance provided in their respective activity descriptions.
- Verify that the parts of the Domain Specification are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.
- Verify that the Product Requirements and Product Design for a work product family are consistent with its Decision Model. This means that these work products only reference decisions in the Decision Model and, conversely, all applicable decisions in the Decision Model are reflected in the work products.
- Verify that the Product Architecture, Component Design, and Generation Design for a work product family are mutually consistent.
- Verify that the Product Design for a work product family is correct and complete with respect to its Product Requirements.

- Verify that the Process Requirements is correct and complete with respect to the assumptions about the Application Engineering process in the Domain Definition. The Application Engineering process is normally not explicitly described in the Domain Definition, but the Domain Definition will typically constrain what is an acceptable Application Engineering process.
- Verify that the Product Requirements and Product Design are correct and complete with respect to the representation of the Product Family in the Domain Synopsis and Domain Assumptions parts of the Domain Definition.
- Use the verification criteria, established for the Domain Specification work products in their respective activity descriptions, as guidance in verifying the Domain Specification.
- Use static analysis techniques (e.g., formal inspections, reviews, analysis tools) to verify the Domain Specification for a work product family. These techniques are appropriate because the Domain Specification is typically represented in document form. If parts of the Domain Specification are represented in an executable form, the use of dynamic analysis techniques may be appropriate.

Step: Verify Domain Implementation

Action

Verify the correctness, consistency, and completeness of each work product family in the Domain Implementation.

Input

- Domain Specification
- Domain Implementation
- Domain Plan: Practices and Procedures

Result

None

Heuristics

- Perform this step for each work product family in the Domain Implementation.
- Establish the criteria that you expect the Domain Implementation to meet before you try to verify it. Identify analysis that you can perform that the Domain Implementation is correct with respect to the Domain Specification. Your plan should minimally establish verification objectives and describe a strategy for meeting those objectives.
- Verify that the parts of the Domain Implementation are correct and complete with respect to the guidance provided in their respective activity descriptions.
- Verify that the parts of the Domain Implementation are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.

- Verify that the Process Support and Product Implementation for a work product family are mutually consistent.

- Verify that the Component Implementation and the Generation Implementation for a work product family are mutually consistent.

- Verify that the Process Support for a work product family is correct with respect to its Process Requirements.

- Verify that documents and automation that make up the Process Support are engineered in a way that adequately addresses human factors concerns. For example, you should establish that the Application Engineering Environment portion of Process Support has the qualities of usability, adequate performance, and tolerance of user errors.

- Verify that the Product Implementation for a work product family is correct and complete with respect to its Domain Specification. The requirements for the Product Implementation are represented in the Product Requirements portion of the Domain Specification. The internal organization for the Product Implementation is represented in the Product Design portion of the Domain Specification.

- Verify that a work product produced using the Process Support has expected properties. Do this by resolving the decisions of the work product family's Decision Model, producing the work product corresponding to that model, and then verifying that the work product has the expected properties. Specifically:

  - Verify that the work product produced by the Process Support is correct and complete with respect to the Product Requirements and Product Design of its corresponding work product family (appropriately instantiated with the decisions from the Decision Model).

  - Verify the usability and correctness of the Delivery Support. This should be established through direct inspection and by using the delivery support to install/deliver the Application Product.

  A good strategy for selecting work products to produce is to try to build all or part of Legacy Products that are within the intended scope of the domain.

- Use the verification criteria, established for the Domain Implementation work products in their respective activity descriptions, as guidance in verifying the Domain Implementation.

- Use conventional verification techniques that are appropriate to the task of verifying the Domain Implementation for a work product family. Static analysis techniques (e.g., inspections) are appropriate for static

representations of the Domain Implementation (e.g., Application Engineering User Guide). Dynamic analysis techniques (e.g., testing) are appropriate for dynamic aspects of the Domain Implementation (e.g., automated support for specification, analysis, and product generation).

3.2 Risk Management

Risk	The criteria used to evaluate the domain engineering work products will be unduly influenced by the final content and form of the work products themselves.
Implication	The effectiveness of the verification effort will be reduced.
Mitigation	Define acceptable levels of correctness, completeness, and consistency for each domain engineering work product prior to examining it.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Definition is incorrect, inconsistent, or incomplete.
Source	Domain Definition Activity
Response	Precisely communicate how the Domain Definition is incorrect, inconsistent, or incomplete.
Contingency	The Domain Specification for a work product family is incorrect, inconsistent, or incomplete.
Source	Domain Specification Activity
Response	Precisely communicate how the Domain Specification is incorrect, inconsistent, or incomplete.
Contingency	The Domain Implementation for a work product family is incorrect, inconsistent, incomplete.
Source	Domain Implementation Activity
Response	Precisely communicate how the Domain Implementation is incorrect, inconsistent, or incomplete.

4.2 Feedback From Product Consumers

None

DE.3. DOMAIN IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Domain Implementation is an activity of Domain Engineering for implementing product and process support for application engineering projects in a business area. The Domain Implementation must satisfy the Domain Specification created by Domain Analysis. Product support consists of a set of production procedures and associated Adaptable Components that can be used to create standardized members of a work product family. Process support consists of procedures, documentation, and, optionally, automation that support opportunistic reuse during Application Engineering. The Domain Implementation Activity is performed for each work product family in the domain.

1.1 Objectives

The objectives of the Domain Implementation Activity are to:

- Create a set of Adaptable Components and associated Generation Procedures as specified in the Product Design for the work product families designated as relevant to the targeted project
- Create standard procedures by which production of application engineering work products takes advantage of provided reuse opportunities

1.2 Required Information

The Domain Implementation Activity requires the following information:

- Domain Specification
- Legacy Products

1.3 Required Knowledge and Experience

The Domain Implementation Activity requires domain and software knowledge and experience in:

- Technologies for creating, adapting, and composing Adaptable Components into work products and the verification of such Adaptable Components and work products
- Documenting and providing automated support for Application Engineering activities
- How work products of existing systems in the domain are implemented

2. PRODUCT DESCRIPTION

Name Domain Implementation

Purpose A Domain Implementation contains sets of Adaptable Components and associated production procedures that you can use to create members of work product families relevant to the targeted project. The Domain Implementation also consists of the procedures, documentation, and, optionally, automation that support opportunistic reuse during Application Engineering.

Content A Domain Implementation consists of two components:

- ***Product Implementation.*** A Product Implementation contains organized implementations of work product families (see Section DE.3.1).
- ***Process Support.*** An application engineering infrastructure that supports the targeted project in performing opportunistic reuse of work products (see Section DE.3.2).

Verification Criteria The Domain Implementation supports each designated work product family identified in the Domain Specification as prescribed by the Product Requirements for that family.

3. PROCESS DESCRIPTION

The Domain Implementation Activity consists of the two steps shown in Figure DE.3-1.

3.1 PROCEDURE

Follow these steps for the Domain Implementation Activity.

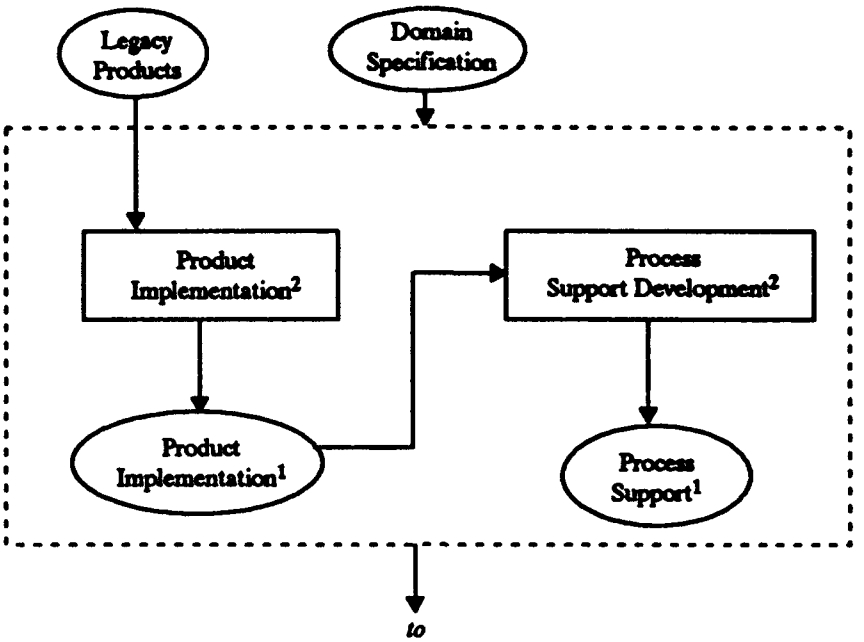
Step: Product Implementation Activity

Action Implement a work product family.

Input
- Domain Specification
- Legacy Products

Result Product Implementation

Heuristics
- Derive the Product Implementation of a work product family from appropriate Legacy Products.
- Describe a mechanical procedure by which application engineers can select, adapt, and compose a draft application work product.
- Implement only the portions of the product (i.e., members of the work product family) required for the targeted project.



[1] One work product component per work product family

[2] Activities repeated for each work product family

Figure DE.3-1. Domain Implementation Process

Step: Process Support Development Activity

Action	Create an application engineering infrastructure to support reuse of existing application engineering work products.
Input	• Product Implementation • Domain Specification: Process Requirements
Result	Process Support
Heuristics	• Document a procedure that application engineers can follow, as part of their normal process for developing a work product, to help them locate and reuse existing work products. • Optionally provide automated mechanisms which support the effective and correct performance of the reuse-related activities of Application Engineering for the targeted project.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Specification is incomplete, ambiguous, or inconsistent.

Source	Domain Analysis Activity
Response	Describe how the Domain Specification is inadequate and suggest how it may be modified. Proceed with Domain Implementation as far as possible with the current Domain Specification.
Contingency	Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.
Source	Domain Analysis Activity
Response	Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback From Product Consumers

Contingency	The Domain Implementation for a work product family is incorrect, inconsistent, or incomplete.
Source	Domain Verification Activity
Response	Request clarification of the intent of the Domain Specification, if necessary. Modify the Domain Implementation to satisfy the Domain Specification.
Contingency	The support for the Application Engineering process is inefficient.
Source	Project Support Activity
Response	Revise the Domain Implementation based on Application Engineering experience.

DE.3.1. PRODUCT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

The Product Implementation Activity is the activity of the Domain Implementation Activity for creating a Product Implementation. A Product Implementation is an implementation of a set of work product families. A conventional implementation is a work product that solves a specific problem. Similarly, a Product Implementation is an implementation that is adaptable to decisions supported by the work product family's Decision Model in order to solve any of a family of problems. A Product Implementation consists of Adaptable Components (e.g., code, documentation, and support for verification/validation) and procedures, as needed, for selecting, adapting, and composing these components. The Adaptable Components and procedures are used to create draft application engineering work products in accordance with an Application Model that describes the work product. The Product Implementation Activity is performed for each work product family in the domain.

1.1 Objectives

The objective of the Product Implementation Activity is to implement the Product Design using artifacts that represent domain knowledge (e.g., code, documentation, test plans) from existing systems. This implementation is used by application engineers to generate required work products for systems in the domain.

1.2 Required Information

The Product Implementation Activity requires the following information:

- Product Requirements
- Product Design
- Decision Model
- Legacy Products

1.3 Required Knowledge and Experience

The Product Implementation Activity requires domain and software knowledge and experience in:

- The design method used in specifying the Product Design
- Existing work products in the domain, including how they are designed, implemented, and verified, and what are their components and architectures

- Target language and platform capabilities
- The technologies for adapting and composing components into work products that make up a product

2. PRODUCT DESCRIPTION

Name — Product Implementation

Purpose — A Product Implementation is an adaptable implementation of a set of work product families. An application engineer must be able to derive members of a work product family by adapting the Product Implementation mechanically based on the work product family's decisions in an Application Model.

Content — A Product Implementation consists of the following parts:

- ***Adaptable Components.*** An implementation of each work product family's Component Design. Adaptable Components include software, documentation, and verification/validation components that are adapted based on the work product family's Decision Model (see Section DE.3.1.1). Adaptable Components may be derived from Legacy Product.
- ***Generation Procedure.*** An implementation of a Generation Design for selecting, adapting, and composing Adaptable Components into draft work products that satisfy the needs of the targeted project as expressed by decisions in an Application Model (see Section DE.3.1.2).

  There is a Generation Procedure per work product family. Each work-product-specific Generation Procedure corresponds to a particular set of Adaptable Components in a Product Implementation.
- ***Organization Structure.*** A grouping of all the Adaptable Components that make up a Product Implementation (i.e., for all work product families in the domain). The grouping supports a view of all Adaptable Components as a cohesive set of domain-specific reusable work products. The grouping must be consistent with all Generation Procedures.

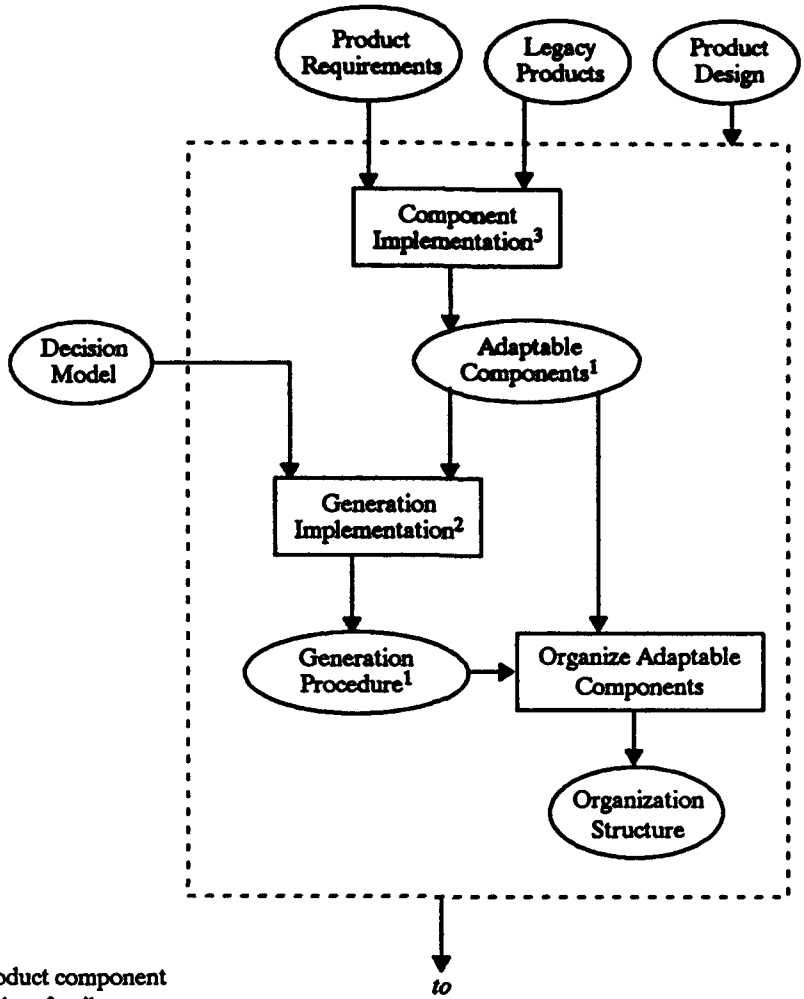
Verification Criteria — The Product Implementation correctly constructs existing or envisioned members of the work product family as specified in the Product Design.

3. PROCESS DESCRIPTION

The Product Implementation Activity consists of the three steps shown in Figure DE.3.1-1.

3.1 PROCEDURE

Follow these steps for the Product Implementation Activity.



[1] One work product component per work product family

[2] Repeated for each work product family

[3] Repeated for each work product family component as defined by the Product Architecture for each work product family

Figure DE.3.1-1. Product Implementation Process

Step: Component Implementation Activity

Action	Create Adaptable Components for the work product family.
Input	• Product Requirements • Product Design • Legacy Products
Result	Adaptable Components

Heuristics

- Derive Adaptable Components from Legacy Products.
- Ensure that the Adaptable Component satisfies and is consistent with relevant aspects of the Product Design and Product Requirements.

Step: Generation Implementation Activity

Action Automate or document a mechanical procedure by which application engineers can derive draft application work products consistent with an Application Model.

Input

- Generation Design
- Decision Model
- Product Design: Product Architecture

Result Generation Procedure

Heuristics Ensure that the Generation Procedure satisfies and is consistent with relevant aspects of the Generation Design.

Step: Organize Adaptable Components

Action Organize the Adaptable Components to facilitate reuse among all work product families.

Input

- Adaptable Components
- Generation Procedure
- Product Architecture

Result Organization Structure

Heuristics

- Develop an organization that can be mapped onto the available mechanisms in the development environment of the targeted project (e.g., a hierarchical structure can be mapped onto files and directories). If you have a reuse library or database management system available, you can organize the components using more complex mechanisms.
- Create a structure that supports definition, in Process Support Development, of procedures by which application engineers can locate, evaluate, and extract work products. A simple approach is to make each work product family a separate directory under a single root that locates the entire library. If you have several work product families of the same type, you may want to group them together.
- Use the Product Architecture as the basis for the Organization Structure. Your structure must allow use of all Generation Procedures.

- Use standard, recognized structures in the domain (e.g., for modules, an information hiding structure) to organize families. This can simplify browsing among components in complex families.

3.2 Risk Management

Risk The Product Implementation will be inconsistent with Product Requirements for a work product family.

Implication Application work products will be generated that do not satisfy the Product Requirements.

Mitigation When uncertainties arise, review the Domain Specification with domain analysts to clarify their intent. Review the Domain Implementation with other experienced engineers to identify omissions and inconsistencies. Derive test work products based on knowledge of existing or anticipated systems for review with experienced engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Specification is incomplete, ambiguous, or inconsistent.

Source Domain Specification Activity

Response Describe how the Domain Specification is inadequate, and suggest how it may be modified. Proceed with Product Implementation as far as possible with the current Domain Specification.

Contingency Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Specification Activity

Response Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback From Product Consumers

Contingency The Product Implementation does not satisfy the Domain Specification for a work product family.

Source Domain Verification Activity

Response Request clarification of the intent of the Domain Specification if necessary. Modify the Product Implementation to satisfy the Domain Specification.

Contingency	Support for selecting, adapting, and composing Adaptable Components is inefficient.
Source	Process Support Development Activity
Response	Revise the Generation Procedures based on Application Engineering experience.

DE.3.1.1. COMPONENT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Component Implementation is an activity of the Product Implementation Activity for creating an Adaptable Component. A component is any work product fragment (e.g., software, documentation, or verification/validation artifact) produced during the Application Engineering process. An application engineering work product consists of a set of components. An Adaptable Component is a representation of a family of components that satisfies a Component Design (i.e., is adaptable to specified variations). The variability of an Adaptable Component enables application engineers to extract components to form a draft application engineering work product.

1.1 Objectives

The objective of the Component Implementation Activity is to implement an Adaptable Component that satisfies a Component Design, consistent with relevant aspects of the Product Requirements and Product Architecture.

1.2 Required Information

The Component Implementation Activity requires the following information:

- Product Requirements
- Product Architecture
- Component Design
- Legacy Products

1.3 Required Knowledge and Experience

The Component Implementation Activity requires domain and software knowledge and experience in:

- Applicable standards and techniques for design, implementation, and verification of software components
- How to design, implement, and verify components of a work product family given a specification for the family
- The design and implementation of existing systems in the domain

2. PRODUCT DESCRIPTION

Name — Adaptable Component

Purpose — An Adaptable Component is a component (e.g., of software, documentation, verification/validation support) that is adaptable with respect to variations specified in the Component Design.

Content — An Adaptable Component is an implementation of a family of components. This family is defined by a Component Design, with support by portions of the Product Requirements and Product Architecture. The Component Design characterizes an Adaptable Component by specifying the permissible adaptations of the component, along with the desired characteristics of its implementation.

Form and Structure — An Adaptable Component is uniquely named and consists of two parts: an adaptability interface and a body.

A component family is characterized by a set of common capabilities and variations in those capabilities. The adaptability interface is a specification of a set of adaptation parameters that provide for the characterization and extraction of a particular instance of a component family.

The body is the sum of the potential implementations of all of the components in the family. The term "potential" is used because the parameters are sufficient to select any component family instance uniquely, but the particular implementation either may not be available or may be extracted from a representation of the family or relevant subfamily. This varies with the mechanism used for implementing adaptability in the Adaptable Component. Three common mechanisms for implementing an Adaptable Component are:

- ***Physical Separation.*** Represent members of the component set as physically distinct entities (e.g., in separate files on a computer).
- ***Target-Language Mechanisms.*** Use the language-specific facilities to represent different component set members. Ada generics, C++ templates, Interleaf effectivity control, and WordPerfect macro features are examples of target-language mechanisms for representing an Adaptable Component.
- ***Metaprogramming.*** Superimpose a language for handling variations on top of the language in which all members of the component set are expressed.

These may be used separately or in combination to implement a particular set of components.

The Booch Ada stack packages (Booch 1987) are an example of an Adaptable Component. There is a unifying concept of what a stack is and how it works. Different stack components are extracted based on decisions such as:

- ***Type.*** The type of data that can be put into the stack.
- ***Iteration.*** Whether an indexing mechanism should be available for moving forward and backward through the stack in addition to simply pushing elements onto and popping elements off of the top of the stack.
- ***Garbage Collection.*** Whether the stack should manage unused stack space dynamically for later use.
- ***Bounding.*** Whether the stack should be bounded in length.

The Booch packages use physical separation to implement 26 different types of stacks. This physical separation approach has the advantage of being simple. If a family has 10 instances, there are 10 implementations and each can be written and verified independently. Physical separation does not take advantage of similarities among the instances, however, nor does it make explicit how variabilities determine the content of each instance.

Ada provides generic packages as a standard facility that you can use to implement a code Adaptable Component whose instances differ only in the values of well-defined placeholders that can be substituted at compile time. The "type" variability of stack packages may be represented using a generic package. The placeholders are parameters defined in the adaptability interface of the Adaptable Component. This approach has the advantage of representing variabilities for the component family more compactly and uniformly; however, only a simple form of parameter substitution is supported.

A metaprogramming approach (Campbell 1989) uses a preprocessing mechanism to extract a concrete component from an Adaptable Component. This approach embeds target text fragments of a work product (e.g., code, documentation, verification/validation support) with a superimposed metaprogramming notation. The metaprogramming notation specifies how the target fragments are to be combined and adapted, based on the parameters in the adaptability interface of the Adaptable Component. Typical metaprogramming adaptations include:

- Substitution of parameter values
- Conditional inclusion of text fragments
- Repetition of text fragments
- Definition and in-line instantiation of parameterized fragments

This approach provides greater flexibility in representing a component family compactly but results in more complex descriptions. Since many implementations may be derived from a single description, domain engineers must both manually inspect that description and extract and verify representative instances. Example DE.3.1.1-1 illustrates a small fragment of a Component

Implementation for a work product family of the TLC domain. This example contains a portion of the implementation for the Adaptable Component specified in Example DE.2.2.4.2-1.

Pedestrian Lanes with Push Buttons (TLC-PL-PB-1)

Purpose: The purpose of this segment is to provide the functionality needed by the controller segment to manage the pedestrian lane indicators at the intersection considering the input provided by the trip mechanism.

Description: The segment is composed of those functions needed to turn on and off the walk and don't walk indicators present in the pedestrian lanes of the intersection.The functions needed to read and reset the pedestrian lanes' push buttons are also contained in this segment.The pedestrian lanes' push buttons assume a <COMP_Interface> interface to the traffic light indicators.

System Capabilities: ....

....

<if COMP_blinking_rate = yes then>
The TLC system shall be capable of setting the blinking rate associated with the walk-don't walk indicators at the pedestrian lanes.
<endif>

....

Example DE.3.1.1-1. Fragment of the TLC Component Implementation for the System/Segment Specification Work Product Family

Verification Criteria

- The Adaptable Components implement their specifications as defined in the Product Architecture and Component Designs.
- The Adaptable Components produce the correct variations in concrete components.
- Behaviors or constraints imposed by Product Requirements or Product Architecture on the Adaptable Component are all supported.

3. PROCESS DESCRIPTION

The Component Implementation process consists of all activities necessary for implementing an Adaptable Component to its Component Design specifications, consistent with relevant parts of the Product Architecture and Product Requirements. This involves the design, implementation, and verification of the work product family of software, documentation, or test-support components. It may involve the reengineering of existing components of work products from previously built systems. The Component Implementation Activity consists of the three steps shown in Figure DE.3.1.1-1.

3.1 PROCEDURE

Follow these steps for the Component Implementation Activity.

Step: Design the Component's Internal Structure

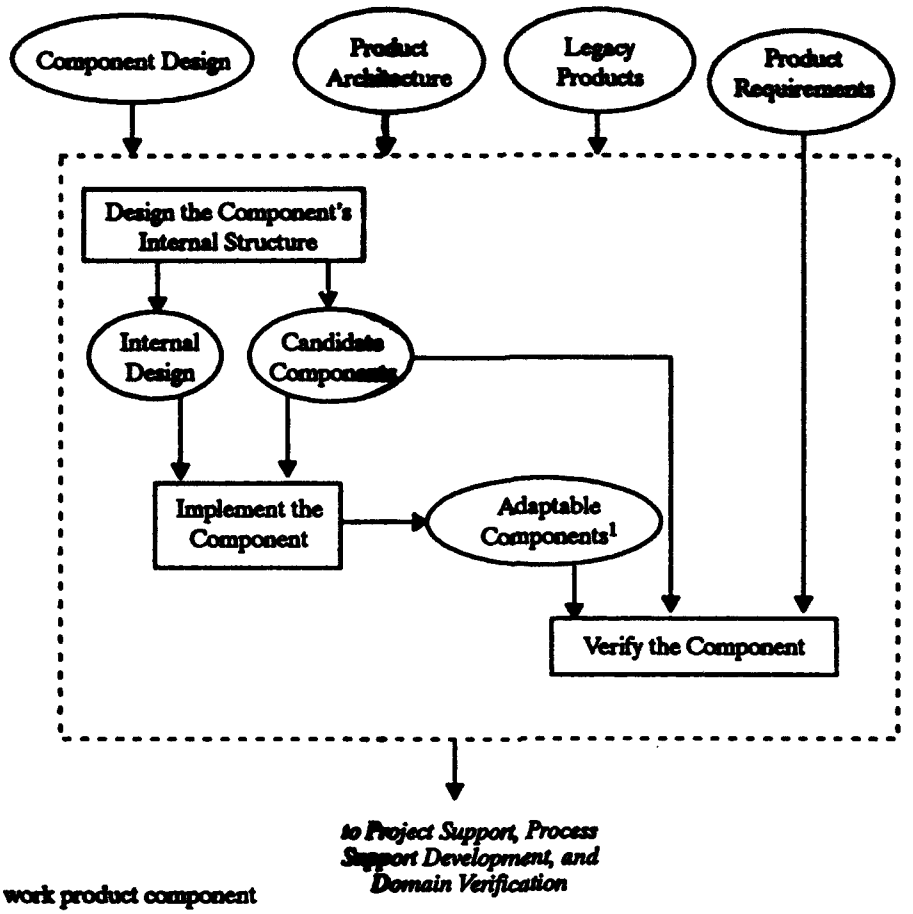


Figure DE.3.1.1-1. Component Implementation Process

Action

Create an internal design that supports the necessary variation among the members of the component set represented by the adaptable component.

Input

- Component Design
- Product Architecture
- Legacy Products

Result

- Adaptable Component: Internal design
- Candidate Components

Heuristics

- The Adaptable Components internal design must satisfy its Component Design specification. The structures of the Product Architecture may impose additional requirements on the internal structure (e.g., for concurrency or level of performance) or define constraints on how other components are used in the implementation.
- Envision how to implement required members of the component's work product family. Create structures, according to the design method used,

that characterize the required implementation. Parameterize each structure, if appropriate, with adaptability parameters that vary the structure, as required, for different members of the component family. Consider the required operations of the members.

- Determine whether a suitable component that approximates one of these desired instances is available from Legacy Products. Identify and evaluate the quality of each such component and designate it as a Candidate Component for further use. Determine which Adaptable Component variations are implicitly addressed by this selected Candidate Component.
- Consider starting with the internal designs of identified Candidate Components. Portions of several candidate components may be used collectively to implement the Adaptable Component. These components may implement different variations that will be required for the family. Characterize which instance of the family corresponds to each component (i.e., by its parameter values). Consider whether each design is sufficiently well-engineered and representative of the family, or at the least of a subfamily, to provide substantial leverage in being refined to represent other variations and the family as a whole. Consider that they may represent different subfamilies that should be structured differently. See if their designs can be unified using variations. Be sure you can still derive each of the components with an acceptable structure.
- Use adaptation mechanisms (e.g., target-language mechanisms, metaprogramming) already present in the existing work products.

Step: Implement the Component

Action Elaborate the internal design to implement the Adaptable Component.

Input
- Adaptable Component: Internal design
- Component Design
- Product Architecture
- Candidate Components
- Legacy Products

Result Adaptable Component

Heuristics
- Fill in the internal structure with the details of the implementation. Keep in mind how the adaptabilities affect the content of the parts of the structure.
- If suitable Candidate Components were used in creating the internal design, then the implementations of those components can be useful as the starting point in implementing the Adaptable Component.

- Parts of the Adaptable Component might have to be engineered from scratch if all elements of the Adaptable Component's implementation cannot be obtained from existing Candidate Components. These areas should be given much greater thought to ensure that you produce the correct content.

- Consider reengineering existing application engineering work products (e.g., Legacy Products) to increase their reusability. For example, you might want to replace arbitrary limits on data structure size with generic parameters. You should consider this if it will relax or remove constraints in your Decision Model. You should take into account any documentation or coding standards for the targeted project when you reengineer existing work products.

Step: Verify the Component

Action Verify that the Adaptable Component satisfies all relevant specifications.

Input

- Adaptable Component
- Component Design
- Product Architecture
- Product Requirements
- Candidate Components

Result Verified Adaptable Component

Heuristics

- Perform rigorous inspection of the Adaptable Component by other experienced engineers. The Component Design, as well as relevant parts of the Product Requirements and Product Architecture, should be verified as being satisfied.

- Derive representative instances of the Adaptable Component and test those instances in a conventional fashion to see if they operate correctly. One part of this activity is the creation of test-case scenarios that can be used in regression testing of the Adaptable Component when it is modified in the future. These scenarios may be made adaptable to the same parameters as the Adaptable Component itself so that a scenario can be derived for a particular test instance.

- Rederive the Candidate Components that influenced the implementation of the Adaptable Component. The original and derived instances can then be compared to see if and how they differ and whether an equivalent result can be produced.

- Use of a Candidate Component may have been based on assurances that the component received with respect to certain desired properties such as

correctness, reliability, certification, and trust (in the security sense). Note, however, that modification of the component can invalidate some of these assurances (i.e., certification and trust). It is important to verify that the desired properties are retained when the component is extracted from the resulting Adaptable Component.

3.2 Risk Management

Risk Certain combinations of adaptability are not fully supported in the Adaptable Component.

Mitigation In verifying the Adaptable Component, use bounds-coverage techniques to identify a variety of adaptability combinations in deriving test instances.

Risk The effort required to implement all specified adaptabilities for an Adaptable Component is not viable compared to that required to develop concrete components which support a single system development effort.

Implication Only a subfamily of the Adaptable Component will be available for production of systems in the domain.

Mitigation Implement the variations required for the targeted project under development. The development of these variations may require less effort than developing all possible variations and can be refined as additional needs arise. The Adaptable Component can be evolved to a completed state over several development iterations of a system or systems.

Risk Determining the value of existing components as a basis for the Adaptable Component will require too much effort (e.g., too many components to search through, too labor intensive to look through complicated components, too difficult to determine whether a component is correctly implemented).

Implication Effort to evaluate and reengineer existing components exceeds the effort to create the Adaptable Component from scratch.

Mitigation If there are too many existing components to search through to find a good baseline component, limit the amount of effort dedicated to the search, and use the best approximation that results from the limited search.

If looking through complicated components is too labor-intensive, reduce the number of components that will be reviewed. If the component is overly complicated, relying on higher-level documentation (i.e., requirements, high-level design, or testing documentation) of the component as an indicator of its worth may be beneficial. Reviewing documentation on the existing component is likely to take less effort than reviewing code.

If determining the correctness of the component is difficult, then determining correctness from previous test documentation may be sufficient. Reliance on existing components may be greater if engineers are available who developed, or at least used, the existing components.

Risk Some component set members are not present in the Adaptable Component.

Implication Application engineers will not be able to derive certain application engineering work products.

Mitigation When verifying the Adaptable Component, make sure you can derive at least a set of components equivalent to those that went into creating the Adaptable Component. (If you reengineered components, you may not be able to derive an identical set.)

Risk Modifying the baseline component may invalidate assurances of quality that the component possessed (e.g., certification).

Implication Modifying a certified component will require that the resulting Adaptable Component must pass, once again, the tests required for assurance of given properties.

Mitigation

- Concentrate effort on areas of particular concern. If the given properties are less important for the component family as a whole, treat that particular member as a special case (i.e., a component subfamily in its own right). That is, if a component family contains several members, only one of which is certified, define two Adaptable Components, one whose component family contains the certified member and another which contains all the uncertified members.
- Try to retain the essential nature of the baseline component in the Adaptable Component so that proving assurance of given properties is not an expensive process.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Component Design is incomplete, ambiguous, or inconsistent.

Source Component Design Activity

Response Describe how the Component Design is inadequate, and suggest how it may be modified. Proceed with Component Implementation as far as possible with the current Component Design.

4.2 Feedback From Product Consumers

Contingency The Component Implementation does not satisfy the Component Design.

Source Domain Verification Activity

Response Request clarification of the intent of the Component Design, if necessary. Modify the Component Implementation to satisfy the Component Design.

This page intentionally left blank.

DE.3.1.2. GENERATION IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Generation Implementation is an activity of the Product Implementation Activity for creating a Generation Procedure. A Generation Procedure is a precise description of how to derive draft application engineering work products consistent with the decisions in an Application Model for a work product family. A Generation Procedure may be automated or may take the form of a precise description that application engineers can mechanically follow to create work products.

1.1 Objectives

The objective of the Generation Implementation Activity is to create a Generation Procedure as specified by a Generation Design.

1.2 Required Information

The Generation Implementation Activity requires the following information:

- Generation Design
- Product Architecture
- Decision Model
- Component Designs

1.3 Required Knowledge and Experience

The Generation Implementation Activity requires knowledge and experience in:

- The notation used in specifying the Generation Design
- The technologies for adapting and composing components into work products

2. PRODUCT DESCRIPTION

Name Generation Procedure

Purpose — This is a procedure that an application engineer uses to create draft application engineering work products for a member of a work product family using Adaptable Components. This procedure is either implemented as a product generator or documented as a manual procedure.

Content — The Generation Procedure is a procedural description for producing an application engineering work product that satisfies the mappings of a Generation Design. The Generation Procedure describes how to select appropriate Adaptable Components, how to apply decisions from an Application Model to adapt them, and how to compose them to create the work product in final form.

Form and Structure — The form of the Generation Procedure depends on whether the procedure is automated or manual. The Generation Procedure is either implemented as an automated product generator or documented as a manual procedure to be followed by application engineers. If the manual form is chosen, then the Generation Procedure form is likely to resemble the form of a Generation Design. If the Generation Procedure is implemented in the form of a product generator, however, it will be a conventional software program.

Verification Criteria

- The Generation Procedure for a work product family can be used to produce application engineering work products that exhibit the internal organization specified in the Product Architecture.
- The Generation Procedure for a work product family can be used to produce application engineering work products that satisfy the Product Requirements.
- If a manual form is used, the Generation Procedure for a work product family clearly describes how draft application engineering work products are constructed from Adaptable Components based upon decisions contained in an Application Model.

3. PROCESS DESCRIPTION

The Generation Implementation Activity consists of the two steps shown in Figure DE.3.1.2-1.

3.1 PROCEDURE

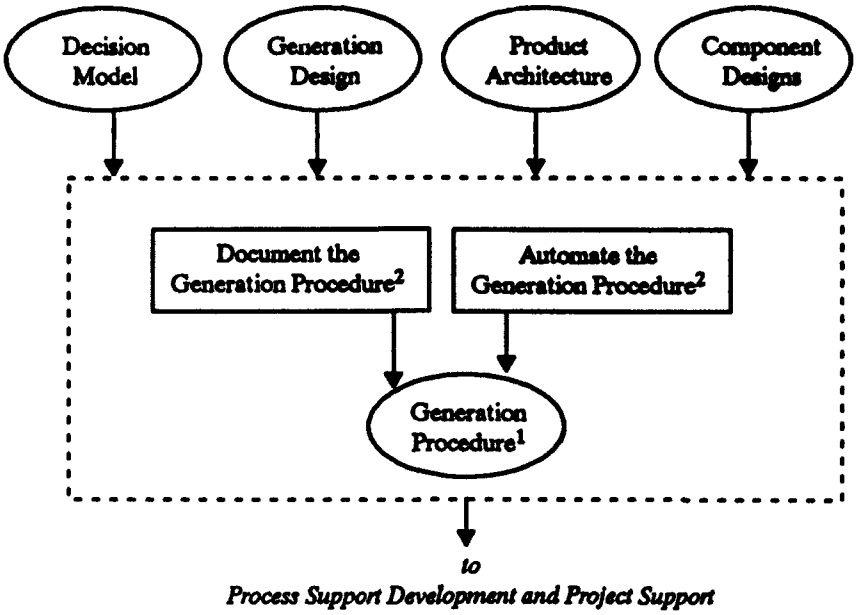
Perform one or both of the following two steps. The appropriate action depends on what automation you determine to have a significant payoff in Application Engineering.

Step: Document the Generation Procedure

Action — Document some or all of the Generation Design.

Input

- Generation Design
- Product Architecture
- Decision Model
- Component Designs



[1] One work product component per work product family

[2] Repeated for each work product family

Figure DE.3.1.2-1. Generation Implementation Process

Result Generation Procedure

Heuristics

- The Generation Design is a precise description of the required Generation Procedure. Document the procedure in a form that is usable by application engineers.
- Include a description of how components are composed to form a work product consistent with its Product Architecture.
- Include a description of how to access the decisions in an Application Model for a work product family. The Decision Model provides the organization of the decisions' conceptual schema.

Step: Automate the Generation Procedure

Action Develop automated tools that implement some or all of the Generation Procedure as defined in the Generation Design.

Input

- Generation Design
- Product Architecture
- Decision Model
- Component Designs

Result Generation Procedure

Heuristics

- If the decision-making Activity in Application Engineering is supported by automation, then the Generation Procedure must access the decisions. If the activity is not automated, then there must be an automated mechanism for providing the decisions as input to the Generation Procedure. The Decision Model provides the organization of the decisions' conceptual schema.
- If a metaprogramming technology, such as described in the Component Implementation Activity (see Section DE.3.1.1), is used to implement the Adaptable Components, then the same metaprogramming technology is used to instantiate those components. Metaprogramming technology may also be useful in implementing portions of the Generation Procedure itself.
- Creating an automated Generation Procedure is a software development task. It requires the design of the required program, implementation to that design in a programming language, testing to verify that the resulting program implements the Generation Design correctly, and documentation so that the program can be correctly modified as the Generation Design changes.
- Tools such as the UNIX make facility may be useful in automating the procedure for composing adapted components into draft work products.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Generation Design for a work product family is incomplete, ambiguous, or inconsistent.

Source Generation Design Activity

Response Describe how the Generation Design is inadequate, and suggest how it may be modified. Proceed with Generation Implementation activity as far as possible with the current Generation Design.

4.2 Feedback From Product Consumers

Contingency The Generation Procedure for a work product family does not satisfy the Generation Design.

Source	Domain Verification Activity
Response	Request clarification of the intent of the Generation Design if necessary. Modify the Generation Procedure to satisfy the Generation Design.
Contingency	A manual Generation Procedure is difficult to use.
Source	Project Support Activity
Response	Investigate new forms for conveying the Generation Procedure to the application engineers.
Contingency	The Generation Procedure cannot be used in its current form in the Application Engineering process.
Source	Process Support Development Activity
Response	Revise the Generation Procedure (e.g., improve automation or upgrade documentation) so that it can be effectively used by application engineers.

This page intentionally left blank.

DE.3.2. PROCESS SUPPORT DEVELOPMENT ACTIVITY

1. GETTING STARTED

The Process Support Development Activity is the activity of Domain Implementation for creating the Process Support component of Domain Implementation. Process Support is the infrastructure that supports the practice of Application Engineering by defining the procedures and standards by which application engineers develop applications (i.e., the Application Engineering process). It optionally provides automated mechanisms which support the effective and correct performance of the reuse-related activities of Application Engineering.

1.1 Objectives

The objectives of the Process Support Development Activity are to:

- Document policies and procedures that facilitate reuse of existing work products during activities of an Application Engineering process
- Determine the appropriate degree of automation that will support the Application Engineering process and construct the automated support

1.2 Required Information

The Process Support Development Activity requires the following information:

- Domain Definition
- Domain Specification
- Product Implementation

1.3 Required Knowledge and Experience

The Process Support Development Activity requires domain knowledge and experience in:

- How application engineers resolve issues in constructing work products in the domain
- The concepts and structures by which domain experts communicate the distinguishing features of work products in the domain

- Documenting, in a coherent and usable form, the use of conventions, policies, and procedures
- Software production processes, methods, and practices

2. PRODUCT DESCRIPTION

Name — Process Support

Purpose — Process Support is a description and explanation of the conventions by which application engineers produce work products (via activities of the Application Engineering process) and automated support for efficient performance of the Application Engineering process.

Content — Process Support consists of two parts:

- ***Application Engineering User's Guide.*** A document that guides application engineers in how to exploit reuse opportunities to produce a set of supported work products.
- ***Application Engineering Environment.*** Automated mechanisms that help the application engineers reuse work products. This includes the mechanisms that create the application engineers' view of the organization of the Adaptable Components and those tools used to access the Adaptable Components.

Verification Criteria

- Draft work products for the family can be produced using the Application Engineering Environment by following the User's Guide.
- Process Support provides the ability to access all work product families supported by Product Implementation.

2.1 Application Engineering User's Guide

Purpose — The Application Engineering User's Guide provides a detailed description of how application engineers can use the Application Engineering Process Support to exploit reuse opportunities. This guide expresses the decision-making process that application engineers follow for a domain.

Content — The Application Engineering User's Guide instructs application engineers in how to recognize reuse opportunities and how to select, adapt, and compose reusable components of a work product family to exploit reuse opportunities. This guide also designates and explains the effective use of automated mechanisms that support the process.

Form and Structure — The User's Guide should conform to your organization's standards and guidelines for documentation.

Verification Criteria — The User's Guide describes how to select, adapt, and compose components for every work product family in the Product Implementation.

2.2 Application Engineering Environment

Purpose

The Application Engineering Environment consists of all automated mechanisms, described in the User's Guide, that provide access to Adaptable Components for all domain-specific work products that application engineers reuse. The Application Engineering Environment automates the mechanical portions of the process for increased consistency within a product and less opportunities for undetected error in .

Content

The Application Engineering Environment consists of both tools that are part of the host operating system and tools developed during this activity. Together, they define a view of the Adaptable Components, along with a set of mechanisms that allow application engineers to access these components.

Form and Structure

- The tools adhere to the organization's standards and conventions for its software development environment.
- The view of the Adaptable Components provided by the Application Engineering Environment must facilitate the tasks application engineers undertake when using it. The Application Engineering User's Guide describes these tasks in detail.
- From the application engineers' perspective, the structure of the Adaptable Components is determined by the tools with which they access the structure. These tools may or may not hide the actual structure.
- The Application Engineering Environment must facilitate application engineers being able to unambiguously locate, evaluate, and extract work products.

Verification Criteria

- All automated tools described in the User's Guide exist and behave as the User's Guide states. All Adaptable Components produced during Component Implementation Activities are accessible.
- Automated mechanisms contain no residual errors.

3. PROCESS DESCRIPTION

The Process Support Development Activity consists of all activities necessary to create appropriate Process Support for an application engineering work product family. You perform this activity for each work product family, augmenting the User's Guide and Application Engineering Environment, to support the new or revised work product families. In many respects, this activity involves work which is similar to that of a conventional software development project. Furthermore, if you decide to automate some or all of the Application Engineering process (i.e., create an Application Engineering Environment), then you apply software development methods to accomplish that goal. The Process Support Development Activity consists of the two steps shown in Figure DE.3.2-1.

3.1 Procedure

Follow these steps for the Process Support Development Activity. Perform these steps in the order listed but iterate through them until you are satisfied with the work product as a whole.

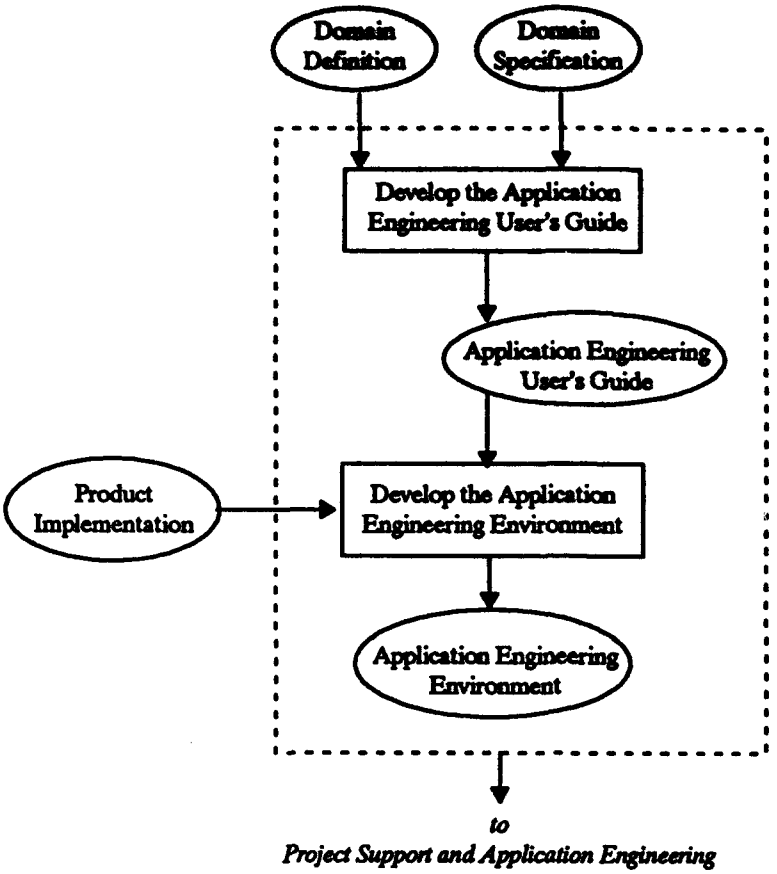


Figure DE.3.2-1. Process Support Development Process

Step: Develop the Application Engineering User's Guide

Action Create a detailed guide for application engineers which instructs them on how to perform every aspect of Application Engineering including manual steps and effective use of any automated mechanisms.

Input

- Process Requirements
- Domain Definition
- Domain Specification

Result Application Engineering User's Guide

Heuristics

- Present, in as much detail as possible, a description of how application engineers would locate and generate work products in the Application Engineering Process Support. Try to describe reuse as a series of steps that application engineers can follow. Do so in terms of any procedures and standards your organization has for performing a given activity. (Section AE describes a prototypical sequence of steps for an Application Engineering activity.)

- Provide an overview that helps application engineers understand the types of work products in the domain. You can draw on the components of the Domain Definition for this overview.
- Create scenarios that describe an Application Engineering activity involving reuse. These scenarios will help you organize reuse and incorporate it into your Application Engineering process. Use the targeted project's environment as the basis for your scenarios.
- Describe, based on the Process Requirements, the activities where application engineers can reuse work products of this type.
- Each work product family has its own Decision Model and Generation Procedures. A simple way to keep them separate is to describe each work product family in its own section.
- When you have described reuse for several work product families, you will notice similarities in the procedures. You should note these in the User's Guide as preliminary organizational standards for reuse. Some of these may come from conventions that exist in your organization, whereas others may have been introduced to facilitate reuse. Be sure that the conventions you describe do not interfere with Application Engineering.
- Describe reuse of a work product as a procedure, i.e., a set of steps to follow. Make each step of reuse as mechanical as possible. This will help you eliminate ambiguity and determine which portions you can automate.
- You can communicate certain concepts to application engineers using domain engineering work products developed for the work product family:
  - Use Product Requirements to describe the common traits of all work product family members and the characteristics of individual members.
  - Incorporate concepts from the Procedure for Work Product Creation (from Process Requirements) into the descriptions of steps for reusing work products. Ideally, you can present reuse to application engineers as a desirable, but optional, step in creating a work product.
  - Use the Decision Model to describe the decisions the application engineer must make to identify an individual work product family members. Incorporate engineering judgment and domain knowledge that domain experts use to formulate a set of answers.
  - Use the Product Architecture to describe the internal organization(s) of a member of the work product family.
  - Use Component Designs to provide detailed interface information that will help application engineers determine whether they can use a given component in their work product.

  - Use Generation Procedures to describe how to extract a member of the work product family. Note that the Generation Procedure does not account for the automation in the Application Engineering Environment. Automation may allow application engineers to use tools to perform certain steps; describe the tools, not the steps.

- You should understand (and document) the expertise expected of application engineers. This will affect the type of information you place in the User's Guide.

Step: Develop the Application Engineering Environment

Action Design, implement, and verify the automated mechanisms needed to support the Application Engineering process.

Input

- Application Engineering User's Guide
- Product Implementation
- Product Architecture

Result Application Engineering Environment

Heuristics

- The Application Engineering User's Guide specifies what aspects of the Application Engineering process are automated. Revise the User's Guide if this cannot be fully satisfied.
- Creating an Application Engineering Environment is a software development task. You must design an environment, implement that design in a programming language (or via equivalent commercially-available software technology), and test it to verify that the resulting environment implements the Application Engineering User's Guide correctly.
  - You must provide, at a minimum, enough automation to allow application engineers to access the Adaptable Components. In this step, specify other tools that you think will be a cost-effective way to simplify the steps of reuse.
  - If the User's Guide asks application engineers to browse through the Application Engineering Process Support, consider using file system directory-changing commands.
- Reduce your up-front development costs by taking advantage of available technology to automate various activities within the infrastructure. For example, there are planning and scheduling tools for project management; object-oriented databases and user interface tools that can support specifying an Application Model; testing, prototyping, and environment simulation tools for validation; simulation and dynamic assessment tools for assessment; and metaprogramming and system generation tools for product generation. However, you must also consider what resources you will

need to integrate these or other technologies into a coherent infrastructure.

- Determine how you want application engineers to view the Adaptable Components and how you want them to perform the task of locating, evaluating, and extracting them. The tools you allow them to use influence this view. At a minimum, you can use the native operating system tools to browse and manipulate the file structure specified as the Organization Structure in the Product Implementation.
- The Organization Structure of Product Implementation provides the basic structure presented by the Application Engineering Environment. You may also want to implement other structures that help application engineers locate and evaluate components. For example, if your tools include database facilities, you can provide alternate indexes.
- Identify the tools that the targeted project has decided to use as part of Application Engineering. Make them part of the Application Engineering Environment, where possible.

3.2 Risk Management

Risk The procedures described in the Application Engineering User's Guide will be hard to follow (i.e., vague, incomplete).

Implication Application engineers will have a difficult time reusing work products. This may cause excessive use of the project support staff. It may also cause application engineers to favor creating work products from scratch rather than reusing existing work products.

Mitigation Review the Process Support documentation with application engineers to see what areas of the process are incomplete, inconsistent, or ambiguous. Have them generate example work products, noting where they misinterpret or misuse the documentation.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Process Requirements work product is incomplete, ambiguous, or inconsistent.

Source Process Requirements Activity

Response Describe specifically where the Process Requirements work product is inadequate and suggest improvements. Proceed with the implementation of Process Support as far as possible while the Process Requirements are being updated.

Contingency	The Generation Procedure cannot be used in its current form in the Application Engineering process.
Source	Generation Implementation Activity
Response	Describe how the Generation Procedure needs to be changed so that it will fit within the Application Engineering process.

4.2 Feedback From Product Consumers

Contingency	The Application Engineering process is difficult to use or is too labor-intensive.
Source	Project Support Activity
Response	Identify where the problems exist and discuss, with the application engineers, ways of reducing (or eliminating) these problems (e.g., through the use of automation).

DE.4. PROJECT SUPPORT ACTIVITY

1. GETTING STARTED

The Project Support Activity is an activity of Domain Engineering for validating Application Engineering Process Support and assisting projects in its use. Application Engineering Process Support (AEPS) is the application engineering name for the Domain Implementation. To ensure that the baselined Domain Implementation is usable and effective, Project Support independently validates it from the perspective of the product and process needs of the targeted application engineering project. Project Support assists application engineers in effective use of the process and supporting materials, through delivery and installation, and consultation for the targeted project. Project Support answers questions about the process, its documentation, and its automation. Based on issues, problems, and future needs identified by application engineers, Project Support coordinates feedback to the rest of Domain Engineering for improvements in the supported process or products of application engineering. The Project Support Activity is performed for each work product family in the domain.

1.1 Objectives

The objectives of the Project Support Activity are to:

- Evaluate the effectiveness and quality of Domain Implementation for use by the targeted application engineering project
- Provide customer support to the targeted application engineering project in the understanding and use of Domain Implementation
- Provide a conduit by which the needs of the targeted application engineering project can influence domain improvements and evolution

1.2 Required Information

The Project Support Activity requires the following information:

- Domain Definition
- Domain Implementation

1.3 Required Knowledge and Experience

The Project Support Activity requires domain and software knowledge and experience in:

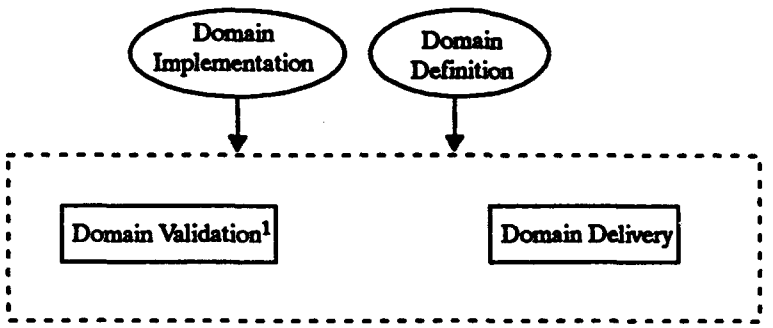
- The methods, practices, and solutions of application development in the targeted project
- Installing and evaluating software products and their documentation
- Assisting engineers and managers in the use of process documentation and automation

2. PRODUCT DESCRIPTION

The Project Support Activity produces no work products. Instead, it is a service activity to the targeted application engineering project.

3. PROCESS DESCRIPTION

The Project Support Activity consists of two steps shown Figure DE.4-1. The first, Domain Validation, is ongoing and must certify each baseline Domain Implementation as it becomes available. The second, Domain Delivery, is initiated at the beginning of each targeted application engineering project and continues until that project's termination.



[1] Activity performed once for each work product family in the Domain Implementation

Figure DE.4-1. Project Support Process

3.1 PROCEDURE

Follow these steps for the Project Support Activity.

Step: Domain Validation Activity

Action — Certify that baselined, deliverable Domain Implementation for the work product family will satisfy the targeted application engineering project's needs, as specified in the Domain Definition (available in future releases).

Input
- Domain Definition
- Domain Implementation

Result — None

Heuristics
- Review the Domain Plan and Domain Definition from the perspective of the targeted application engineering project. Ensure that the product and

process needs of the targeted project are properly represented. Advise the rest of Domain Engineering on the realistic product and process needs of the targeted application engineering project.

- Perform an independent evaluation of each baseline of the Domain Implementation as it becomes available. Evaluate whether it properly satisfies and balances the intended mix of general business objectives and specific application engineering project/customer needs.

- Perform independent validation, including extensive, scenario-based testing of the (integrated) Product Implementation and Application Engineering Environment portions of the Domain Implementation baseline. Evaluate the correctness and usability of the Application Engineering User's Guide as it relates to use of the Product Implementation and Application Engineering Environment.

- Attempt to build typical products that reflect realistic project experience on existing systems in the domain. Identify capabilities or characteristics of those products that the Domain Definition accommodates but that are not attainable with the provided Domain Implementation baseline.

- Evaluate the impact of the Domain Implementation baseline on the efficiency and effectiveness of the targeted application engineering project. Identify improvements in realistic and practical Domain Implementation usability.

Step: Domain Delivery Activity

Action Deliver Domain Implementation to the targeted application engineering project, assist with its use, and identify needed product or process improvements (available in future releases).

Input Domain Implementation

Result None

Heuristics

- Initiate an instance of this activity at the beginning of each targeted application engineering project; continue this activity until the project terminates.

- Provide copies of process documentation (i.e., the Application Engineering User's Guide) to the engineers and managers of the application engineering project.

- Install the Application Engineering Environment (and subsequent upgrades), including the Adaptable Components from the Product Implementation, for project use and check it for proper operation.

- Explain use of the Application Engineering User's Guide for understanding and performing the process of developing draft application

engineering work products. Explain the use of the Application Engineering Environment as described in the User's Guide.

- Provide consultation services to application engineers as they perform Application Engineering. Consulting requires extensive domain knowledge to answer application engineers' questions accurately and fully. Consultants should be knowledgeable in all aspects of Domain Implementation. There also needs to be a core of expert consultants who are sufficiently familiar with other domain engineering work products to provide complete, detailed, in-depth information, rationales, and assistance when complex problems are encountered by a project. In small organizations, the entire domain engineering team may be called on as a consulting resource.

- In response to the delivery services provided, application engineers will identify problems, improvements, and future needs that Domain Engineering should consider for possible action. Some of these ideas will relate directly to meeting customers' needs while others will relate to how efficiently application engineers can use the process and associated domain assets. Properly record and communicate these ideas and their motivations to the rest of Domain Engineering as feedback from application engineering. This is a key part of Project Support and is essential to continual project responsive improvement of a domain.

3.2 Risk Management

Risk The needs of a particular application engineering project will conflict with a simple interpretation of prescribed standards and procedures.

Implication The project will be forced to work in conflict with that interpretation and to be less effective and efficient.

Mitigation Try to interpret standards and guidelines flexibly so that they best fit the needs of the targeted project. Be aware of variations in the Process Support, particularly in environment installation, that support different needs. Tailor consultation to the targeted project's needs.

Risk Changes in the circumstances of a project may conflict with the previous interpretation of prescribed standards and procedures.

Implication The project will be forced to work around obsolete support and will be less effective and efficient than necessary.

Mitigation Reconsider the support given to a project whenever circumstances change significantly. Be prepared to adjust the environment and consulting advice to fit current needs better.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 FEEDBACK TO INFORMATION SOURCES

Contingency	Application engineers are having difficulty using the Application Engineering process or Domain Implementation to develop draft work products.
Source	Process Support Development Activity
Response	• Suggest better ways to the project for performing the process within the prescribed standards. • Document the nature of the difficulties and suggest improvements in the prescribed process or in its documentation or automation.
Contingency	Particular, noncommon customer requirements cannot be expressed in an Application Model.
Source	• Domain Definition Activity • Process Requirements Activity • Process Support Development Activity
Response	• Identify unrecognized domain variations that application engineers need. • Suggest to the project how it can best work around current limitations.

4.2 FEEDBACK FROM PRODUCT CONSUMERS

None

This page intentionally left blank.

AE. APPLICATION ENGINEERING OVERVIEW

1. GETTING STARTED

Application Engineering is a Synthesis process for creating and supporting an application product that satisfies specified customer requirements. A product is represented by a set of associated work products that result from analysis of those requirements. Application Engineering is characterized by a comprehensive life-cycle process for the management, analysis, production, and support of a product as a set of work products that offer opportunities for reuse.

In an organization practicing opportunistic Synthesis, the Application Engineering process combines activities that create work products from scratch with activities that reuse existing work products, in whole or in part, available from Application Engineering Process Support. Activities involving reuse focus on application engineers resolving standardized decisions relating to the work product to determine whether members of existing work product families will satisfy Application Engineering needs. Based on these decisions, application engineers obtain useful work products from Application Engineering Process Support and tailor them into work products that completely meet customer requirements.

1.1 Objectives

The objectives of Application Engineering are to:

- Understand the needs of customers, balancing concerns of cost versus value, to produce a product that fulfills those needs most effectively
- Organize and direct resources for the production and support of the product
- Produce software and documentation that support the delivery and use of the product
- Maximize productivity in creating work products through appropriate use of reusable components provided by Application Engineering Process Support

1.2 Required Information

Application Engineering requires the following information:

- Application Engineering Process Support
- Customer requirements

1.3 Required Knowledge and Experience

Application Engineering requires domain, business, and software knowledge and experience in:

- The problems that the products in the domain are intended to solve and the engineering tradeoffs to be considered in creating a viable solution
- Understanding and interpreting customer requirements and developing applications that satisfy those requirements
- The management, production, and delivery of software work products
- How to use the Application Engineering Process Support to develop an application
- The information required by the Application Engineering process employed by the project

2. PRODUCT DESCRIPTION

The product of Application Engineering is a set of work products as determined by the process being followed. Projects use their organization's normal software development process producing familiar work products. Among the work products that an Application Engineering process might produce are software requirements documents, software system architectures, and software partitioned into separately developed components.

3. PROCESS DESCRIPTION

The Application Engineering Activity consists of a set of activities specific to the software development process your organization uses. You perform the same set of activities whether or not you practice opportunistic Synthesis. Within activities, however, you seek opportunities to reuse existing work products; thus, how you perform activities may differ when you incorporate opportunistic Synthesis into your software development.

Figure AE-1 shows an ESP-derived and prototypical process for Application Engineering. It is recognizable as a conventional waterfall software development process and therefore straightforward to tailor to the needs of your organization. Reuse within this process is entirely localized to focus on rapid production of draft individual application engineering work products.

The activities of the Application Engineering process are organized into three classes. (In the ESP model, activities are grouped into subclasses, and the subclasses are grouped into classes.) The classes are as follows:

- ***Project Management***. These management and administrative activities support planning, monitoring, and controlling the project. These activities include the following:
  - Plan and initiate those activities required to initially start the project, produce the project's master plan and schedule, estimate overall cost, and acquire necessary resources.
  - Determine the objectives, alternatives, constraints, and success criteria for a cycle.
  - Analyze the risks inherent in the cycle's objectives and suggest risk aversion strategies.
  - Produce the Cycle Development Plan which organizes and provides the details for the activities which will be performed during the cycle.

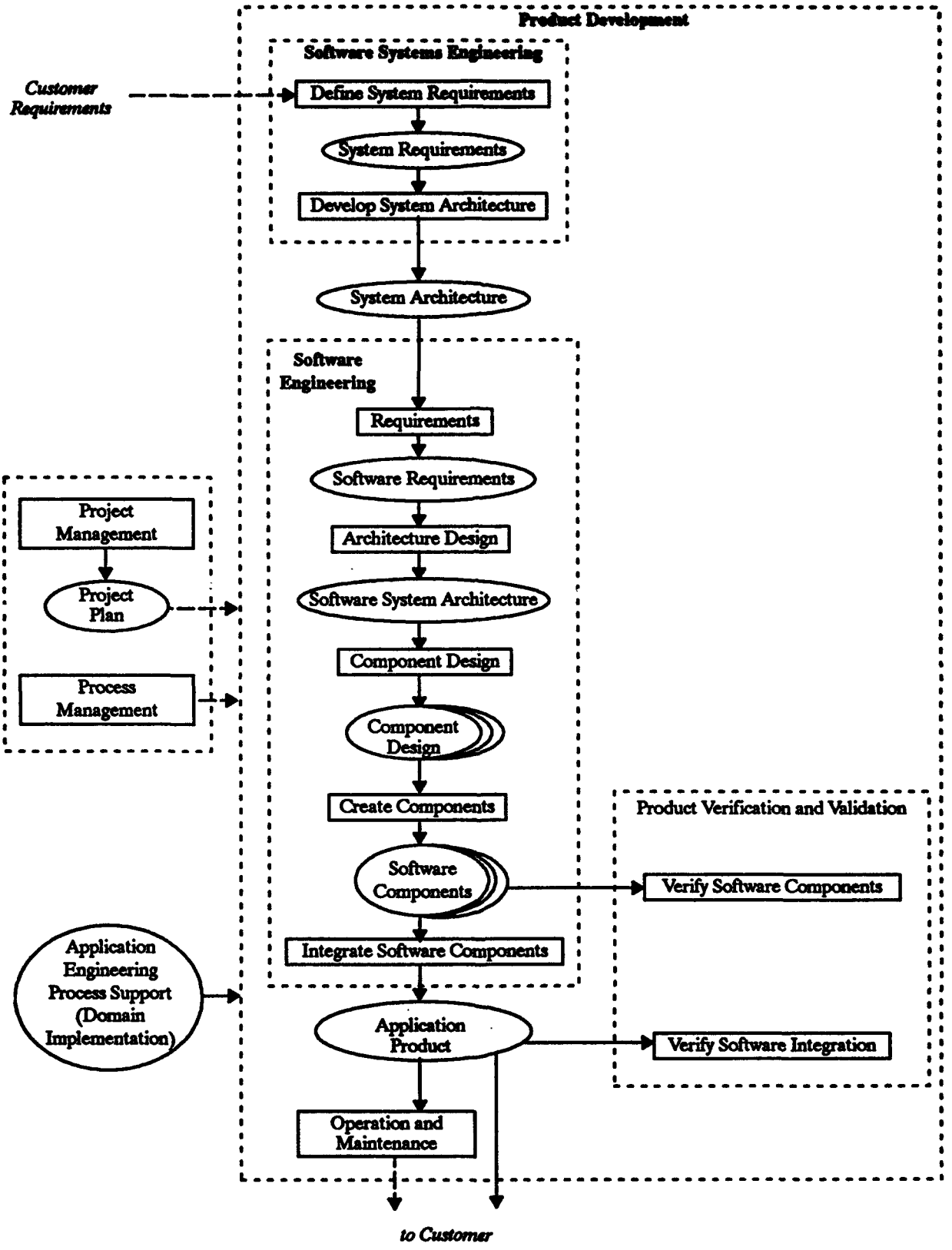


Figure AE-1. A Prototypical Application Engineering Process

- Initiate the cycle development activities and observe the project's progress and current status.

- ***Process Management.*** These activities manage the technical control and quality of the delivered product. These activities include the following:
  - Define quality standards and ensure that the work products conform to these standards.
  - Assess the project's current process, identify areas that need process improvement, and improve the process.
  - Baseline each increment of the product(s) and control any changes to the product(s).
  - Control the editing, production, and distribution of project documentation.
  - Create and maintain the software systems that the application engineers use to develop and test the product(s).
  - Develop, validate, and administer the training program for the developers and users of the system.

- ***Product Development.*** Activities in this class specify, design, implement, verify, and maintain the end product. Product Development consists of the following activities and subclasses:
  - ***Software Systems Engineering.*** Activities in this subclass provide the system's specification and design. It consists of the following activities:
    - -- ***Define System Requirements.*** This activity collects, integrates, specifies, relates, and organizes the project's needs and objectives to provide the foundation for system design and implementation. These needs and objectives are expressed in the system requirements.
    - -- ***Develop System Architecture.*** This activity identifies a system-level architecture, including hardware and software components, that satisfies the system requirements within budget and schedule constraints. It documents this architecture in the system architecture.
  - ***Software Engineering.*** This subclass includes all activities related to software requirements, software design, and software implementation:
    - -- ***Requirements.*** This activity analyzes the system requirements and architecture to derive software requirements that further clarify the allocated system requirements.
    - -- ***Architecture Design.*** This activity derives a software system configuration that performs the required functions at the necessary level of performance and reliability. This configuration is documented in the software system architecture.
    - -- ***Component Design.*** This activity refines the software system architecture to identify the lowest level components and their interfaces. It produces a detailed design document, the Component Design.

-- ***Create Components.*** This activity develops and documents the source code for each software component in accordance with the detailed design provided in Component Design.

-- ***Integrate Software Components.*** This activity integrates the software components into larger software configuration items through an incremental process of adding software components to grow the core of software into the finished system.

- ***Product Verification and Validation.*** This subclass includes all activities related to the verification and validation of the system and software requirements.

  -- ***Verify Software Components.*** This activity verifies the software components.

  -- ***Verify Software Integration.*** This activity verifies the integrated software components up to the final system.

- ***Operation and Maintenance.*** This subclass includes all activities related to system installation and operational support.

  Provide Operational Support. This activity provides technical assistance in response to customer requests and performs regular maintenance to ensure that the finished system performs efficiently.

For brevity, this process concentrates on software products, omitting supporting deliverables (e.g., user manuals). A complete process would show supporting deliverables and the activities that produce them.

3.1 Procedure

In this opportunistic Synthesis process, reuse occurs while performing certain Application Engineering activities—specifically, those yielding work products for which Domain Engineering has implemented application engineering work product families. Thus, application engineers perform the activities of the process shown in Figure AE-1.

The way each activity is performed depends in part on whether Domain Engineering has provided work product families that support reuse within that activity. The consequences of Domain Engineering support for opportunistic reuse are described in the following five representative steps for creating a work product. An application engineer performs these steps to create a work product whether or not reuse is supported. However, when reuse is supported, application engineers perform these five steps somewhat differently. The heuristics for each step include descriptions of differences due to reuse.

Step: Define Success Criteria for the Work Product

Action	Determine the success criteria and characteristics that the work product must have to be considered complete.
Input	Required information of the application development process relevant to the work product. This information would include customer requirements or appropriate work products from preceding activities.

Result Success criteria

Heuristics

- Review the company's policies and procedures to determine the requirements and restrictions relevant to this particular type of work product. Define your success criteria as meeting relevant policies and procedures.
- Review customer requirements or appropriate work products from preceding activities to determine the success criteria relevant to this particular type of work product. Define your success criteria as meeting relevant customer requirements.
- Determine reviewers of the work product who can provide insights and alternative viewpoints relevant to the work product's subject matter.

Step: Develop Internal Structure

Action Design the internal structure of the work product based upon the needs of the application engineering project.

Input

- Success criteria
- Application Engineering Process Support: User's Guide for the type of work product

Result Work product's internal structure

Heuristics

- Select from the work product families (described in the User's Guide) for an existing family that might directly contribute to this activity's work product. For example, if you are producing a requirements specification document, search for a requirements specifications document family that could be used in producing the new specification.
- The internal structure depends upon the work product type. For a document, this might be an outline; for a plan, a set of milestones; for source code, an internal design.
- If you cannot find a suitable member of the work product family, you must create the structure.
- Modify or extend the structure to suit specific needs.

Step: Produce the Work Product

Action Produce the work product by filling in its internal structure.

Input

- Success Criteria
- Application Engineering Process Support: User's Guide for the type of work product
- Work product's internal structure

Result Work product

Heuristics

- The User's Guide describes a procedure for creating an Application Model (i.e., Application Modeling). This procedure identifies what decisions to make and how to apply those decisions to select a member of the work product family.
- Develop the desired work product by applying the decisions of the Application Model to select and tailor work product component families into components that fill in parts of the work product's internal structure.
- If you cannot find suitable component family members, you must create the work product.

Step: Verify the Work Product

Action Verify that the work product meets the success criteria.

Input

- Work product
- Success criteria

Result None

Heuristics For each success criterion, determine whether the completed draft of the work product satisfies the success criteria. If it does not, alter the draft.

Step: Baseline the Work Product

Action Provide a copy to configuration management for baselining. Produce the necessary reports characterizing the work product and its cost.

Input Work product

Result None

Heuristics Include information in standardized reports characterizing the extent that work-product reuse occurred. To the extent possible, tell the domain engineers what kinds of work product components were needed and not found, or will be needed in the subsequent (follow-on) activities of this particular application project.

3.2 Risk Management

The risks associated with Application Engineering depend on the specific software development process followed by the application engineers. Nonetheless, the following risks are (more or less) independent of the particular Application Engineering process.

Risk Application engineers will select totally inappropriate (unknown to them) members of a work product family.

Implication Modifying or extending the selected family member so that it suits the application engineer's needs will be time-consuming, thereby making the cost of developing a work product more expensive than developing it from scratch.

Mitigation Have the Project Support staff initially work together with the application engineers to help them understand how to effectively express their needs in an Application Model.

Risk Application engineers will inadvertently invalidate certain desired properties of a family member when they modify or extend the member to suit their needs.

Implication Work product verification costs will increase because application engineers will have to perform more extensive testing than normally necessary to ensure that all properties of the family member are still valid.

Mitigation Have the Project Support staff available to review the impact of proposed modifications or extensions not addressed in the User's Guide.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency Application Engineering Process Support does not provide the information needed to decide which families and which components are useful. Consequently, application engineers do not find reusable components or spend excessive time finding them.

Source Domain Engineering

Response

- Request immediate (verbal) clarification from the project support staff.
- Characterize the additional information needed to assess the reuse potential of a family and its members. Proceed with the Application Engineering process using the Application Engineering Process Support to the extent possible given the inadequacies.

Contingency Application engineers are unable to identify work products, within the domain, that are relevant to the current application development project.

Source Domain Engineering

Response

- Identify what kinds of families need to be developed.
- Proceed with the Application Engineering process using the Application Engineering Process Support to the extent possible given the inadequacies.

Contingency The Application Engineering Process Support provided is incomplete or deficient.

Source	Domain Engineering
Response	Describe the inadequacies in the Application Engineering Process Support. Indicate which sections are incomplete or deficient. These may include: missing work product families, an incomplete User's Guide, errors in the User's Guide, improperly described Adaptable Components, malfunctioning generation procedure(s), and bugs in interface software provided by Domain Engineering.

4.2 Feedback From Product Consumers

Contingency	The customer requests new or modified capabilities for the system.
Source	Customer
Response	• Build a new version of the system. • Reject the suggestions as out of scope.
Contingency	The customer identifies system anomalies, changes to the target environment, inadequate system performance, or inadequate reliability.
Source	Customer
Response	• Correct system anomalies, accommodate changes to the target environment, tune the system. • Reject changes as out of scope. • Ask Domain Engineering to make necessary changes in the domain.

This page intentionally left blank.

PART LEV: LEVERAGED SYNTHESIS

This page intentionally left blank.

OV. OVERVIEW OF A LEVERAGED SYNTHESIS PROCESS

This part of the guidebook presents a leveraged Synthesis process. This is a process suitable for an organization that has targeted the goals associated with the leveraged stage of reuse capability, as defined by the RCM. To help you understand whether a leveraged process suits your organization, this section discusses the assumptions that underlie the process and the nature of software development when using the process.

1. UNDERLYING ASSUMPTIONS

The RCM defines four stages of reuse capability implementation and characterizes each stage by a set of goals. The leveraged Synthesis process described in this part of the guidebook was designed to fit the goals associated with the leveraged stage as described in the Fundamentals section of the overview to this guidebook (Section OV.2). To adopt this process, an organization must have targeted those goals as a minimum. Your organization may choose to adopt the leveraged process even if it has the potential to attain goals associated with the more advanced anticipating stage of reuse capability implementation. However, this process does not depend upon nor require your organization to attain any of the more ambitious goals associated with the anticipating stage.

The goals associated with the leveraged stage introduce requirements for a process to be used by organizations targeting that stage. The Synthesis process described in this part of the guidebook is one example of a process that an organization targeting the leveraged stage could adopt. It differs from other such processes because it reflects assumptions about an organization's circumstances that extend those implied by the RCM goals for the leveraged stage. Section 2.3 in Part Syn of the guidebook describes characteristics common to all Synthesis processes, particularly that the process comprises iteratively cooperating Domain Engineering and Application Engineering subprocesses. Additional assumptions that distinguish the leveraged Synthesis process are as follows:

- ***Assigned managers and engineers have sufficient knowledge and expertise in the domain.*** This assumption means that business-area management is willing and able to commit people to this endeavor who have the needed knowledge and expertise to ensure its success. The people assigned are the experts in this business area; they understand both the 'whys' and 'hows' of the existing systems and work products upon which the domain is based. They also understand the future of the business area and the organization's objectives for it.

- ***The role of Domain Engineering is to standardize the practice of Application Engineering.*** This assumption means first, that Domain Engineering standardizes the scope of the domain to a specified product family. Each member of this family is a product (i.e., a software system and all associated work products) that the organization will be able to produce for a given customer.

Secondly, it means that Domain Engineering concurrently engineers a standardized process of Application Engineering that is tailored to the effective and efficient production of individual products. To the degree that the process can be made more effective or efficient, Domain Engineering may enhance it with automated support. Business objectives act as constraints that limit the scope of the domain, making process and product family standardization feasible.

- ***Instituting a standardized Application Engineering process requires Domain Engineering to support projects with automation, documentation, training, and consulting.*** Domain Engineering supports application engineering projects not only by providing them with appropriate reusable assets but also by supporting their overall effort to produce a customer product with those assets. Domain Engineering provides automated support for the prescribed Application Engineering process; documents the process sufficiently for effective use; provides training in efficient use of the process; and consults with each project on how best to use the process.

- ***Reuse focuses on system-level variations in problems and solutions.*** Standardization of a product family involves the identification, preferably at the system-level for maximum leverage, of commonalities and variations that characterize such products. Commonalities provide the basis for potential leverage but depend on accommodation of variations for sufficient flexibility. Variations correspond to either problem-level or solution-level decisions that application engineers need to make in order to produce a particular product. The essence of the Application Engineering process is making and evaluating required decisions and using those decisions in mechanically generating corresponding work products.

- ***An effective domain depends on consistent objectives and schedules which requires coordinated planning and staffing of domain engineering and application engineering projects.*** Traditionally, application engineering projects have operated autonomously and in isolation from similar projects. This assumption, in contrast, means that projects are viewed as agents of a domain. Projects attempt to satisfy customer needs by projecting the capabilities supported by the domain and stimulating domain evolution when the domain is inadequate. Project staffing comes, at least partially, by allocation from the resources of the domain. Conflicting needs of a project with Domain Engineering objectives or schedule, or with other projects, must be resolved through coordinated management of the domain.

- ***Domain viability depends on iterative domain evolution.*** A viable domain is one that supports rapid delivery of high-quality products that meet customer needs. Since customer needs are diverse and changing, as is technology, the domain must evolve to reflect those changes. The primary insights for domain evolution come as a result of initiation of projects for new customers and feedback from existing projects.

2. SOFTWARE DEVELOPMENT WITH A LEVERAGED SYNTHESIS PROCESS

This section is a general overview of the leveraged Synthesis process presented in this part of the guidebook. It gives an overall feel for software development and management as practiced by an organization with leveraged reuse capabilities, without elaborating every activity or every possible variation or interpretation of the process. As Figure OV.2-1 in Part Syn depicts, Domain Engineering and Application Engineering activities, and the interactions between them, are defining aspects of any Synthesis process. This section describes this leveraged Synthesis process from three perspectives:

- A business-area organization, which encompasses a domain engineering group and one or more application engineering projects
- A domain engineering group, which is responsible for process and product family standardization across a domain
- An application engineering project, which is responsible for producing and delivering a system to a particular customer

This section concludes with a brief scenario of how Domain Engineering and Application Engineering might typically work and interact.

2.1. Organizational Perspective

In leveraged Synthesis, a business-area organization sets explicit, long-term objectives that direct capital investment in process and product standardization. The scope of this investment is a family of systems that represents the strategic focus of the organization. This scope indicates the organization's market, or customer base, where it expects to find its future business opportunities. Reuse at the leveraged stage is a key mechanism for attaining strategic business-area or organizational mission objectives.

Within this framework, a domain is the organizational vehicle for all domain-specific software development which includes both Domain Engineering and Application Engineering efforts (see Figure OV-1). Domain Engineering focuses on developing, instituting, and evolving effective process/product standardization suited to the needs of the business. Client application engineering projects are formed in the domain whenever appropriate opportunities arise to build systems that fall within the domain's scope. As the organization recognizes changing customer needs or technology, Domain Engineering evolves the domain to reflect those changes.

The benefit of this approach is higher application engineering productivity, accompanied by more predictable schedules and costs, more consistent quality in the product, and more rapid responses to diverse and changing customer needs. Domain Engineering funding will usually come from a combination of organizational capital and client project receipts. The guiding principle is systematic investment in process/product standardization as a means to leverage expertise and increase productivity in the delivery of systems to customers. Investments that promise long-term leverage are generally to be preferred over short-term profits.

2.2. Application Engineering Perspective

Application Engineering under leveraged Synthesis is more similar to a prototyping process than to a conventional software development process, but is oriented nonetheless to producing production-quality products. Application engineers express customer requirements in the form of a unified model of a problem and its solution. This model allows application engineers to express the decisions that they must make, which correspond to supported variations in systems within the domain's scope. The resulting Application Model is a medium for evaluating the corresponding software product and for mechanically generating associated work products, including code and documentation, using reusable components. As requirements are better understood or change, the application engineers modify the Application Model, reevaluate it, and generate a revised product.

2.3. Domain Engineering Perspective

The goal of Domain Engineering under leveraged Synthesis is to provide application engineering projects with a capability both to create an accurate model of any product that is within the designated

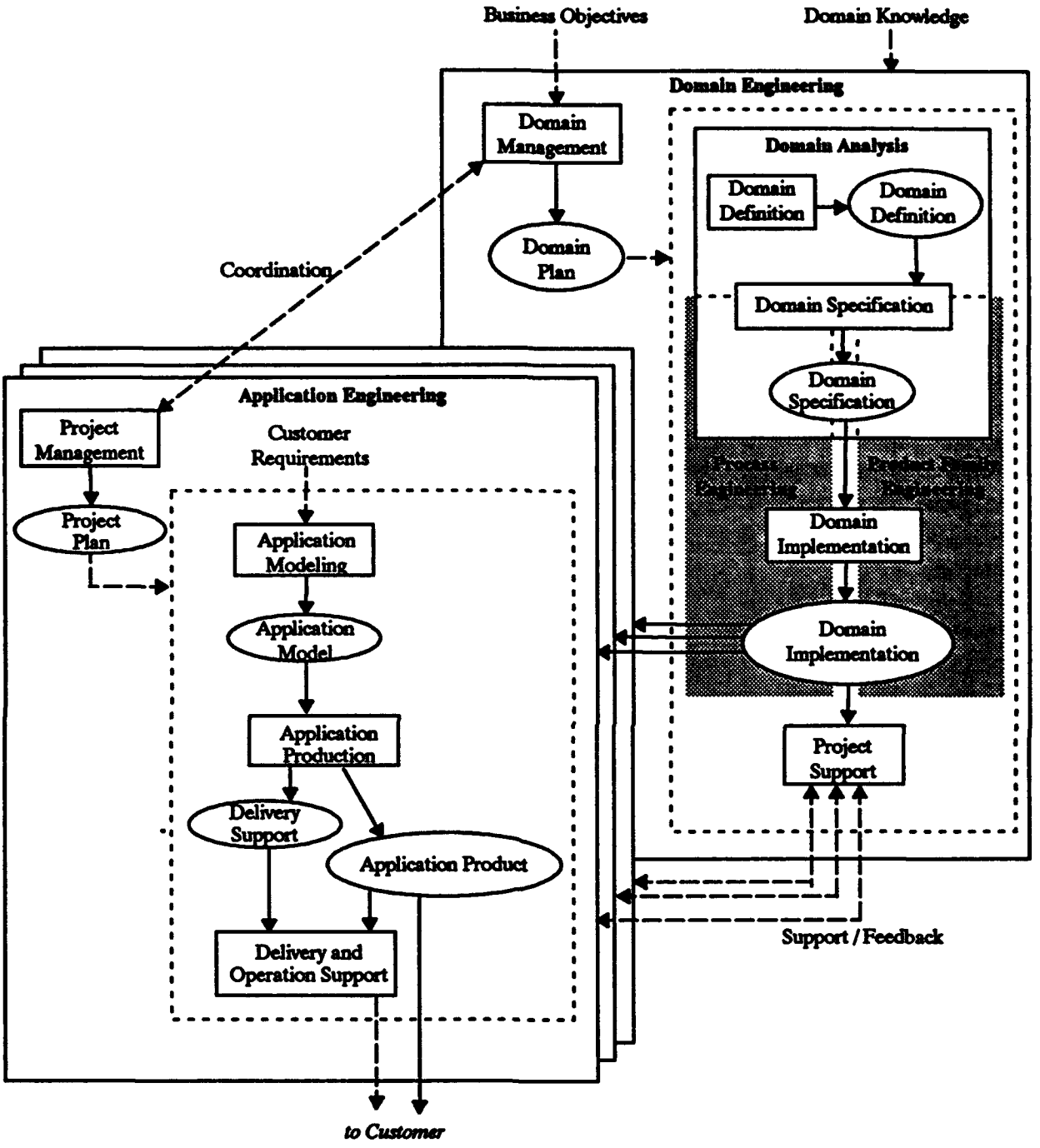


Figure OV-1. Interaction Between Application Engineering and (Simplified) Domain Engineering

scope of the domain and to generate a valid software system product consisting of all required work products corresponding to the model. Domain Engineering accomplishes this goal through concurrent process and product family engineering for the domain in accordance with specified domain objectives. Additionally, this goal requires continual evolution of the domain as customer requirements and technology change. The challenge is to manage and accomplish this evolution in a timely and systematic fashion. Although existing systems are a valuable source of reusable components particularly

when a domain is first being established or expanded, in leveraged Synthesis, the ability to evolve the domain rapidly as needs change is key to the long-term success of the business.

2.4. An Example Scenario

As an example, consider a hypothetical process where a business-area organization has established a domain for one of its business lines. The organization initiates Domain Engineering to formalize the domain in light of the organization's business objectives and its past experience building systems for this business. The organization chooses the domain engineering staff to include some of the most experienced managers and engineers in the organization so that the resulting, concurrently-engineered process and product family standardization will reflect the best available knowledge and expertise about the problems to be solved and how valid solutions are created. Domain Engineering rapidly proceeds through several iterations in which they achieve a consensus on the scope, commonalities, and variabilities of the domain, the decisions that application engineers must make to build any particular system, the requirements, design, and implementation of the supported product family, and the preferred Application Engineering process and its automated support. The resulting Application Engineering Process Support, including reusable assets, documentation, and automation, is baselined for subsequent use by client application engineering projects.

When a customer is found who needs a system that fits within the scope of the domain, the organization forms an application engineering project. The project manager and lead engineer have been assisting in Domain Engineering and understand the needed system in domain-specific terms. Domain Engineering dedicates some of its project support staff to assist the project in using the domain most effectively. From customer information and discussions, Application Engineering creates an Application Model that seems to correspond to what the customer needs. Application Engineering evaluates the model and explains it to the customer. Based on this interaction, Application Engineering discovers inaccuracies and revises the model. A software product is then generated. Based on reviews and exploratory use of the product, Application Engineering discovers additional shortcomings, leading to further revision of the model and generation of a modified product.

In evaluating the model or generated product, Application Engineering discovers a need that cannot be properly modeled. In consultation with Domain Engineering, they decide that the need is in fact within the proper scope of the domain and work starts on evolving the domain accordingly. In the meantime, Application Engineering completes the model and generates a product that approximates a solution to the understood need. Based on discussions with the customer, Application Engineering finds a workaround that achieves some of the unmet need and extends the model and product accordingly. When Domain Engineering has completed the agreed-upon domain evolution, Application Engineering again revises the model, using the added capability that is now available, to create an enhanced solution for the customer.

As the organization initiates new application engineering projects or as the needs of existing client projects change, Domain Engineering revises its consensus on the scope and substance of the domain. Each revision results in a new baseline of Application Engineering Process Support for use by application engineering projects. This evolution continues until the domain is no longer a viable business base.

This page intentionally left blank.

DE. DOMAIN ENGINEERING OVERVIEW

1. GETTING STARTED

Domain Engineering is an activity of a Synthesis process that creates and supports a standardized Application Engineering process and products in a business area. Domain Engineering is a comprehensive iterative life-cycle process with management, analysis, implementation, and support activities for a product family of primary value to a business-area organization.

1.1 Objectives

The objectives of Domain Engineering are to:

- Organize and direct resources to accomplish the business objectives of an organization
- Define the nature, extent, and substance of a product family that complements those business objectives
- Provide leverage by which projects within a domain can deliver a product more effectively, predictably, and reliably

1.2 Required Information

Domain Engineering requires the following information:

- Domain knowledge
- Business objectives

1.3 Required Knowledge and Experience

Domain Engineering requires domain and software knowledge and experience in:

- The needs that motivate systems in the domain (i.e., application engineering work products)
- The environments in which the systems in the domain will operate
- How the systems in the domain are built
- How application engineering projects in the domain are managed

2. PRODUCT DESCRIPTION

Domain Engineering creates four work products: Domain Plan, Domain Definition, Domain Specification, and Domain Implementation. Domain engineers evolve these products in subsequent iterations of Domain Engineering to support future projects, consistent with organizational business objectives.

2.1 Domain Plan

Purpose	A Domain Plan (see Section DE.1) describes a plan for domain evolution and defines the tasks and resource allocations for domain development increments.
Verification Criteria	The expected needs of the projected product market in the business area are sufficient to compensate for projected costs and risks of domain development.

2.2 Domain Definition

Purpose	A Domain Definition (see Section DE.2.1) defines the informal scope and orientation that characterize a viable domain.
Verification Criteria	The Domain Definition captures sufficient information to allow domain engineers to describe any existing or potential system in the domain.

2.3 Domain Specification

Purpose	A Domain Specification (see Section DE.2.2) formalizes expert knowledge of how to express problems in the domain and how to create corresponding solutions for the problems.
Verification Criteria	The Domain Specification precisely expresses the domain as captured in the Domain Definition.

2.4 Domain Implementation

Purpose	A Domain Implementation (see Section DE.3) is an implementation (with documentation and automated support) of the Application Engineering process and product family for the domain, as prescribed by the Domain Specification.
Verification Criteria	The Domain Implementation provides the standardized Application Engineering process and product family described in the Domain Specification.

3. PROCESS DESCRIPTION

Domain Engineering is an interaction among the four steps shown in Figure DE-1.

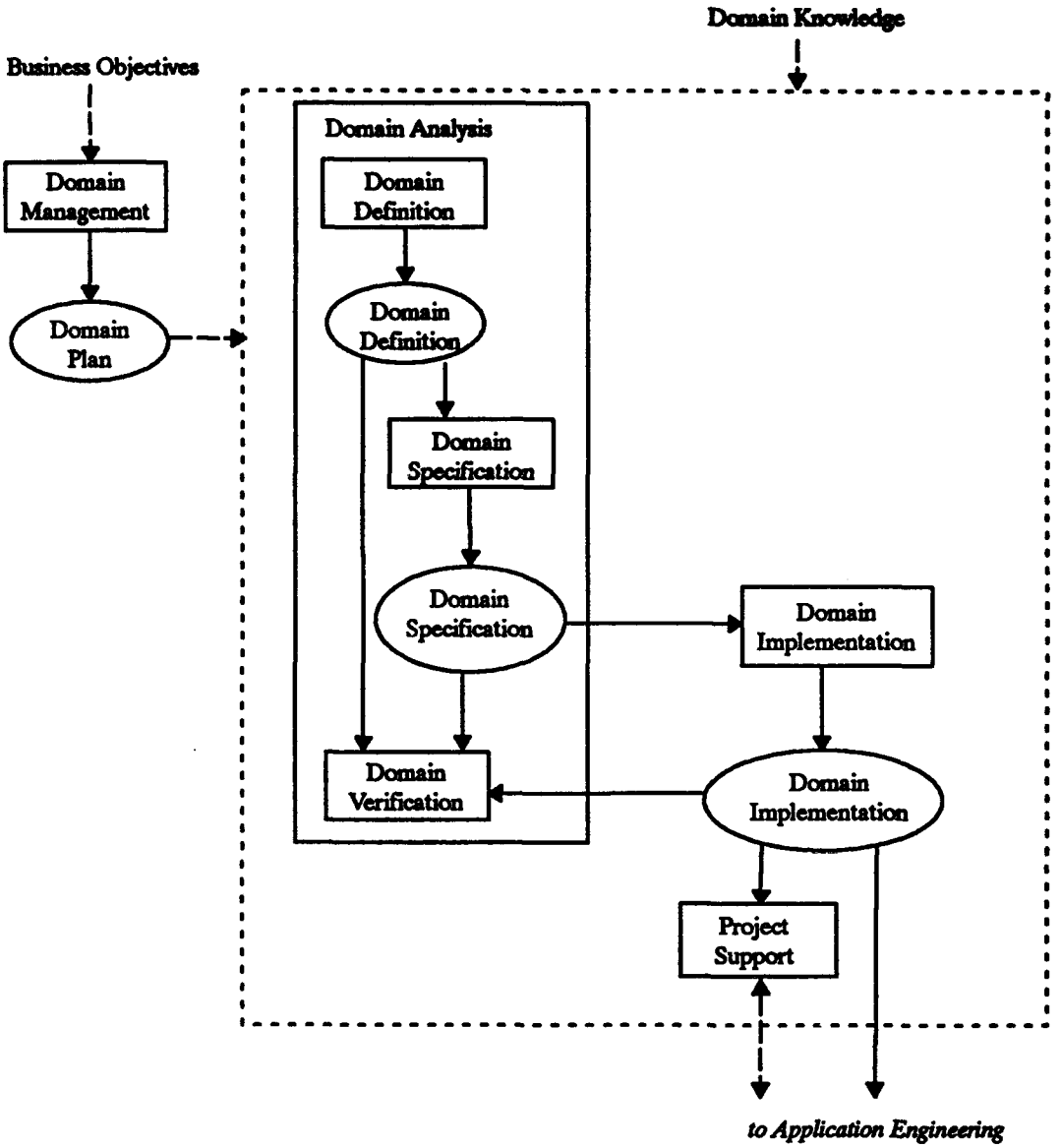


Figure DE-1. Domain Engineering

3.1 PROCEDURE

Follow these steps for the Domain Engineering Activity.

Step: Domain Management Activity

Action	Plan, monitor, and control the use of domain resources to provide standardized process and product family for a domain of interest to achieve organizational business objectives.
Result	Domain Plan
Heuristics	Define long-range and near-term objectives for a business area. Organize and manage domain resources to achieve those objectives.

Step: Domain Analysis Activity

Action Scope and specify a domain based on an analysis of business objectives of an organization.

Input Domain Plan

Result

- Domain Definition
- Domain Specification

Heuristics

- Create an informal definition of the domain. Characterize its scope, the aspects common to all systems in the domain, and the features that vary across systems in the domain. Explicitly state what is not part of the domain. Provide a glossary of common terms. Assess the viability of supporting each of the aspects you have characterized.
- Precisely specify problems within the scope of the domain. Describe both common problems and variations in those problems. Specify solutions to the problems in the domain so that the solutions vary in the same way as the problems. Specify the standardized Application Engineering process for building applications in the domain.

Step: Domain Implementation Activity

Action Implement the domain as defined by the Domain Specification.

Input

- Domain Definition
- Domain Specification
- Domain Plan

Result Domain Implementation

Heuristics

- Implement the product family and process described in the Domain Specification. Incorporate variations into the implementation of the solutions.
- Create the standards and procedures, in documentation and supporting automation, that institute a standardized Application Engineering process as specified in the Domain Specification.

Step: Project Support Activity

Action Support a project in performing the Application Engineering process.

Input Domain Implementation

Heuristics Deliver, install, and support the Domain Implementation for use by projects in the domain.

3.2 Risk Management

Risk — The products of Domain Engineering (standardization of Application Engineering products and process) will not lead to standardized domain practices on projects.

Implication — The Domain Engineering investment will not produce projected benefits for the business organization.

Mitigation

- Staff the Domain Engineering work with experienced project managers and engineers. Ensure that all work is actively reviewed by other experienced managers and engineers and is adequately reviewed by all participants of application engineering projects.
- Evaluate the effectiveness of the Domain Engineering process and work products relative to past project experiences. Ensure that the characteristics of that experience or the resulting systems are not in conflict with the process and work products.
- Provide unified management of domain engineering and application engineering projects. Establish an organizational commitment to the combined success of all participants.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

None

4.2 Feedback From Product Consumers

Contingency — The standardized product family is inadequate to support the needs of particular customers.

Source — Application Engineering

Response

- Determine that expressed needs are outside of or otherwise conflict with chosen domain objectives or cannot be viably satisfied given available domain resources.
- Evolve the domain to accommodate changing needs.

Contingency — The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source — Application Engineering

Response

- Determine that the benefits of process standardization outweigh the interests of the particular project.

- Evolve the definition of the Application Engineering process to reflect this project's experience or to be adaptable to the particular conditions of concern.

DE.1. DOMAIN MANAGEMENT ACTIVITY

1. GETTING STARTED

Domain Management is an activity of Domain Engineering for managing business-area resources to achieve assigned business objectives. The business-area organization provides resources and direction for both domain engineering and associated application engineering projects.

Domain Engineering develops and evolves a domain through a series of increments. The Domain Plan lays out both a master plan for evolution through projected increments (evolution plan) and, as each increment is initiated, a detailed plan for each increment (increment plan). The evolution plan determines the nature of the market addressed and how resources are allocated between domain engineering and application engineering projects. An increment plan determines how domain engineering resources are applied to create an efficient Application Engineering process and a high-quality product family.

Domain Management monitors domain engineering performance to assess progress, ensure proper adherence to plans, and guide needed revisions to the evolution and increment plans. A key concern of Domain Management is coordinating Domain Engineering activities to support the needs and priorities of targeted application engineering projects in satisfying customers' needs and in achieving the overall objectives of the business area. Domain Management assists the management of targeted application engineering projects to create plans that ensure optimal leverage from the domain and to identify enhancements needed by the projects for inclusion in timely increments of domain evolution.

1.1 Objectives

The objective of Domain Management is to manage business-area resources to achieve the organization's business objectives. Management establishes domain objectives for the organization to guide the creation and revision of an increment plan for incremental domain development and evolution. Domain evolution is concerned with the overall trends in the market for the business area and how resources should be applied to best serve the evolving market. The primary concern in planning for domain evolution is projecting the evolution of market need and organizational capability to meet that need over time.

Each increment of domain development should result in the ability to serve a particular level of market need. For each increment of development, Domain Management develops a plan to deliver capabilities that match the needs of targeted application engineering projects. Application engineering projects are planned in coordination with Domain Management and with an awareness of domain objectives and capabilities, to meet particular customer needs.

1.2 Required Information

The Domain Management Activity requires the following information:

- Business objectives, specifically the priorities of executive management for business area development
- Domain Definition: Domain Status

1.3 Required Knowledge and Experience

The Domain Management Activity requires domain and business-area knowledge and experience in:

- The characteristics of the market for the business area
- All aspects of strategic business development and business-area management in the organization, including how to create a long-range business plan
- All aspects of application project management in the organization

2. PRODUCT DESCRIPTION

Name Domain Plan

Purpose Define long-range and near-term objectives and organize and manage domain resources to achieve those objectives.

Content A Domain Plan consists of three parts:

- ***Domain Evolution Plan.*** The Domain Evolution Plan defines long-range objectives for the domain and organizes resources to achieve them. This plan is strategic in nature and recognizes that both an organization's capabilities and its opportunities for profitable business change over time. Not all objectives can be met initially but must develop in the course of time, in increments that balance alternative uses of available resources against the potential for return.
- ***Practices and Procedures.*** Practices and Procedures prescribe the preferred practices and procedures that are to guide the proper performance of domain development.
- ***Domain Increment Plans.*** A Domain Increment Plan specifies how to organize and manage Domain Engineering resources to achieve near-term domain objectives.

Both the Domain Evolution Plan and the Domain Increment Plans include the following:

- ***Risk Analysis.*** Identification of uncertainties in meeting allocated business (for the Domain Evolution Plan) or domain (for a Domain Increment

Plan) objectives, assessment of the risks of failure, and identification of mitigation strategies.

- *Objectives.* The scope and focus of support to be provided for the domain or the increment of domain development, reflecting the needs and priorities of targeted application engineering projects. Scope is indicated by an identification of previously built systems upon which the domain will be based; focus is indicated by the mix of near-term and long-term business objectives. Objectives are divided into risk objectives and product objectives. Risk objectives attempt to mitigate risks identified in the risk analysis. Product objectives establish goals and success criteria for creating specific work products.

- *Schedule.* The allocation of domain resources to development increments that satisfy domain objectives or to activities within an increment that satisfy increment objectives. The schedule establishes specific milestones and success criteria for domain development increments or for the activities of a development increment.

- *Issues.* A description of issues that arise in performing the plan.

Form and Structure

To the extent possible, the form of a Domain Evolution Plan should be the form your organization currently uses: the Domain Evolution Plan should follow the form used for long-range business planning; Practices and Procedures should follow the form used for standardizing the practices of application projects; and the Domain Increment Plans should follow the form used for application project planning.

Verification Criteria

The verification criteria for the Domain Evolution Plan are:

- The plan must show how near-term objectives contribute to long-range objectives. All near-term objectives must support long-range objectives.

- The projected market for products in the business area must be large enough to compensate sufficiently for projected costs and risks.

- For the Domain Evolution Plan:

  - The plan is complete (i.e., it addresses all business objectives).

  - The plan is credible (i.e., it sets forth a strategy that is feasible, given projected resources).

  - Domain objectives are realistic, given the projected availability of resources and strength of the competition.

  - All important risks are identified and addressed.

  - Criteria are defined to measure progress against objectives and to choose among alternative plans or indicate a need for replanning.

- Each Domain Increment Plan institutes a plan that seems likely to achieve the objectives assigned to that Domain Engineering increment by the Domain Evolution Plan.

3. PROCESS DESCRIPTION

The Domain Management Activity consists of three steps as shown in Figure DE.1-1.

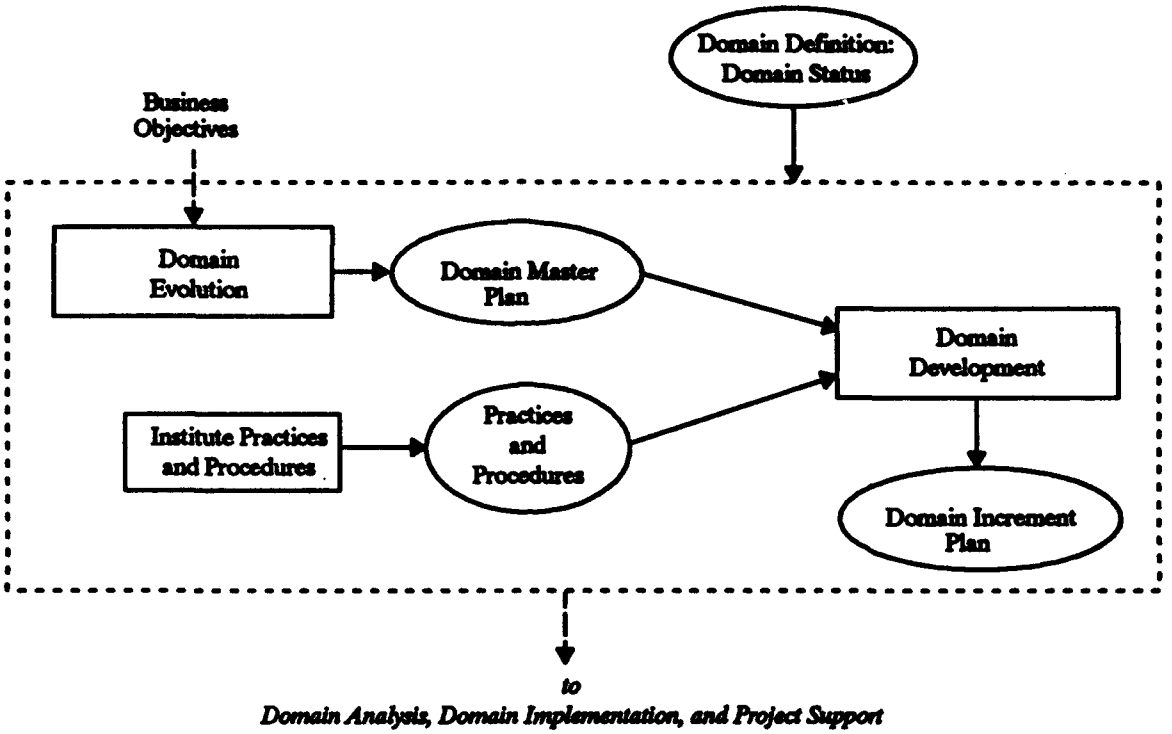


Figure DE.1-1. Domain Management Process

The Domain Evolution step begins upon creation of a domain and continues until the domain is no longer judged to be economically viable. The Domain Evolution Plan prescribes a series of Domain Development increments. Each increment is planned and performed iteratively until its objectives in the Domain Evolution Plan are met. The Domain Evolution Plan is subject to revision after the completion of each increment to reflect progress or changing needs of the market or targeted application engineering projects and their customers. The step to Institute Practices and Procedures occurs before the initiation of the first increment of Domain Development and is revisited as needed to update the Practices and Procedures to ensure an effective and efficient approach to Domain Engineering.

3.1 PROCEDURE

Follow these steps for the Domain Management Activity.

Step: Domain Evolution

Action	Create a plan for Domain Evolution.
Input	• Business objectives

- **Domain Definition: Domain Status**

Result

Domain Plan: Domain Evolution Plan

Heuristics

- Develop a set of domain objectives that will guide you in the long-term evolution of the domain. Develop a preliminary statement of domain objectives that expresses the strategic mission of the organization. (Refer to the heuristics for the Domain Development step for suggestions on a risk-based management process that you can also apply in performing this step.)

- Develop market projections as a basis for understanding current and future customer needs to be met. Refine the objectives to match perceived market opportunities. Consider the following questions:

  - What are the critical aspects of your market?

  - Who are your customers and what factors are most important to them in deciding to award contracts?

  - What type of products will you produce to address this market?

  - Who are your competitors? What are their strengths and weaknesses?

  - What are your strengths and weaknesses?

  - How many systems do you expect to produce both in the first year and as the domain matures?

- Describe a strategy for achieving domain objectives. The life cycle of a domain comprises four phases:

  - ***Conception.*** A small, cohesive group explores the boundaries of a domain to establish a viable market basis; project support is not viable yet.

  - ***Elaboration.*** One or a few very similar projects are directly supported; domain evolution emphasizes needs that are important to most customers.

  - ***Expansion.*** Supporting the planned diversity of projects is now viable; market opportunities drive further domain evolution.

  - ***Consolidation.*** Little additional diversification is viable; projects are managed to fit within the supported/variations in domain capabilities as much as possible.

- Develop a profile of the market to be supported as the domain matures. Provide details in terms of evolving domain objectives.

- Develop a profile of the evolution of automation to be developed in support of application engineering projects. Consider the degree of automation you expect to deploy in support of application engineering, both initially and as the domain matures.
- Develop a profile of the resources needed to fulfill domain objectives. Consider both quantitative and qualitative needs in all skill categories. Project how these resources are to be allocated between domain engineering and application engineering projects. Consider how resources will be allocated between domain development and application engineering projects.
- Define a series of domain development increments, and define objectives, consistent with domain objectives and allocate them to increments. Consider the following questions:
  - How do you expect the market, your business, and your competition to change over time?
  - What are the long-term risks in developing the domain, and how will you mitigate each one?
  - What are the objectives for the domain, and what are the objectives of each increment of development that is to achieve those objectives?

Step: Institute Practices and Procedures

Action Develop and document standard practices and procedures to be followed in the activities of Domain Engineering.

Input None

Result Domain Plan: Practices and Procedures

Heuristics

- Prescribed practices and procedures should encompass administrative, software development (e.g., requirements and design methods, coding and documentation standards), project management and control, and quality assurance (e.g., testing, walkthrough, and review procedures).
- Configuration management procedures are a key element for controlling iterative domain development. Each iteration of domain development is represented by one version of each domain engineering work product that you produce. Feedback on the use of a product version leads to the creation of a new version in a later iteration of development.
- Consider how consistency and quality standards will be achieved in domain practices.

Step: Domain Development

Action Create a plan for developing a domain increment.

Input

- **Domain Evolution Plan**
- **Practices and Procedures**
- **Domain Definition: Domain Status**

Result

Domain Increment Plan

Heuristics

- **A Domain Development increment consists of repeated cycles through a process comprised of four steps as shown in Figure DE.1-2. This guidance assumes you are experienced in project management. Refer to the descriptions of the process model and activities for the ESP (Software Productivity Consortium 1992b) for a detailed project management method that you can follow to tailor and elaborate this process.**

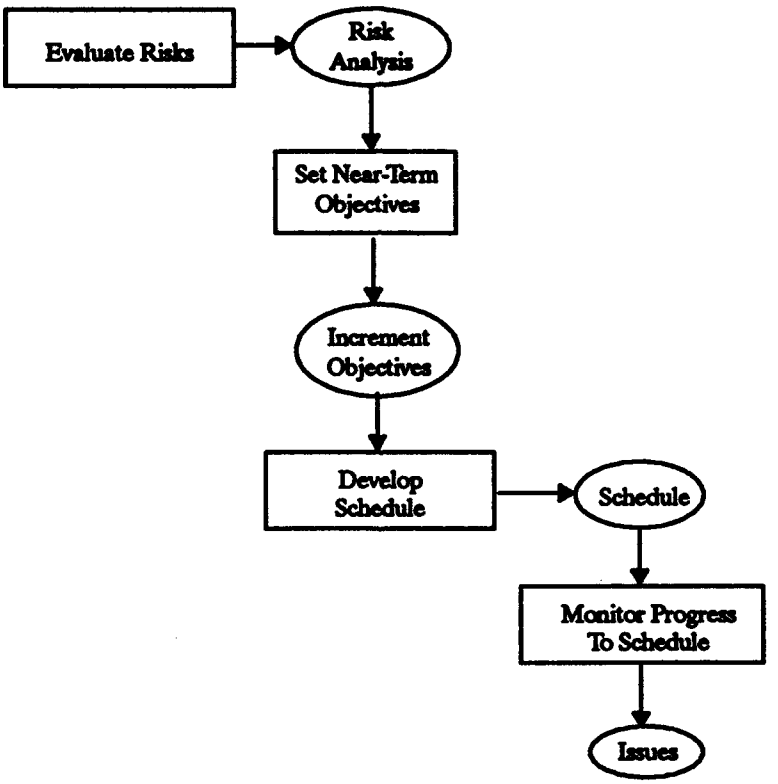


Figure DE.1-2. A Risk-Based Process for Increment Management

- **The Domain Evolution Plan identifies the objectives to be met by the increment. Identify and rank the domain development risks faced by the organization in meeting these objectives.**
  - **A recurring class of risks that domain development must face is the near-term ability of targeted application engineering projects to create required products. Another recurring class of risks involves how to evolve the domain to meet long-range domain objectives. Actions to mitigate these risk classes may conflict.**

  - Another important class of risks relates to problems discovered in previous iterations of the Domain Engineering process.

- Develop a prioritized set of near-term increment objectives that address the risks identified in the risk analysis. These objectives may be revised, as Domain Engineering iterates, to meet the Domain Evolution Plan objectives for the increment.
  - A domain-development approach must balance long-term domain objectives against the short-term needs of targeted projects.
  - Each objective should have associated success criteria that are used to determine whether the objective has been met. The success criteria should be written in such a way that they are directly measurable (if possible).
  - Objectives are often stated in terms of variations that characterize systems in the domain or variations to be allowed in the practice of Application Engineering.
- Develop a schedule that allocates resources to tasks.
  - Create specific goals and completion criteria for each task. Each task is characterized by a Domain Engineering activity to be performed and completion criteria appropriate to that activity. The full set of tasks must address the near-term objectives within the resource budget provided. If the resource budget does not allow all the objectives to be addressed, the objectives with the highest priority should be addressed.
  - Plan for short iterations so that mistakes made in front-end tasks may be caught and corrected quickly in a subsequent iteration. Iterations at the beginning of the life cycle of a domain should be particularly short (three to four months) to compensate for the likely learning curve in domain concepts.
- Monitor domain engineering work progress to planned milestones and completion criteria.
  - The schedule establishes milestones and completion criteria for activities that are used to evaluate progress. Whenever new issues are identified or progress differs from that planned, evaluate whether to document your concerns for future planning or to revise the current plan for immediate action.
  - Document the source, implications, and possible and actual resolutions of each issue.

3.2 Risk Management

Risk Domain plans will not be met within schedule with allocated resources.

Implication	Domain capabilities will fall short of plans.
Mitigation	• Review plans with experienced engineers to ensure that planned development is technically viable. • Reevaluate domain objectives and replan domain evolution to provide for shorter iterations that achieve essential capabilities sooner; defer work on less important objectives.
Risk	Market needs will not be met by projected development.
Implication	Demand for projects will not be sufficient to justify planned investment in the domain.
Mitigation	Review objectives and plans with marketing and major customers to ensure that market needs are properly represented.
Risk	Domain engineers resist using standardized practices and procedures.
Implication	Inefficient operation and employee dissatisfaction will reduce productivity.
Mitigation	• Involve domain engineers in developing practices and procedures. • Provide education and apprenticeships. • Conduct pilot projects that emphasize learning new skills over product delivery.
Risk	Domain engineers fail to recognize when to terminate an iteration.
Implication	• There will be excessive detail in products without adequate foundation or potential benefit. • Schedules will slip.
Mitigation	Review objectives and completion criteria to make sure they are specific and understood by the domain engineers.
Risk	Project needs will not be met by planned development.
Implication	Provided reusable assets will not have sufficient value to targeted projects to justify costs of the domain.
Mitigation	Review objectives and plans with project managers to ensure that the needs of their projects and the projects' customers are properly understood.
Risk	Domain plans will not be met within schedule with allocated resources.
Implication	Domain capabilities will fall short of plans.
Mitigation	• Review plans with experienced engineers to ensure that planned development is technically viable.

- Reevaluate objectives and project needs to focus support on key needs of the targeted projects first; defer work on less urgent objectives.
- Revise the Domain Increment Plan to add or defer activities, or to reallocate time and resources among planned activities, as priorities dictate.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 FEEDBACK TO INFORMATION SOURCES

Contingency The Domain Definition and/or Domain Specification fails to provide the needed capabilities required by the Domain Plan.

Source Domain Analysis Activity

Response Describe ways in which the Domain Definition and/or Domain Specification fail to provide the necessary capabilities. Modify schedule to allow completion of indicated Domain Definition or Domain Specification revisions.

4.2 FEEDBACK FROM PRODUCT CONSUMERS

Contingency Customer needs are not being met by the domain.

Source Project Support Activity

Response
- Revise the Domain Plan to accommodate new needs.
- Determine that needs are incompatible with your organization's business objectives.

Contingency Project needs are not being met by the domain.

Source Project Support Activity

Response
- Revise the Domain Plan to accommodate new needs.
- Determine that the needs of a project are incompatible with the organization's business objectives and therefore outside the proper boundaries of the domain.

Contingency Practices and procedures are either ineffective or inefficient.

Source
- Domain Analysis Activity
- Domain Implementation Activity
- Project Support Activity

Response Revise practices and procedures to reflect domain experience.

Contingency	The Domain Plan is too ambitious for available resources or expertise.
Source	• Domain Analysis Activity
	• Domain Implementation Activity
Response	• Allocate additional resources or time to domain development.
	• Refine the Domain Plan to reduce the scope.

This page intentionally left blank.

DE.2. DOMAIN ANALYSIS ACTIVITY

1. GETTING STARTED

Domain Analysis is an activity of Domain Engineering for studying and formalizing a business area as a domain. The purpose of formalizing a domain is to standardize and leverage knowledge of how recurring and varying customer requirements affect the form and content of a product. The scope of a domain is a business decision based on evaluations of available expertise and potential business opportunities. Domain Analysis specifies a standardized Application Engineering process and product family and verifies that a corresponding Domain Implementation meets that specification.

1.1 Objectives

The objectives of Domain Analysis are to:

- Determine scope and to evaluate the economic viability of a domain
- Establish, manage, and evolve a repository of domain knowledge
- Specify an Application Engineering process and product family appropriate to the domain

1.2 Required Information

Domain Analysis requires the following information:

- Business area knowledge
- Domain Plan: Domain Objectives
- Domain Implementation

1.3 Required Knowledge and Experience

Domain Analysis requires domain and software knowledge and experience in:

- The needs that motivate systems in the domain
- The environments in which these systems operate
- How these systems are built

2. PRODUCT DESCRIPTION

Domain Analysis creates two work products: Domain Definition and Domain Specification.

2.1 Domain Definition

Purpose — A Domain Definition (see Section DE.2.1) is an informal description of the systems in a business area that form a domain. A Domain Definition characterizes how existing systems, systems being developed in ongoing projects in the domain, and potential future systems are similar and how they differ.

Verification Criteria — The Domain Definition captures sufficient information to allow domain engineers to describe accurately any existing or potential system.

2.2 Domain Specification

Purpose — A Domain Specification (see Section DE.2.2) precisely characterizes a product family for the domain and an Application Engineering process for constructing members of that family.

Verification Criteria — The Domain Specification precisely expresses the domain as captured in the Domain Definition.

3. PROCESS DESCRIPTION

The Domain Analysis Activity consists of the three steps shown in Figure DE.2-1.

3.1 Procedure

Follow these steps for the Domain Analysis Activity.

Step: Domain Definition Activity

Action — Characterize the domain to satisfy domain objectives (see Section DE.2.1).

Input — Domain Plan

Result — Domain Definition

Heuristics

- Characterize the domain by defining its scope (i.e., classes of systems, characteristics, or functions included and excluded from the domain) and how included systems are distinguished from one another. These definitions are a basis for judging the qualitative and economic characteristics of the domain to determine whether the domain as defined will be economically viable.
- Consider how a product for a system is similar and distinguishable from a product for existing systems.

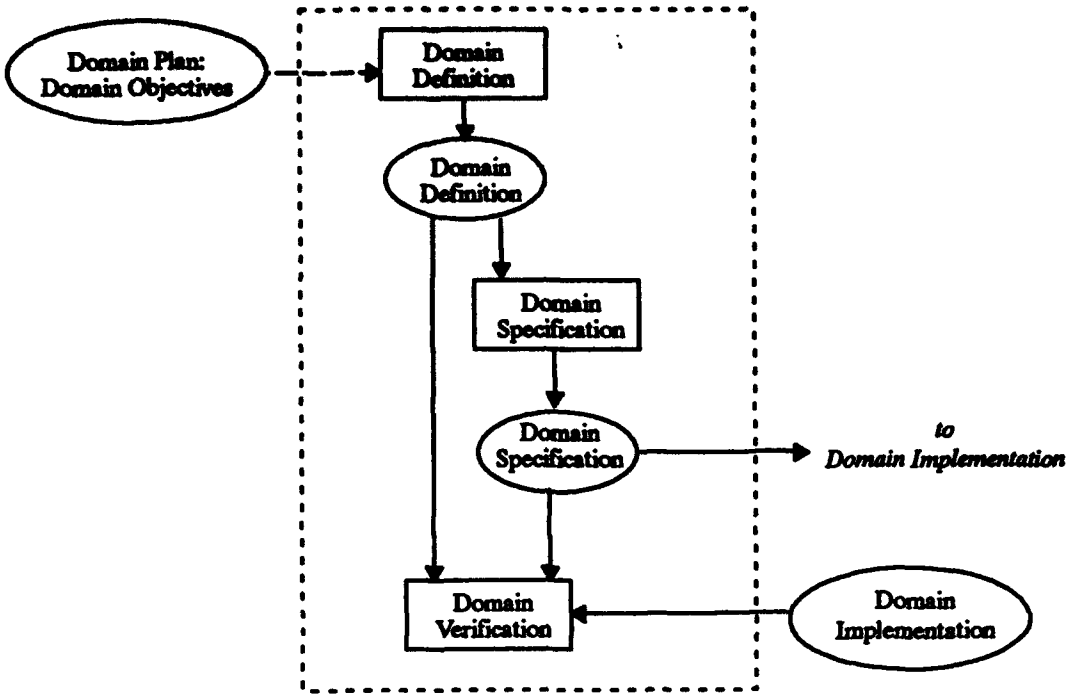


Figure DE.2-1. Domain Analysis Process

- Use this definition as a basis for judging the qualitative and economic characteristics of the domain to determine whether the domain, as defined, will be economically viable. If analysis of the Domain Definition fails a test of economic viability, reevaluate the scope of the domain in terms of domain objectives.

Step: Domain Specification Activity

Action Specify Application Engineering Process Support (see Section DE.2.2).

Input Domain Definition

Result Domain Specification

Heuristics

- Create a standard Application Engineering process for the domain. Ensure that all needs of projects are flexibly supported.
- Design an Application Modeling Notation for communication of system requirements and constraints among customers and application engineers. Identify the decisions that an application engineer must make to describe fully the variations in a system. This notation must accommodate aspects appropriate for the product family (such as functional [e.g., behavioral] and nonfunctional aspects [e.g., size, timing, fault tolerance, hardware architecture, hardware/software configuration]) so that the application engineer can adequately express customer requirements. This notation should be based on existing (formal or informal) notations used by domain experts.

- Ensure that the Application Modeling Notation is precise enough to be used as a source for mapping into exact system solutions. Create standardized requirements for the domain. This description must establish both the common and variable aspects of the behavior and constraints of product family members. An unambiguous specification of requirements is needed so that domain implementors can determine what impact a decision has on a system. This also provides a basis for explaining the notation to application engineers.

- Create a standardized design for the product family. The design must satisfy both the common and variable aspects of the product family. A standardized design includes both design structures that define various views of the product structure and components from which a product is constructed to satisfy customer requirements.

Step: Domain Verification Activity

Action Verify the correctness, consistency, and completeness of domain engineering work products (see Section DE.2.3 for motivation).

Input

- Domain Definition
- Domain Specification
- Domain Implementation

Result None

Heuristics

- Verify the consistency and completeness of the Domain Definition.
- Verify that the representation of the Application Engineering process in the Domain Specification is consistent and complete with respect to its representation in the Domain Definition.
- Verify that the representation of the application engineering product in the Domain Specification is consistent and complete with respect to its representation in the Domain Definition.
- Verify that the Product Implementation is consistent and complete with respect to the Domain Specification.
- Verify that the representation of the Application Engineering process in the Process Support is consistent and complete with respect to its representation in the Domain Specification.

3.2 Risk Management

Risk The cost of an increment of Domain Analysis is projected to exceed the budget.

Implication Insufficient resources exist to complete a planned iteration of Domain Engineering.

Mitigation

- Reduce the current scope.
- Seek a change in domain objectives or an increase in the budget for the increment from Domain Management.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Definition and a Domain Specification that satisfy the Domain Plan as closely as possible.

Contingency The Domain Implementation does not satisfy the Domain Specification.

Source Domain Implementation Activity

Response Clarify the intent of the Domain Specification.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Implementation Activity

Response

- Revise the Domain Specification.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Domain Definition and/or Domain Specification fails to provide the capabilities required by the Domain Plan.

Source	Domain Management Activity
Response	Evolve the Domain Definition and the Domain Specification to be consistent with the Domain Plan.
Contingency	The Domain Specification is incomplete, ambiguous, or inconsistent.
Source	Domain Implementation Activity
Response	Revise the Domain Specification to correct the inadequacies.
Contingency	The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.
Source	Project Support Activity
Response	• Determine that the benefits of process standardization outweigh the interests of the particular project. • Evolve the definition of the Application Engineering process to reflect the project's experience or to be adapted to the particular conditions of concern.

DE.2.1. DOMAIN DEFINITION ACTIVITY

1. GETTING STARTED

Domain Definition is an activity of Domain Analysis for creating a Domain Definition. A Domain Definition is an informal description of the systems in the business area that form a domain. A Domain Definition characterizes how systems in the domain are similar and how they differ.

1.1 OBJECTIVES

The objectives of the Domain Definition Activity are to:

- Establish a conceptual basis and bounds for more detailed Domain Analysis
- Determine whether planned development and evolution of the domain is viable relative to the organization's business objectives
- Establish criteria by which management and engineers can judge whether a proposed system is properly within the domain

1.2 REQUIRED INFORMATION

The Domain Definition Activity requires the Domain Plan.

1.3 REQUIRED KNOWLEDGE AND EXPERIENCE

The Domain Definition should be developed by people with a variety of backgrounds in the business area understudy, including engineering, business, and management experience. Specific expertise is needed for:

- Broad business-area knowledge: What systems have been built; the nature of systems likely to be requested and built; new features and technology that will impact on customer expectations.
- Broad market awareness: What contracts are forthcoming; what kind of competition there will be for those contracts; the long-term growth potential of this business area.
- Proposal development experience: What aspects of a system are critical to winning a contract proposal in this business area.
- System development and management experience: Size, cost, and schedule estimation for systems in this business area; an intuitive understanding of the technical difficulty and

feasibility of a given feature; knowledge of the common areas of confusion or vagueness in requirements or high-level design; knowledge of the common variations or changes to systems in this business area over the entire life cycle, including development, installation, and maintenance.

2. PRODUCT DESCRIPTION

Name

Domain Definition

Purpose

A Domain Definition establishes the scope of a domain and a justification of its economic viability. It provides a basis for determining, informally, whether a system is properly within that scope.

The Domain Definition does not answer detailed questions of scope, but clearly includes and excludes broad classes of systems. Assumptions of commonality and exclusion identify the common features of systems in the domain, thereby establishing a family. Assumptions of variability identify how systems in the family are distinguished from one another. Justification provides a basis for judging technical and economic feasibility and market potential of the domain to evaluate whether there is sufficient confidence in the viability of developing the business area as a domain.

Content

A Domain Definition consists of the following components:

- ***Domain Synopsis.*** An informal statement of the scope of the domain.
- ***Domain Glossary.*** Definitions of significant terminology used by experts in discussing needs and solutions in the domain.
- ***Domain Assumptions.*** A description of what is common, variable, and excluded among systems in the domain.
- ***Domain Status.*** An assessment of the current maturity and viability of the domain relative to its planned evolution.
- ***Legacy Products.*** A representative collection of work products from existing systems in the product line which may be a suitable source of information and raw material for developing the domain.

Verification Criteria

- Every Domain Assumption must be endorsed for every customer type in the customer base identified in a Domain Status.
- The needs of all customer types in the customer base must be fully met by the Domain Synopsis and Domain Assumptions.

2.1 Domain Synopsis

Purpose

The Domain Synopsis is an informal statement of the scope of the domain. It characterizes systems included in the domain.

Content

A Domain Synopsis includes an informal characterization of the systems that make up the domain.

Form and Structure

A Domain Synopsis is a simple narrative using terms defined in the Domain Glossary. Example DE.2.1-1 illustrates a fragment of a Domain Synopsis for the TLC domain. This fragment depicts typical information contained in a Domain Synopsis and the use of terms from the Domain Glossary.

> The Traffic Light Control Software System (TLC) domain is a family of embedded computer systems to control the operation of traffic lights at a given intersection. The TLC domain is limited to controlling traffic at intersections of two roads with at most one road dead-ending at the intersection. Systems in the TLC domain control traffic at the intersection by changing the indicators of each traffic light (called a traffic light sequence). Each traffic light sequence is coordinated with the other traffic lights in the intersection to prevent accidents while vehicles traverse the intersection. Usually, a TLC system generates its traffic light sequence based on a clock-generated cycle which may last from one (1) to three hundred (300) seconds. The traffic light sequence generated from the clock may be modified based on signals from optional input devices.
>
> One optional input device is a trip mechanism buried under the roadway. A trip mechanism may be associated with either a left-turn lane, a right-turn lane or a thru-traffic lane. Input from a trip mechanism associated with a left-turn lane controls whether the left-turn indicator of the traffic light is turned on during a traffic light sequence. A trip mechanism associated with a right-turn signal may act in an analogous manner for a right-turn signal. Inputs from any trip mechanisms may alter the traffic light sequence. A trip mechanism is not required for the proper operation of the turn lane.
>
> Another optional input device is the pedestrian crosswalk push button. This input controls whether the walk/don't walk indicator is on during a traffic light sequence. It may also modify the traffic light sequence. A push button is not required for the proper operation of the pedestrian lane.
>
> ....

Example DE.2.1-1. Fragment of TLC Domain Synopsis

Verification Criteria

- The Domain Synopsis must give an intuitive feel for the definitive characteristics of systems in the domain. It should, in itself, adequately describe any existing or potential system.
- A term that could have different meanings to different readers may be used in the Domain Synopsis only if it is defined in the Domain Glossary.

2.2 Domain Glossary

Purpose

The Domain Glossary is a compendium of precise definitions for all significant terminology used by experts for discussing problems (customer needs) and solutions (systems) in a domain. This domain terminology is organized into a taxonomy of terms.

Content

A Domain Glossary has two parts:

- A set of standard terms and their definitions
- A list of references to external sources which define and elaborate on relevant topics and terminology

Form and Structure

A reference to an external source is written using an accepted documentation style for a reference (e.g., author-date). Standard terms are defined in alphabetical order using the following forms:

Term 1 definition (source)

Term 2 (1) first definition (source); (2) second definition (source)

The source of the term's definition (source) is listed after the definition. Example DE.2.1-2 illustrates a fragment of a Domain Glossary for the TLC domain. This fragment depicts typical terminology needed to discuss systems, problems, and solutions in the TLC domain.

Term	Definition
Crosswalk	A specially paved or marked path for pedestrians crossing a street or road.[1]
Crosswalk Push Button	A monitoring device which allows a pedestrian to signal to the system his presence at a crosswalk.
Trip Mechanism	A traffic monitoring device used to determine whether a vehicle is present in a lane.
Traffic Control	A synchronized set of traffic light sequences specified as a function of time and traffic monitoring device inputs.
Traffic Light	A set of indicators placed at the intersection of streets to regulate traffic.
Traffic Light Cycle	One iteration through a traffic light sequence, i.e., from the display of the red indicator through to the next display of the red indicator.
Traffic Light Sequence	The order in which a set of traffic light indicators are displayed during a traffic cycle. Typically, this ordering is red, green, amber, but other orderings are possible.
Traffic Monitoring Device	A device that monitors the flow of traffic.
....	....

[1] Webster's Third New International Dictionary of the English Language Unabridged.

Example DE.2.1-2. Fragment of TLC Domain Glossary

Verification Criteria

- The Domain Glossary must contain precise definitions of all significant terminology used by domain experts for discussing the requirements or engineering of systems in the domain.
- Any term used in a definition that could have different meanings to different readers must also be defined.

- All independently-used terms that are generalizations, specializations, or components of defined terms must also be explicitly defined.
- Terms defined in the Domain Glossary must be sufficient for a domain expert to give an accurate description of any existing or potential system.

2.3 Domain Assumptions

Purpose

Domain Assumptions describe what is common to all systems in the domain and in what significant ways those systems vary and can be distinguished. These assumptions determine, informally, whether a system is within the scope of the domain.

Content

There are three types of assumptions:

- ***Commonality Assumptions.*** A set of assumptions about the characteristics that are common to all systems in the domain (commonalities).
- ***Variability Assumptions.*** A set of assumptions about the characteristics that distinguish systems in the domain (variabilities).
- ***Exclusionary Assumptions.*** A set of assumptions about the characteristics of systems that are outside the scope of the domain (exclusions).

Every assumption is composed of a description and justification.

Assumptions may also be elaborated with associated, subordinate assumptions. For example, a commonality assumption may have specific variabilities associated with it. Similarly, a particular resolution of a variability assumption can be thought of as characterizing a subfamily of the product family. The subfamily then may have additional, more specific commonalities and variabilities that further distinguish the members of the subfamily.

Form and Structure

An assumption description and justification are informal text. Assumptions which elaborate another assumption should be presented in adjacent, indented text. Examples DE.2.1-3 and DE.2.1-4 illustrate fragments of some commonality and variability assumptions for the TLC domain. The justification provides rationale on why the domain engineers believe the assumption to be valid.

Verification Criteria

- Commonality and variability assumptions must capture all important aspects that are common to all systems in the domain and the significant ways in which these systems can vary. Exclusionary assumptions must not exclude needed capabilities.
- Systems must only vary as implied by the variability assumptions.
- Every commonality assumption must apply equally well, without qualification, to any system in the domain. Systems must not violate a stated commonality, either by excluding an included feature in the commonality assumptions or by including an excluded feature in the exclusionary assumptions.

....

- A TLC system controls the traffic light sequences at an intersection.

  Justification The purpose of a TLC system is to control when traffic light sequences change, and the interactions of the traffic lights at an intersection.

....

- A TLC system coordinates all traffic lights at an intersection.

  Justification Safe transit of intersection requires that streams of traffic not cross, e.g., east bound traffic and north bound traffic cannot have green indicators concurrently.

....

- The entire week is divided into traffic cycles. The traffic lights at the intersection are synchronized based on these traffic cycles.

  Justification Traffic patterns and loads vary over the course of a day and of a week. The timing of the traffic cycles must be varied to deal with the variations in the traffic load.

....

- A TLC system must process signals from a trip mechanism or push button within a specified time.

  Justification Smooth traffic flow depends upon the TLC system detecting and responding to requests to a traffic light sequence in a timely manner.

....

Example DE.2.1-3. Fragment of TLC Commonality Assumptions

- All reviewers must agree that domain experts will consider Domain Assumptions to be consistent and unambiguous, relative to the definitions in the Domain Glossary. A term that could have different meanings to different readers may be used in a Domain Assumption only if it is defined in the Domain Glossary.

2.4 Domain Status

Purpose

Domain Status describes the current technical maturity of the domain that the organization has achieved relative to planned evolution, and assesses the viability of evolution. Of particular concern are unsupported variability assumptions (i.e., default commonalities).

The Domain Status describes evaluations that establish whether the domain, as defined, will be economically and technologically viable. Qualitative and quantitative criteria assess whether current development and future evolution of the domain will support the organization's business objectives.

- An intersection may be formed from two through roads (an X intersection), or from one through road and a dead-ending road (a T intersection).

  Justification The number of roads that can meet at an intersection can vary from intersection to intersection. At least two roads must cross to have an intersection, but intersections of three or more roads are common. The marketing department believes that the vast majority of intersections are of the T and X variety. It has decided to concentrate on this large subset of traffic intersections.

....

- The number of lanes of traffic on any road approaching or leaving the intersection may vary. The minimum number of lanes of traffic is one (1). The maximum number is six (6).

  Justification Any road bringing traffic into the intersection must have at least one lane of traffic into the intersection. Any road leaving the intersection must have at least one lane of traffic from the intersection. Engineering has determined that the hardware components of the system cannot control more than six lanes of traffic in a timely fashion.

....

- Any road approaching the intersection may have a trip mechanism buried under the traffic lanes to alert the system to the presence of vehicular traffic. There may be no trip mechanism or there may be one (1) per traffic lane.

  Justification The traffic light sequence may take into account the presence or absence of vehicular traffic at the intersection.

....

- The duration of a traffic control schedule will vary.

  Justification Traffic patterns and volumes vary from intersection to intersection. Effective traffic control at these intersections requires schedules that take these variations into account. Traffic control schedules must also vary to account for differences in system configurations.

....

Example DE.2.1-4. Fragment of TLC Variability Assumptions

Content

The Domain Status is an informal characterization of the degree to which Domain Objectives are satisfied by past development. It includes the effects of further planned development.

At a minimum, the Domain Status must include:

- An endorsement that the Domain Synopsis and Domain Assumptions define economically viable domain.
- A concise statement that characterizes the confidence (or risk) associated with this endorsement. If possible, this should include a justification of the endorsement and a list of major unresolved issues or risks that may jeopardize the domain's viability.

Depending on the size of the potential business area and the attendant commitment to the domain, however, you may need a more detailed Domain Status consisting of some or all of the following:

- The target customer base. This is a profile of each type of customer and/or contract that comprises this area of business. Each profile on the list consists of:
  - The name of the customer (or contract) type.
  - A short description.
  - A list of attributes that characterize this type of customer. These attributes fall into two categories: technical expectations and administrative expectations. Technical expectations are features of the delivered products or the process for developing, testing, maintaining, or delivering and installing the products. Administrative expectations are gross cost, profit, and schedule aspects of a contract.
  - A list of specific (potential or current) contracts and/or customers that fall under this type.
- A grouping of the different alternatives of variability assumptions into priority subsets. Minimally, there are two subsets: "must have" and "nice to have." A system of numerical weighting may be desirable.
- A statement of the potential value of the domain. You may need to analyze alternative scopes of the domain separately. The following factors express the potential value of a particular scope:
  - The size of the potential market (consisting of the target customer base)
  - The planned share of that market
  - The projected income from that share
  - The expected cost of supporting that market
  - An expected profit margin to justify the risk of entering the market
  - A pricing structure for systems to be produced that supports the cost/profit expectations

Form and Structure

The maturity of a domain can be expressed as limitations in satisfying variability assumptions. Characterize the technical expectations of the target customer base as the subset of possible alternatives of variability assumptions that are supported. Risks can be mitigated by imposing limits on variability assumptions.

Verification Criteria

- Every alternative of every domain variability assumption must have an explicit (or overall group) economic justification from the potential value analysis and/or an endorsement from the customer base.
- The price/capability of systems offered (as supported by the domain value analysis for a particular domain alternative) must compare sufficiently well against the perceived capabilities of the competition in this business area to justify the anticipated profit and market share.

2.5 Legacy Products

Purpose

Legacy Products provide access to work products from existing systems that may be useful sources of information and raw material for developing the domain.

Content

Legacy Products consists of a representative collection of work products (or portions thereof) from existing systems in the product line to be supported by the domain.

Form and Structure

- Work products may be physically stored, on paper or in electronic media, or may only be identified by reference when sufficiently accessible in this way (e.g., in an organization's local library or in an accessible repository set up for another, existing domain).
- Work products are kept in Legacy Products in the form in which they were produced. Other, consuming activities of Domain Engineering will copy and excerpt or adapt these work products, as needed, in order to create reusable assets.
- The work products comprising the Legacy Products are organized in a suitable manner to provide access by other Domain Engineering activities to a particular system's work products or to individual work products of a particular type.

Verification Criteria

Each work product in Legacy Products must come from an existing system that was determined to be in the domain.

3. PROCESS DESCRIPTION

The Domain Definition Activity consists of the five steps shown in Figure DE.2.1-1.

3.1 Procedure

Follow these steps for the Domain Definition Activity.

Step: Define the Domain Informally

Action

Create a description of the domain, characterizing key technical objectives of included systems.

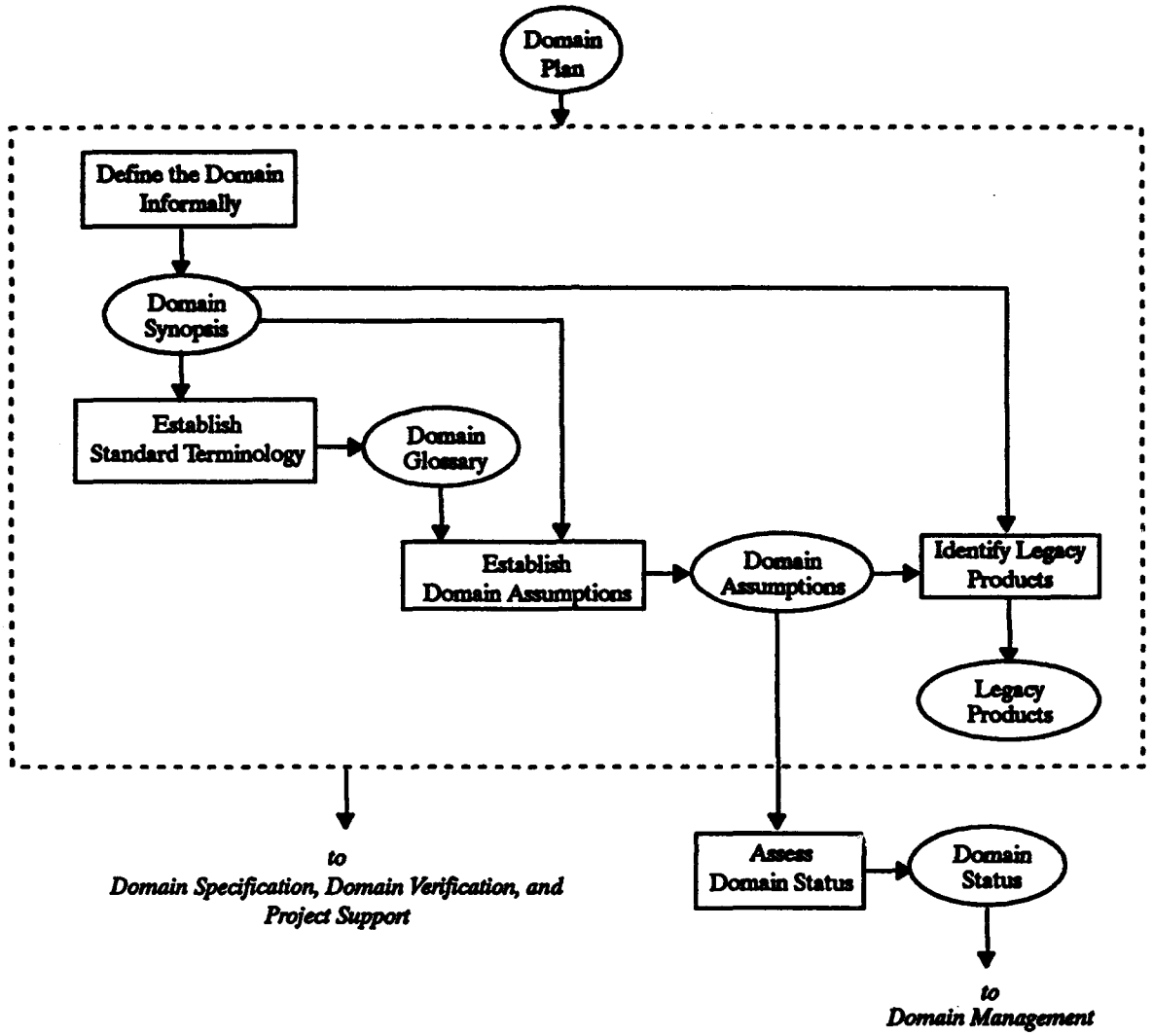


Figure DE.2.1-1. Domain Definition Process

Input Domain Plan: Domain Objectives

Result Domain Definition: Domain Synopsis

Heuristics

- Start with a one-sentence description of the family of systems that constitutes the domain.
- Refine the Domain Synopsis to two pages, at most, of intuitive and not overly-restrictive text. Impart, concisely, an informal but complete sense of the domain in the first paragraph. Try to focus on the essential nature, scope, and variety of systems in the domain.
- Characterize the type of problem that systems in the domain solve, and the external environment (i.e., devices, systems, and users) with which systems interact. Describe the observable behavior that systems exhibit in solving the problem. You might also establish significant constraints concerning

how the systems operate in terms of performance, reliability, or distribution concerns.

- Cover the primary functions performed by every system in the domain and any important functions performed by only some systems. Maintain a black-box perspective when describing functional aspects of the system.
- Use terms defined in the Domain Glossary to keep the Domain Synopsis short.
- If the domain (e.g., process control systems) is based on formal theories that provide experts with a common language of communication about problems, refer to those theories in the Domain Synopsis.

Step: Establish Standard Terminology

Action Create definitions of all significant terms used by domain experts in discussing the requirements or engineering of systems in the domain.

Input Domain Definition: Domain Synopsis

Result Domain Definition: Domain Glossary

Heuristics

- Maintain term definitions in alphabetical order for ease of reference. Provide cross-references to related terms.
- Use definitions from standard glossaries where possible. Make note of such sources in each definition for future traceability.
- Make definitions as precise as possible.
- Make sure that all terminology used in the Domain Synopsis is defined in the Glossary.
- Create a structure that shows term specializations and relationships among similar concepts. This action will reveal missing terms that represent generalizations or specializations of known terms.
- Create a structure that shows the composition of terms and the interrelationship of independent concepts in the formation of logical structures. This will reveal missing terms that are necessary to complete the definition of other terms or terms that tie other terms together into more complex concepts.

Step: Establish Domain Assumptions

Action Create lists of the assumptions that allow you to think of the envisioned set of systems as a family and the assumptions that allow you then to distinguish among them.

Input

- Domain Definition: Domain Synopsis
- Domain Definition: Domain Glossary

Result

Domain Definition: Domain Assumptions

Heuristics

- State only those assumptions that affect the system software and associated delivered products (e.g., documentation, test support).
- To create a preliminary set of assumptions:
  - Create a commonality assumption for each characteristic specified in the Domain Synopsis that is shared by all systems in the domain.
  - Create a variability assumption for each characteristic specified in the Domain Synopsis that is not shared by all systems in the domain.
  - For each term in the Domain Glossary, determine whether the term indicates a commonality or a variability among systems in the domain. Create an assumption accordingly.
- Make variability assumptions precise by indicating the type of decision the application engineer must make to resolve the variability. It is not sufficient to note only that some characteristic varies. You must establish how much flexibility the application engineer needs to characterize different systems adequately.
- Elaborate commonality assumptions to uncover specific variabilities assumptions associated with them. This will more precisely characterize a subset of the product family.
- Elaborate variability assumptions to find more specific commonality and subsequent variability assumptions that further distinguish members of the subfamily.
- Compare the characteristics of existing systems to facilitate the identification of common features and variations.
- Consider characteristics anticipated for future systems to identify additional variability assumptions.
- Look at the maintenance history of existing systems for an indication of how systems in the domain change over their life cycles. The histories of these systems indicate likely variabilities that characterize the evolution of individual systems.
- Use information on technological advancements that could impact the development of future systems in the domain to identify potential variations.

- Distinguish between system-generation-time and run-time variations when developing assumptions about variable aspects of the domain. Treat a run-time variation that is characteristic of all systems in the domain as a commonality.

- Use exclusionary assumptions to clarify a domain's boundary. Do not enumerate every type of system or function that is outside the domain. Rather, exclude explicitly those functions or characteristics that a domain expert might incorrectly assume to be part of a system when reading the Domain Synopsis. Thus, you can answer the question of whether a particular system belongs within the domain more directly by checking the exclusions. Exclusions often result from a viability analysis of the domain.

- Uncertainties that arise from analysis of customer requirements are often a good source of needed variability. These uncertainties are questions that customers, in the end, must resolve, but must be asked or be given additional information to be able to do so.

Step: Assess Domain Status

Action Evaluate the technical maturity of the domain in terms of Domain Objectives and plans for domain development and evolution.

Input

- Domain Plan
- Domain Definition: Domain Assumptions

Result Domain Definition: Domain Status

Heuristics Domain Status must result in an endorsement of and commitment to a specific domain scope (set of assumptions). This endorsement may come directly from the intuitions of experienced personnel, or it may derive from a more extensive, quantitative analysis. Even in the latter case, however, all estimates and interpretations will rely on the best judgment of senior people. Many of the following heuristics are couched in qualitative terms to capture the essence of the decision being made, but you can always augment that decision by the suggested, optional quantitative analyses.

Determine Marketability

- Are systems of this sort marketable? Look at the Domain Synopsis. The Domain Synopsis imparts an intuitive feel that it encompasses systems that the profiled customers will buy. If not, identify what is missing and add the missing elements to the Domain Synopsis.

- Do the commonality assumptions correspond to the essential needs of the target customers? Are they really interested in systems that do everything implied by these assumptions, or may the scope be reduced? Conversely, are features they always require (possibly in variant forms) missing? Change the assumptions accordingly.

- Do the variability assumptions represent the real issues that determine whether a system addresses the individual needs of the target customers? Are there inconsequential variations (to the target customers) that you can constrain without losing business? Are there important differences in the needs of target customers that are not captured in the variabilities? Change the assumptions accordingly.

- Do expectations of target customers include not only those of the system end-user, but also the expectations of analysts and decision makers who influence awarding of contracts? In particular, the administrative expectations should include items such as allowable contract costs.

Determine Implementability and Risk

- A straightforward way to reduce the number of variations supported is first to decide what customers (or customer types) must be retained for viability of the business. Then determine must-have and nice-to-have variability alternatives for each of these customers. The aggregation of the must-have alternatives determines the minimal scope of your domain. Now you can consider all other proposed variability alternatives with regard to their incremental value added.

- Determine whether your organization has a good understanding of the problems such systems are intended to solve, compared with your competition, and whether your organization has authoritative expertise that will be committed to developing this domain.

- Determine whether your organization can build such systems. Consider whether it has built such systems in the past. Consider the success of such prior experience with particular regard to demonstrated mastery of the relevant software technology.

- If particularly difficult technical issues arise, determine which Domain Assumptions are affected. If you can reduce a high technical risk at the cost of removing or constraining an assumption, consider how much domain value is lost by this reduction in scope and risk.

- Calculate a range for estimated system cost, potential market, anticipated income, and the like, representing your best- and worse-case expectations, since good estimates are typically hard to come by and are speculative in nature.

Step: Identify Legacy Products

Action — Identify existing systems in the product line that are considered representative of the domain and whose work products may prove useful as sources of information and raw materials in developing the domain.

Input

- Domain Synopsis

- Domain Assumptions

Result Domain Definition: Legacy Products

Heuristics

- Use the Domain Synopsis as a guide to select existing systems that are within the domain (or subsystems that would be parts of such systems).
- Based on Domain Assumptions, identify work products (or fragments, if appropriate) from these systems that reasonably satisfy some or all of the Domain Assumptions.
- Create a brief description of the selected systems and work products as a guide to their use as a source of information and raw materials by other Domain Engineering activities.

3.2 Risk Management

Risk There is a lack of critical expertise.

Implication The Domain Definition cannot be completed or there is unacceptably low confidence in the results.

Mitigation

- Commit time and resources to acquiring the expertise.
- Restrict Domain Assumptions sufficiently to reduce the need for expertise.

Risk The scope of the domain may be too narrow, precluding useful variations.

Implication

- Opportunities for additional projects are lost.
- Application engineering projects miss opportunities for reuse.

Mitigation Review the Domain Definition with management, experienced engineers, and potential customers to identify additional variations.

Risk The scope of the domain may be too broad.

Implication Resources are misapplied to solve an unnecessarily general problem.

Mitigation Review the Domain Definition with management and experienced engineers to identify under-constrained Commonalities.

Risk Domain Assumptions are too precise or too vague.

Implication Flexibility is reduced unnecessarily, or key decisions are left to the discretion of domain engineers.

Mitigation Review the Domain Definition with management and experienced engineers to identify over- or under-constrained Domain Assumptions.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Definition that satisfies Domain Objectives as closely as possible.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency	The Domain Definition fails to provide the capabilities required by the Domain Plan.
Source	Domain Management Activity
Response	Evolve the Domain Definition to be consistent with the Domain Plan.
Contingency	The Domain Definition is incomplete, ambiguous, inconsistent, or incorrect.
Source	• Domain Specification Activity • Domain Verification Activity
Response	Revise the Domain Definition to correct the inadequacies.

DE.2.2. DOMAIN SPECIFICATION ACTIVITY

1. GETTING STARTED

The Domain Specification Activity is an activity of Domain Analysis for creating a Domain Specification. A Domain Specification is a precise characterization of the product family denoted by a domain and of a process for constructing members of that family.

The product family is characterized from two perspectives: how problems are stated and how solutions are structured. Problems are expressed in the form of a requirements specification. Solutions are expressed in the form of a standardized design. Both forms are adaptable to anticipated variations in problems and solutions.

1.1 Objectives

The objectives of the Domain Specification Activity are to:

- Create a precise specification of the problems and solutions supported by the domain
- Define an Application Engineering process that is suited to the needs of building a product in the domain

1.2 Required Information

The Domain Specification Activity requires the Domain Definition.

1.3 Required Knowledge and Experience

The Domain Specification Activity requires domain and software knowledge and experience in:

- The process that projects in the organization use to construct an application engineering work project
- How systems in the domain are constructed, including the issues that application engineers must resolve to create a particular system
- The concepts and structures that are convenient forms by which to communicate about the distinguishing features of systems in the domain
- The principles and use of appropriate software product development methods

2. PRODUCT DESCRIPTION

Name — Domain Specification

Purpose — The Domain Specification is a specification for a product family and an associated Application Engineering process for producing members of the family.

Content — A Domain Specification consists of one of each of the following components:

- ***Decision Model.*** A Decision Model identifies the application engineering requirements and engineering decisions that determine how members of the product family can vary (see Section DE.2.2.1).
- ***Product Requirements.*** Product Requirements determine the behavior and operational characteristics of problems solved by the product family (see Section DE.2.2.2).
- ***Process Requirements.*** Process Requirements determine how Application Engineering is performed and which work products are produced as a result (see Section DE.2.2.3).
- ***Product Design.*** A Product Design determines the structure and composition of solutions provided by members of the product family (see Section DE.2.2.4).

Verification Criteria

- All aspects of the Domain Definition are accurately captured in the Domain Specification.
- Existing or envisioned systems can be described in terms of the Domain Specification. No systems exhibit behavior not indicated in the Domain Specification.

3. PROCESS DESCRIPTION

The Domain Specification Activity consists of the four steps shown in Figure DE.2.2-1.

3.1 PROCEDURE

Follow these steps for the Domain Specification Activity.

Step: Decision Model Activity

Action — Define the set of requirements and engineering decisions that an application engineer must resolve to select an instance from a designated product family.

Input — Domain Definition

Result — Decision Model

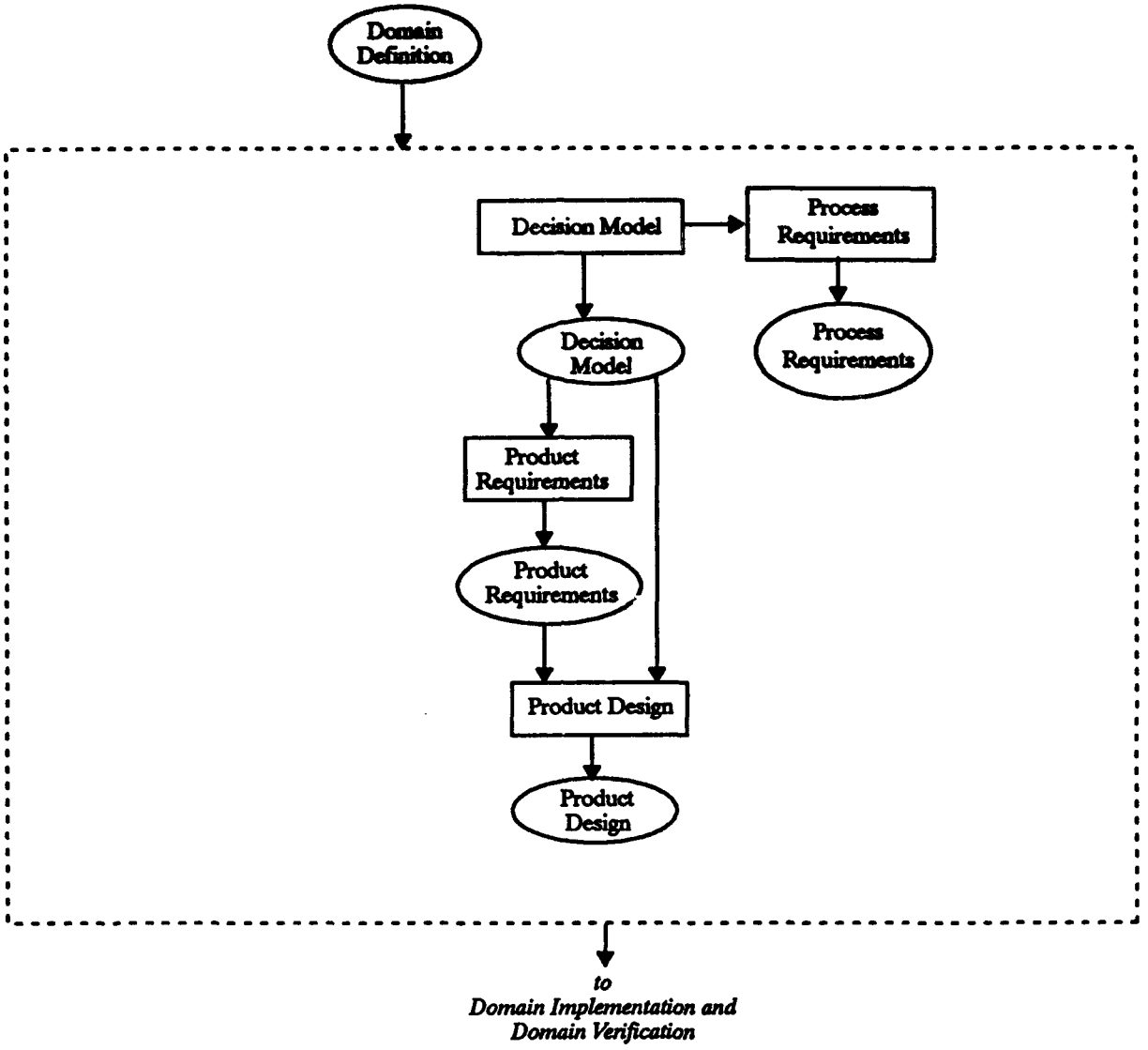


Figure DE.2.2-1. Domain Specification Process

Heuristics

- Define the decisions that lead to expected differences among the members of the family.
- A Decision Model for a product family should reflect the decisions that application engineers had to make when creating such products in previous projects.
- The Decision Model for a product family should reflect the variability assumptions from the Domain Definition.

- Ensure that supported decisions are sufficient to distinguish each existing product from other members of the family.
- Identify logical relationships among the decisions that characterize a product family and use them to structure the Decision Model. Such relationships can reduce the number and complexity of separate decisions that application engineers have to make.

Step: Product Requirements Activity

Action Specify the behavior and operational characteristics of problems solved by the product family.

Input
- Domain Definition
- Decision Model

Result Product Requirements

Heuristics
- Create a software requirements specification for the product family.
- To the degree that application engineering decisions change work product content, describe how content varies with respect to those decisions. This description will provide a partial basis for explaining the meaning of decisions to application engineers.

Step: Process Requirements Activity

Action Specify a standardized Application Engineering process.

Input
- Domain Definition
- Decision Model

Result Process Requirements

Heuristics
- Define the work products, activities, and process of Application Engineering.
- Develop the form and structure of the Decision Model as presented to the application engineer.

Step: Product Design Activity

Action Define the design (i.e., composition and structure) of the members of a designated product family.

Input
- Decision Model
- Product Requirements

- Domain Definition: Legacy Products

Result — Product Design

Heuristics

- Create a design for the product family, including a design for each required work product. An annotated outline is one model of a Product Design for a document work product. An information hiding structure and process structure from the ADARTS℠ (Software Productivity Consortium 1993) design method are models of a Product Design for a software work product.
- A key element of domain knowledge is how existing instances of the designated product family are designed. When feasible, derive the initial Product Design for a family by extracting the design essentials of existing instances. Ensure that the composition and structure of existing instances are appropriately reflected in the design.
- To the degree that Application Engineering decisions change work product composition and structure, describe how composition and structure vary with respect to those decisions. This description will provide a further, but still partial, basis for explaining the meaning of decisions to application engineers.

3.2 Risk Management

Risk — The Domain Specification does not accommodate a product family that meets the needs of the building products in the domain.

Implication — The domain will not provide sufficient opportunities for building unforseen products.

Mitigation — Compare previously developed systems that should be within a product family with expected needs of the domain. Check that likely differences are accommodated.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency — The Domain Definition is incomplete, ambiguous, or inconsistent.

Source — Domain Definition Activity

Response — Describe the inadequacies in the Domain Definition. Proceed with Domain Specification, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency — The Domain Plan cannot be satisfied with available technical capabilities.

Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Domain Specification that satisfies the Domain Plan as closely as possible.
Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency	Suggestions are made for Domain Specification changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.
Source	Domain Implementation Activity
Response	• Revise the Domain Specification. • Refer opportunities to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Domain Specification fails to provide the capabilities required by the Domain Plan.
Source	Domain Management Activity
Response	Evolve the Domain Specification to be consistent with the Domain Plan.
Contingency	The Domain Specification is incomplete, ambiguous, inconsistent, or incorrect.
Source	• Domain Implementation Activity • Domain Verification Activity
Response	Refine the Domain Specification to correct any inadequacies.
Contingency	The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.
Source	Project Support Activity

Response	Determine that the benefits of process standardization outweigh the interests of the particular project. Evolve the Application Engineering process to reflect this project's experience or to be more flexible under the particular conditions of concern.
Contingency	Supported product family (as represented by its constituent work product families) is not useful for a particular project.
Source	Project Support Activity
Response	• Determine that the nature of the problem and the consequent costs of upgrading the product family outweigh expected benefits to the particular project. • Evolve the domain engineering work product family to reflect this project's experience or to be more flexible under the particular conditions of concern.

This page intentionally left blank.

DE.2.2.1. DECISION MODEL ACTIVITY

1. GETTING STARTED

The Decision Model Activity is an activity of the Domain Specification Activity for producing a Decision Model. A Decision Model defines the set of requirements and engineering decisions that an application engineer must resolve to describe and construct a deliverable application engineering work product. A Decision Model is an elaboration of a domain's variability assumptions and is the abstract form (i.e., concepts and structures) of an Application Modeling Notation for a product family. These decisions, and the logical relationships among them, determine the variety of products in the domain. To construct a product, these decisions must be sufficient to distinguish the product from all other members of the family. The decisions establish how work products of application engineering, including software and documentation, can vary in form and content.

1.1 Objectives

The Decision Model Activity defines a set of decisions that are adequate to distinguish among the members of an application engineering product family and to guide adaptation of application engineering work products.

1.2 Required Information

The Decision Model Activity requires the Domain Definition.

1.3 Required Knowledge and Experience

The Decision Model Activity requires domain and software knowledge and experience in:

- Conceptual modeling skills similar to those needed to create a database conceptual schema; see, for example, Kent (1978) and Borgida (1985)
- The issues that experienced engineers resolve when constructing systems in the domain

2. PRODUCT DESCRIPTION

Name	Decision Model
Purpose	A Decision Model specifies the decisions that the Application Modeling Notation must allow an application engineer to make in describing a system

in the domain. These decisions determine the extent of variation in form and content that is possible in the work products that compose the products in the domain.

To interpret fully the effects of decisions (i.e., to understand all properties of the family member identified by a set of decisions) requires both a Decision Model and a Product Requirements. The Decision Model specifies only the variations among members of a family. It does not specify their common properties. Product Requirements state the common properties, plus the effects of the decisions in a Decision Model.

Content

The Decision Model work product consists of three components:

- ***Decision Specifications.*** Specifications of the set of decisions that suffice to distinguish among systems in the domain.
- ***Decision Groups.*** A structuring of the decision specifications into logical groups, based on domain-related criteria.
- ***Decision Constraints.*** A set of constraints on the resolution of interdependent decisions.

Form and Structure

A Decision Model can be represented by one of the following forms:

- List of questions
- Tabular format

In the question-list format, each decision is phrased as a question and a non-empty set of valid answers. The question identifies the decision that an application engineer must make. The set states all permissible answers to that question.

In the tabular format, each horizontal row in the table expresses a decision specification. The horizontal row is divided into columns. A column identifies either the decision that an application engineer must make, the permissible answers for that decision, or a brief description of the decision.

Each decision and each decision group must have a unique identifier. Domain engineers use this identifier when they define adaptable work products. Each decision group has one list or table that is labeled with a mnemonic appropriate to the group. The group is a set of related decisions. Each entry is an independent decision that has its own distinct mnemonic label, a specification of allowed values that can resolve the decision, and a short explanation of the meaning of the decision.

If a set of related decisions is always resolved as a unit, you can define the set to be a composite decision. Composite decisions are shown in tabular form using a combination of the composite of indicator and indentation. If the application engineer can choose to resolve one (and only one) decision from a set of

alternatives, you can define the set to be an alternative decision. Alternative decisions are shown in tabular form using a combination of the alternative of indicator and indentation.

You can also use a tabular format to specify constraints on decision making. Decision constraints may be either structural or dependency. In both cases, a decision group (the Decision Group column) is specified as the focus of the decision constraint. A structural constraint is a decision constraint that limits the number of instances of a decision group in an Application Model. Valid entries include exactly-one, one-or-more, zero-or-one, zero-or-more, and one-for-each X, where X corresponds to other identified decision groups. A dependency constraint is a decision constraint that specifies how decisions made by an application engineer affect subsequent decisions.

Example DE.2.2.1-1 illustrates a fragment of a Decision Model for the TLC domain. The figure portrays decision groups (e.g., Street, Lane_Group) and their corresponding decisions, along with appropriate constraints.

Verification Criteria

- Every decision must be an elaboration of one or more variability assumptions.
- The Decision Model must accommodate all variability assumptions.

3. PROCESS DESCRIPTION

The Decision Model Activity consists of three steps shown in Figure DE.2.2.1-1.

3.1 PROCEDURE

Follow these steps for Decision Model Activity.

Step: Identify Decisions

Action Identify the decisions that application engineers can make to resolve all of the variations for a system in the domain.

Input Variability assumptions

Result Decision specifications

Heuristics

- Derive decisions directly from variability assumptions. You must have (at least) one decision for each variation specified in the assumptions. You will likely derive multiple decisions from a single variability assumption; each decision is an elaboration of some aspect of the basic variability.
- Keep in mind that the relevant decisions are those concerning system generation time, rather than run-time variation. If you followed a similar heuristic in identifying Domain Assumptions, run-time decisions should not be an issue here. Your focus now should be on how members of a product family differ, rather than on ways in which a member varies its behavior at run-time. However, if members of a product family have variable run-time behavior, then a valid decision may concern whether or how a particular member varies its behavior.

```
Traffic_Light_Controller: composed of
    Schedule: one of (Fixed_Schedule, Programmable)     {designates the type of traffic light sequence scheduling
                                                            the TLC system must accommodate}
    Geometry: one of                                    {intersection geometry}
        X: list length 4 of Street                      {street characteristics for an X intersection}
        T: list length 3 of Street                      {street characteristics for a T intersection}

Fixed_Schedule: composed of
    Start_Time: (0:00 .. 23:59)                         {start time for this traffic light sequence schedule}
    Stop_Time: (0:00 .. 23:59)                          {stop time for this traffic light sequence schedule}
    ....

Street: composed of
    Name: identifier                                    { street name}
    Right_Turn_Lanes: Lane_Group                        {characteristics for the right-hand turn lanes}
    Left_Turn_Lanes: Lane_Group                         {characteristics for the left-hand turn lanes}
    Through_Lanes: Lane_Group                           {characteristics for the through lanes}
    Pedestrian_Crosswalk: one of (Xwalk, NO_Xwalk)      {designates the presence of a pedestrian crosswalk
                                                            for this street}
    Crosswalk_Button: one of (CB, NO_CB)                {designates the presence of a pedestrian crosswalk
                                                        pushbutton for this street}
    ....

Lane_Group: composed of
    Number_of_Lanes: numeric(1..2)                      {number of traffic lanes in this Lane_Group}
    Sensor: one of (Sensor, NO_Sensor)                  {indicates whether there is a traffic monitoring
                                                            device for each lane in this Lane_Group}
    ....

Project_Information: composed of
    Name: identifier                                    {name for the TLC system}
    Mnemonic: identifier                                {TLC system mnemonic}
    ....
-------------------------------------------------------------------------------

Constraints
    - The number of through lanes for Street(1) must be the same as for Street(3).
    - There can be at most 4 different schedules in the Fixed_Schedule.
    - A Through_Lanes group must be specified for each Street.
    - ....
```

Example DE.2.2.1-1. Fragment of TLC Decision Model

- If a variability assumption asserts that a certain characteristic of systems in the domain is variable without saying exactly how it varies, you must determine exactly how the characteristic can vary. Specify the precise type of information that will resolve a decision.

- Avoid routinely providing decisions that dictate arbitrary implementation limits (e.g., maximum number of users) unless those limits reflect a policy decision. Optimization of a system requires adequate flexibility.

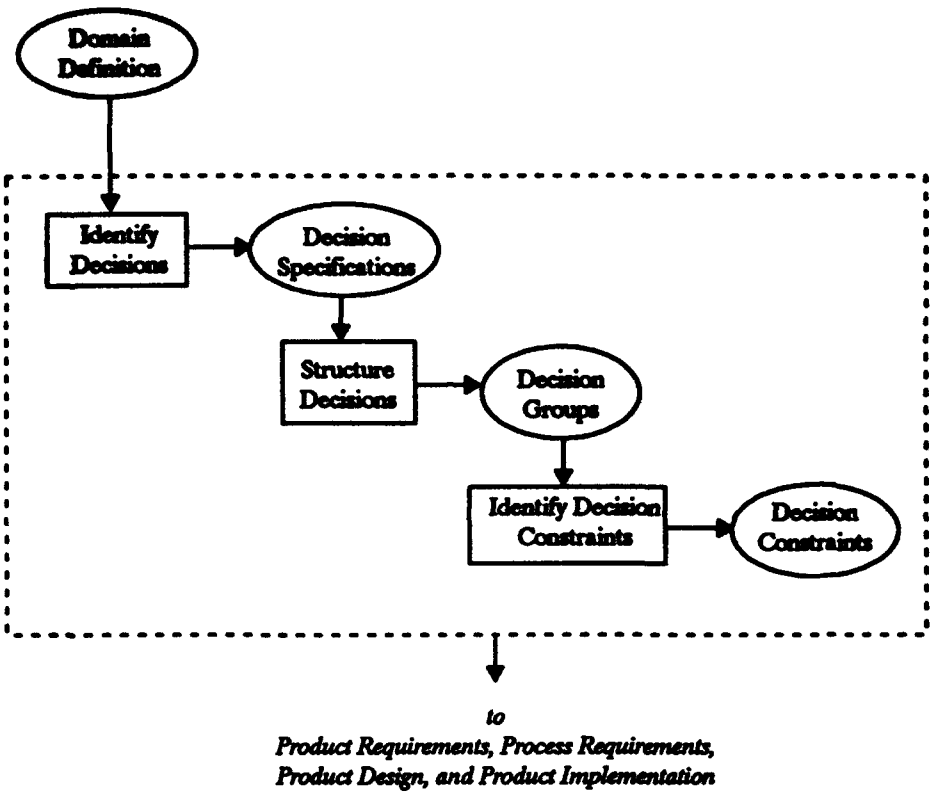


Figure DE.2.2.1-1. Decision Model Process

Step: Structure Decisions

Action Organize decisions into logically-related and interconnected groups.

Input Decision specifications

Result Decision groups

Heuristics

- Each decision group should represent a coherent and cohesive concept to domain experts. Such concepts usually have recognizable names. A concept may be independent of other concepts, or may be an aggregate concept that unifies other simpler concepts. In other words, a decision group may include both individual decisions and decision groups as elements.
- Structure the set of decisions based on the principle of separation of concerns (Dijkstra, Dahl, and Hoare, eds. 1972). For example, create a decision group for decisions that correspond to features of a single, physically-distinct entity.
- Group together mutually-dependent decisions, i.e., those that are unlikely to change independently. Domain experts often rely on a single concept that ties dependent decisions together.
- Group together decisions that repeat. For example, if you need to describe multiple types of a particular device, the engineer may make similar

decisions for each type. You can group these decisions to create a single concept as a focus for decisions.

- Group together decisions if they are derived either from a corresponding single variability assumption or from separate assumptions that were grouped in the Domain Definition. A single assumption that motivates several decisions often represents a single concept, while assumption groupings often suggest how domain experts organize their thoughts about such systems.
- The principles of database schema normalization form a valid model for this step. As is the case with normalization, the goal here is to identify and organize a set of concepts without redundancy or inconsistency.
- Define explicit logical connections between the decision groups. These define the relationships between the decision groups.

Step: Identify Decision Constraints

Action Define structural and dependency constraints that limit how decisions are resolved.

Input Decision groups

Result Decision Model

Heuristics

- Define a structural constraint for each decision group; specify limits on when the group can validly occur in an Application Model.
- Define a dependency constraint whenever one decision narrows the resolution that the application engineer can provide for another decision.
- You may sometimes create decision groups where the cross-product of the decision specifications implies family members that do not exist. You should examine existing systems and specify constraints that omit these members from the Decision Model.

3.2 Risk Management

Risk The Decision Model is inadequate for descriptions of intended systems.

Implication The domain will not provide effective support for planned projects.

Mitigation Try to describe one or more existing systems in terms of the Decision Model. Review these descriptions with experienced engineers to identify erroneous assumptions or unacceptable limitations.

Risk The decision space is too large or complex.

Implication Effort required to develop the Decision Model and subsequent adaptable work products will exceed a reasonable level.

Mitigation

- Focus on a set of well-understood decisions and make the assumption, explicitly, that the other decisions have fixed values (i.e., temporarily constrain them to be commonalities). Plan to relax these assumptions in subsequent iterations, or, in extreme cases, suggest that the Domain Definition Activity consider narrowing the domain scope.
- Reorganize the decision space to achieve a more effective separation of concerns.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity

Response Describe the inadequacies in the Domain Definition and suggest appropriate refinements. Proceed with Decision Model, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Decision Model that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency The Decision Model fails to support all the variation described in the Domain Definition.

Source

- Product Requirements Activity
- Product Design Activity

Response Refine the Decision Model to be consistent with the Domain Definition.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source	• Product Requirements Activity • Process Requirements Activity • Product Design Activity • Product Implementation Activity
Response	Refine the Decision Model to correct inadequacies.
Contingency	The structure or content of the Decision Model conflicts with domain experts' conception of an Application Model.
Source	Process Requirements Activity
Response	Refine the Decision Model to support an Application Modeling Notation acceptable to domain experts.

DE.2.2.2. PRODUCT REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Product Requirements Activity is an activity of the Domain Specification Activity for creating Product Requirements. A requirements specification describes needs that are satisfied by creating an Application Product. Needs are expressed in terms of the required behavior and operational environment of an envisioned application. Similarly, Product Requirements is a requirements specification that is adaptable to the decisions supported by the product family's Decision Model. The Product Requirements describes the set of problems solved by the members of a product family. By applying the decisions that characterize a particular product (i.e., its Application Model) to the Product Requirements, a standardized description of that product is produced. A Product Requirements gives meaning to an Application Model as a description of a member of a product family.

1.1 Objectives

The objective of the Product Requirements Activity is to define the requirements for a product family. The specification must be adaptable to decisions allowed by the product family's Decision Model.

1.2 Required Information

The Product Requirements Activity requires the following information:

- Domain Definition
- Decision Model

1.3 Required Knowledge and Experience

The Product Requirements Activity requires domain and software knowledge and experience in:

- The nature, purpose, and use of work products for existing applications
- The issues that application engineers must resolve in constructing applications in the domain
- The concepts and structures that are appropriate for describing the behavior and operational environment of applications in the domain
- The principles and use of an appropriate software requirements specification method (e.g., informal, structured, semi-formal, or formal [Heninger 1980])

2. PRODUCT DESCRIPTION

Name

Product Requirements

Purpose

Product Requirements specify the requirements of members of a product family. Product Requirements also define the meaning of an Application Model created in accordance with the corresponding product family's Decision Model. You can use the Product Requirements to understand (and explain to application engineers) the implications of decisions in an Application Model (which describes the problem solved by an application).

Content

The Product Requirements is an adaptable requirements specification for a product family. A specification contains four types of information:

- ***Concept.*** An overall characterization of purpose and objectives.
- ***Context.*** A characterization of the relevant environment and relationships within it.
- ***Content.*** A characterization of the expressed or contained substance, meaning or behavior, and scope.
- ***Constraints.*** A characterization of limits and demands on context of use or content.

As a whole, this information is sufficient to characterize each particular member of a product family as implied by the decisions allowed by the family's Decision Model.

Form and Structure

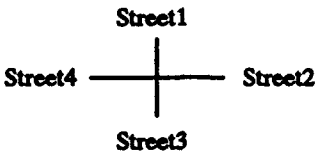
Product Requirements may be expressed in any well-defined form, for example:

- Structured, informal text
- Assertions
- A formal or semi-formal specification

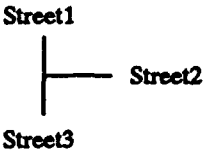
The assertions form of Product Requirements is a set of assertions that describe the (black-box) behavior of applications in the domain. Assertions may be simple or parameterized to reflect decisions defined in the Decision Model. Assertions can be structured into a hierarchy to facilitate separation of concerns.

For all forms, parameterization can be used to express the effects of decisions on Product Requirements. A metaprogramming notation can describe text substitution, conditional inclusion, and iteration over repetitive decisions. Example DE.2.2.2-1 illustrates a fragment of a Product Requirements for the TLC domain. This fragment depicts a portion of the content (e.g., externally visible behavior) and context (e.g., inputs from the environment) of systems in the

The Traffic Light Control Software System (TLC) controls the traffic light sequence for an intersection. The streets are laid out in the following configuration. <if Traffic_Light_Controller.Geometry = X then>



<else>



<endif>

Each traffic light sequence in the intersection is coordinated with other traffic lights in the intersection. The intersection arms have the following characteristics.

<forall streets S in Traffic_Light_Controller.Geometry>
    Street (<S.Name>):
        <if S.Right_Turn_Lanes specified then>
        - <S.Right_Turn_Lanes.Num_of_Lanes> right turn lanes
        <endif>
        <if S.Left_Turn_Lanes specified then>
        - <S.Left_Turn_Lanes.Num_of_Lanes> left turn lanes
        <endif>
        ....
<endfor>

....

<if there exists at least one Street such that Street.Crosswalk_Button = CB then>
The Crosswalk_Button device interrupts the TLC system each time the crosswalk button is pushed. The message received from this device has the following characteristics:

    ....

<endif>
....

The TLC system also has an interface to a real-time clock. The clock is used by the TLC system to determine when to activate a traffic light cycle (in the absence of any trip mechanisms in the streets) and the duration of each traffic light indicator in a traffic light sequence. A TLC system also uses the real-time clock to keep track of the current time-of-day to determine how quickly it must respond to signals received from trip mechanisms or pedestrian crosswalk push buttons.

....

Example DE.2.2.2-1. Fragment of TLC Product Requirements

TLC domain. This fragment also depicts the use of parameterization (in terms of appropriate decisions from the product family's Decision Model shown in Example DE.2.2.1-1) to express requirements that characterize particular members of the product family. For example, the block of text describing the

message format received from the crosswalk button device is only included in the Product Requirements when there is at least one Street in the TLC system which has that device.

COMMENT: A black-box description for Product Requirements reduces the tendency to choose software design and implementation solutions prematurely. By parameterizing the description, it will apply equally well to all members of the product family. Figure DE.2.2.2-1 illustrates how an Application Model for a product is applied to a parameterized Product Requirements to yield a standardized software-requirements specification for that product.

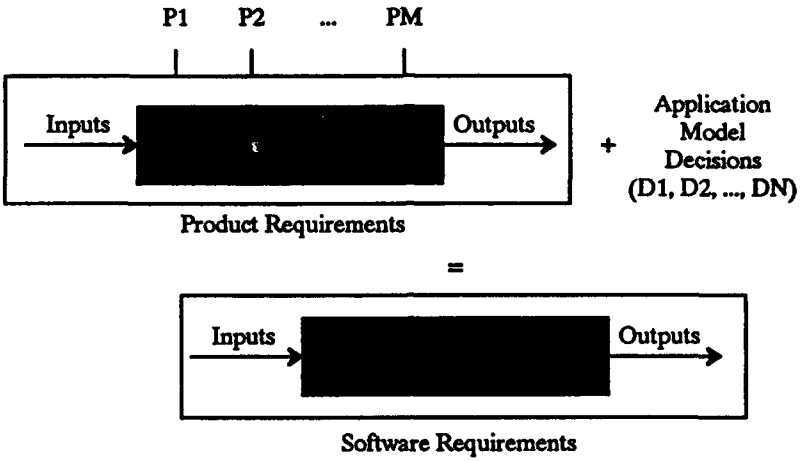


Figure DE.2.2.2-1. Instantiating Product Requirements

Verification Criteria

- All implicit requirements must be an elaboration of one or more commonality assumptions.
- The Product Requirements must elaborate all commonality assumptions.
- If decisions that characterize a particular particular system are applied to the Product Requirements, the result should be a requirements specification that describes that system correctly.

3. PROCESS DESCRIPTION

The Product Requirements Activity consists of four steps shown in Figure DE.2.2.2-2.

3.1 PROCEDURE

Follow these steps for the Product Requirements Activity.

Step: Define the Concept

Action Describe overall purpose and objectives for the product family.

Input

- Domain Definition
- Decision Model

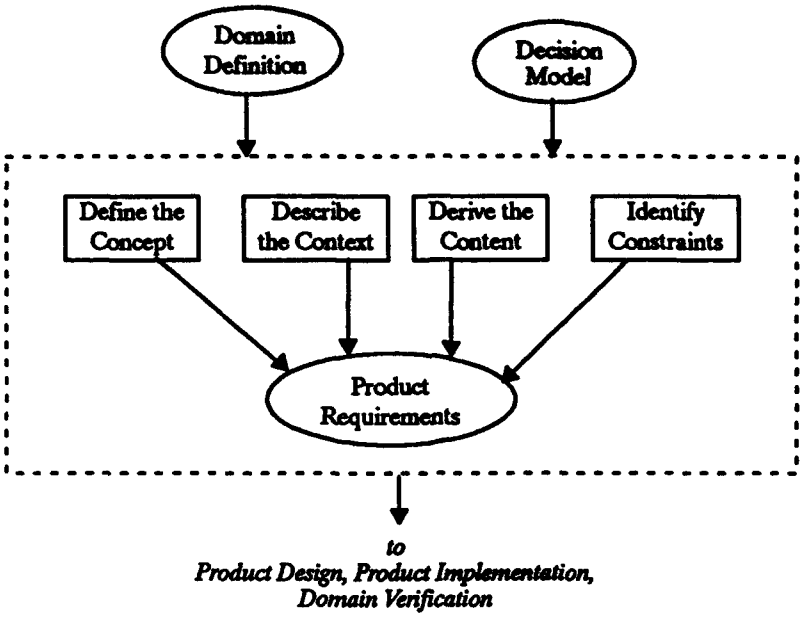


Figure DE.2.2.2-2. Product Requirements Process

Result Product Requirements: Concept

Heuristics

- Select a requirements method that best supports an abstract description of the product family.
- Describe an application's purpose and concept of operations. For associated documents, describe objectives and provide an overview of content.
- Examine commonality assumptions to identify additional aspects of concepts that apply to all members of the product family.
- Examine variability assumptions to derive additional aspects of concepts that distinguish particular members of the product family. Capture these requirements by parameterizing concept descriptions in terms of the appropriate decisions from the product family's Decision Model.
- Examine Legacy Products from the Domain Definition to derive additional concept requirements that apply to all or some members of the product family. For common concepts, determine whether these will apply to future members of the product family. Describe variations in concepts in terms of decisions in the product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Describe the Context

Action Describe the relevant environment and relationships for the product family.

Input

- Domain Definition
- Decision Model

Result Product Requirements: Context

Heuristics

- Describe the environment (e.g., devices, connected systems, users/roles) in which an application operates. Also describe the interface to that environment (e.g., inputs from the environment, outputs to the environment). For documents, describe their audience, expected benefits, and relation to other work products.
- Examine commonality assumptions to derive additional context requirements that apply to all members of the product family.
- Examine variability assumptions to derive additional context requirements that characterize a particular member of the product family. Capture these requirements by parameterizing context descriptions in terms of the appropriate decisions from the product family's Decision Model.
- Examine Legacy Products to derive additional context requirements that apply to all or some members of the product family. For common requirements, determine whether these will apply to future members of the product family. Describe variations in context in terms of decisions in the product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Derive the Content

Action Describe the externally visible behavior and information content of applications in the product family.

Input

- Domain Definition
- Decision Model

Result Product Requirements: Content

Heuristics

- Define an information model of an application's information content. Describe the externally visible behavior of an application, including conceptual modes of operation and functions that produce output to the environment. For documents, identify the topics to be covered.
- Examine commonality assumptions to derive requirements that apply to all members of the product family.
- Examine variability assumptions to derive additional requirements that characterize particular members of the product family. Capture these requirements in the Product Requirements by parameterizing content descriptions in terms of the appropriate decisions from the product family's Decision Model.

- Examine Legacy Products of the Domain Definition to identify and extract additional common and varying requirements for content. For common content, consider whether these requirements will apply to future members of the product family. Describe variations in content in terms of decisions in the product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

Step: Identify Constraints

Action Describe limits and demands on members of the product family.

Input
- Domain Definition
- Decision Model

Result Product Requirements: Constraints

Heuristics
- Describe environmental limits (e.g., reliability and responsiveness of devices), performance demands (e.g., timing and accuracy goals for inputs and outputs), and design or implementation dictates (e.g., targeting of software to operate on particular computers). For documents, describe any formatting guidelines.
- Examine commonality assumptions to derive additional constraints that apply to all members of the product family.
- Examine variability assumptions to derive additional constraints that characterize particular members of the product family. Capture these requirements by parameterizing constraint descriptions in terms of the appropriate decisions from the product family's Decision Model.
- Examine Legacy Products to derive additional constraints that apply to all or some members of the product family. For common constraints, determine whether these will apply to future members of the product family. Describe variations in constraints in terms of decisions in the product family's Decision Model. If no such decisions exist, then decide whether you want to extend the Decision Model to accommodate these requirements variations.

3.2 Risk Management

Risk Product Requirements do not capture all Domain Assumptions accurately.

Implication A derived requirements specification will not accurately describe the problem that the corresponding product family member solves.

Mitigation Create an Application Model for one or more existing systems and derive their respective requirements specifications. Review the specification with

customers, experienced engineers, and domain experts to identify any inaccuracies.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity

Response Describe the inadequacies in the Domain Definition. Proceed with Product Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete Product Requirements that satisfy the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures that will make them more effective.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity

Response Describe the inadequacies in the Decision Model. Proceed with Product Requirements, and document any assumptions made regarding the inadequate portions of the Decision Model.

4.2 Feedback From Product Consumers

Contingency Product Requirements fail to describe a product family that is consistent with the Domain Definition.

Source Domain Management Activity

Response Modify the Product Requirements to be consistent with the Domain Definition.

Contingency	Product Requirements are incomplete, ambiguous, or inconsistent.
Source	• Product Design Activity • Product Implementation Activity
Response	Refine the Product Requirements to correct inadequacies.

This page intentionally left blank.

DE.2.2.3. PROCESS REQUIREMENTS ACTIVITY

1. GETTING STARTED

The Process Requirements Activity is an activity of the Domain Specification Activity for creating Process Requirements. Process Requirements is a specification of a standardized Application Engineering process for a domain and an associated Application Modeling Notation. A process specification tailors and standardizes the activities, methods, and procedures by which Application Engineering is practiced within a domain. This process replaces conventional, domain-independent approaches to software development.

As a part of a standardized Application Engineering process, application engineers create an Application Model that describes a required system. An Application Modeling Notation defines the form and mechanisms that application engineers can use to describe and evaluate an Application Model for products in the domain. This notation must be usable by engineers knowledgeable in domain concepts and must produce an Application Model that is an accurate model of the resulting product. The information content expressible with an Application Modeling Notation for a domain must be equivalent to the information content defined by the Decision Model for that domain. The organization and form of the notation, however, are tailored to the needs of application engineers.

1.1 Objectives

The objective of the Process Requirements Activity is to define a standardized process for Application Engineering within a domain and an accompanying Application Modeling Notation for creating a precise description of a required product. The process should be tailored to the needs of the organization so that established productivity (process efficiency and product quality) goals are most attainable.

1.2 Required Information

The Process Requirements Activity requires the following information:

- Domain Definition
- Decision Model

1.3 Required Knowledge and Experience

The Process Requirements Activity requires domain and software knowledge and experience in:

- The conventional life-cycle process of systems in the domain and the role of customers and standards in that process

- How application engineers resolve issues in constructing systems in the domain
- The concepts and structures that are convenient forms by which domain experts communicate concerning the distinguishing features of systems in the domain

2. PRODUCT DESCRIPTION

Name Process Requirements

Purpose Process Requirements define a standard process that application engineers follow to develop and evolve systems in the domain. The process described in the Process Requirements may be manual or may incorporate varying levels of automation.

Content The Process Requirements work product consists of:

- ***Process Specification.*** A definition of the work products, activities, and process of Application Engineering. For each activity, Process Specification describes its purpose, the work products created, and interactions with other activities.
- ***Application Modeling Notation Specification.*** A description of the form of the Application Modeling Notation. The description of the notation consists of:
  - ***Presentations.*** The form of each set of logically-related decisions from the Decision Model that application engineers tend to treat as a unit.
  - ***Structure.*** The structure of the Application Modeling Notation, which organizes Presentations into a decision process based on dependencies and constraints among decisions.

Form and Structure The Process Specification defines the products, activities, and process of Application Engineering. Each application engineering work product has a specified content and form. The form of a product may be defined explicitly or by reference to customer or industry standards. Each activity of Application Engineering is described by its purpose, the work products to be created, a procedure that defines and organizes the steps of the activity, and interactions with other activities. The Application Engineering process is described as a set of Activities (e.g., Specify Model, Assess Model) that are to be performed in a specified (partial) order. Each activity consists of a number of Steps (e.g., Specify Platform and Start Simulation). The description of the activity includes a specification of the order in which steps may be performed. Each step is specified in terms of a presentation that describes the particular information the application engineer sees during that step and a set of actions that he can perform.

The Application Modeling Notation can be specified as a set of paper or automated forms that allow application engineers to make requirements or

engineering decisions about the needed product. By completing these forms, application engineers construct an Application Model for a product. In addition, the Notation organizes the forms into a network that defines a sequence in which application engineers can address any particular decision.

The description of an activity in an Application Modeling Notation is written in terms of a standard presentation paradigm. A presentation paradigm defines a generic form in which information may be presented interactively and is conceived as a notational aid for describing the Application Modeling Notation. Presentation paradigms for simple decisions are textual, graphical, or iconic. Presentation paradigms for grouped decisions organize simpler presentations into aggregate (possibly hybrid) forms (e.g., textual, graphical, tabular).

Verification Criteria

- All decisions specified in the Decision Model should be accessible through the Application Modeling Notation.
- The Application Modeling Notation should support only those decisions that can be expressed in some equivalent way in the Decision Model.
- The Process Requirements must enforce all of the constraints specified in the Decision Model.

3. PROCESS DESCRIPTION

The Process Requirements Activity consists of two steps shown in Figure DE.2.2.3-1.

3.1 PROCEDURE

Follow these steps for the Process Requirements Activity.

Step: Design a Process

Action Specify a process for creating and evaluating an Application Model and the generation of work products from the Application Model. Define the process by organizing the Presentations defined in the Application Modeling Notation into activities and specifying ordering constraints and prerequisites that structure the process. Define the points at which analyses may be performed and work products may be produced.

Input Domain Definition

Result Process Specification

Heuristics

- Identify the deliverables that the Application Engineering process must produce. This set of products is determined by the needs of customers. The form and content of each product should be defined in keeping with corporate, customer, or industry standards, as appropriate. If an adequate standard is not available for any product, create a standard for your organization.

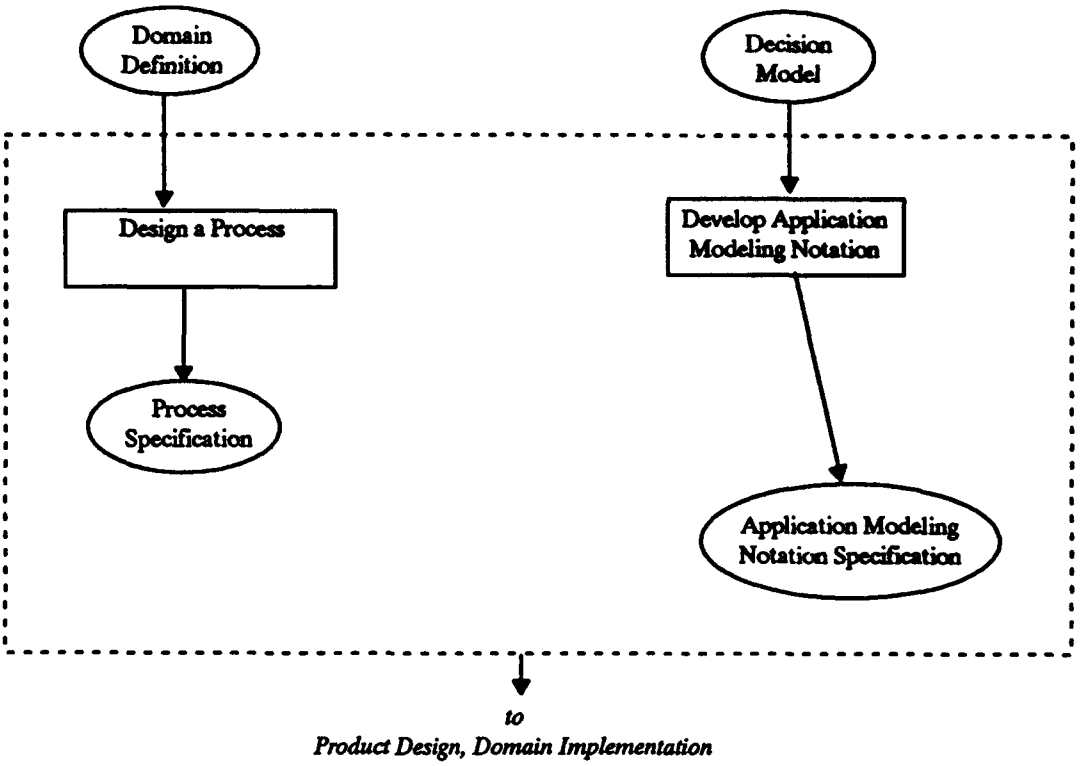


Figure DE.2.2.3-1. Process Requirements Process

- Identify additional (i.e., intermediate or auxiliary) work products that result from the Application Engineering process. These work products support the need for quantitative and qualitative analyses of deliverable work products and the Application Engineering process.

- Define the activities to be performed during the process in terms of necessary inputs, expected results, and procedures for attaining those results. Identify risks in attaining acceptable results and interdependencies with other activities. Define the checks to be performed during the process and specify when they are to be performed. Specify what is to be done when checks fail.

- The process should be described to a level of detail that allows you to formalize standard practice and the engineer's past experience. For example, Project Management, Delivery and Operation Support, and configuration management practices may be based on conventional experience with those tasks.

- Creating an Application Model and using that Application Model to create work products are essential elements of any Application Engineering process. As a starting point, consider the prototypical, iterative Application Engineering process described in Section AE (consisting of Project Management, Application Modeling, Application Production, and

Delivery and Operation Support activities), refined appropriately to the needs of your projects.

Step: Develop Application Modeling Notation

Action Organize decisions into a set of presentations in terms of a set of standard presentation paradigms.

Input Decision Model

Result Application Modeling Notation Specification

Heuristics

- Identify a set of presentation paradigms; consider the ways domain experts generally represent various problem facets in a manner consistent with the ways decisions are grouped in the Decision Model. Identify a set of paradigms that are adequate to represent decisions in all the facets of a problem description.
- First, identify presentations by describing each decision group in the Decision Model as a presentation, then describe the presentation in terms of allowed decisions and any auxiliary (e.g., labeling, layout) information required by the appropriate presentation paradigm.
- Create the structure of the Application Modeling Notation by considering how presentations interrelate (derivable from how decision groups in the Decision Model interrelate). Some presentations may be meaningful only if accessed in the context of, or in combination with, other related presentations. These relationships determine a reasonable structure for the Notation. The ideal structure for an Application Model is one that domain experts would consider natural and appropriate as a model of a system. Refine the structure by consulting with domain experts as to how decision making is best organized.
- Identify any analyses that the application engineer should be able to perform on individual presentations and on the Application Model as a whole. To be most effective, these analyses should provide insights into the functional and operational characteristics of the described system in terms of alternative resolutions of decisions, rather than in terms of the details of generated work products.

3.2 Risk Management

Risk The Application Engineering process will not meet all of the needs of a project.

Implication Projects will have to modify the process in an ad hoc fashion and work around incorrect assumptions of the Process Requirements.

Mitigation Review the process with experienced project managers to ensure that it encompasses all activities required of a project, those products that customers require, and those

products that benefit a project. Variations in project needs should be anticipated and supported.

Risk The Application Modeling Notation will not be usable by application engineers.

Implication Application engineers will have difficulty creating valid Application Models.

Mitigation Review the Notation with experienced engineers to ensure that it is understandable and that it uses terminology and notations familiar to application engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 FEEDBACK TO INFORMATION SOURCES

Contingency The Domain Definition is incomplete, ambiguous, or inconsistent.

Source Domain Definition Activity

Response Describe the inadequacies in the Domain Definition. Proceed with Process Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete the Process Requirements to satisfy the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity

Response Describe the inadequacies in the Decision Model. Proceed with Process Requirements, and document any assumptions made regarding the inadequate portions of the Domain Definition.

Contingency The structure or content of the Decision Model conflicts with domain expert's conception of an Application Model.

Source Decision Model Activity

Response Describe the elements of an acceptable Application Modeling Notation not supported by the Decision Model.

4.2 Feedback From Product Consumers

Contingency The Process Requirements violate constraints on the Application Engineering process established in the Domain Definition.

Source Domain Management Activity

Response Evolve the Process Requirements to be consistent with the Domain Definition.

Contingency The Process Requirements are incomplete, ambiguous, or inconsistent.

Source Process Support Development Activity

Response Refine the Process Requirements to correct inadequacies.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for a particular project.

Source Project Support Activity

Response

- Determine that the benefits of process standardization outweigh the interests of the particular project.
- Evolve the Process Requirements to reflect the generally-applicable insight of this project's experience or to be adapted to the particular conditions of concern.

This page intentionally left blank.

DE.2.2.4. PRODUCT DESIGN ACTIVITY

1. GETTING STARTED

The Product Design Activity is an activity of the Domain Specification Activity for creating a Product Design. A Product Design specifies the design for product family, rather than for a single product. A design describes an application that solves a specified problem. Similarly, a Product Design is a design that varies according to the decisions supported by the product family's Decision Model. By applying the decisions that characterize a particular product to the Product Design, a standardized design of that product is produced.

1.1 Objectives

The objective of the Product Design Activity is to create a design for a product family. The product family's design must satisfy its Product Requirements and must be adaptable to the decisions allowed by the family's Decision Model.

1.2 Required Information

The Product Design Activity requires the following information:

- Decision Model
- Product Requirements
- Legacy Products

1.3 Required Knowledge and Experience

The Product Design Activity requires domain and software knowledge and experience in:

- The principles and use of an appropriate design method
- How systems in the domain are designed, including an appreciation of typical engineering tradeoffs to be resolved
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name Product Design

Purpose A Product Design specifies the design of members of a product family.

Content The Product Design consists of the following parts:

- ***Product Architecture.*** A specification of the internal organization of each application engineering work product that can be produced for the family (see Section DE.2.2.4.1).
- ***Component Design.*** A specification of the design of each of a set of Adaptable Components that can be adapted to compose deliverable application engineering work products for the family (see Section DE.2.2.4.2).
- ***Generation Design.*** A specification of how a product family's Application Model is used to select, adapt, and compose Adaptable Components to create work products that satisfy the Product Requirements and Product Architecture (see Section DE.2.2.4.3).

Verification Criteria

- All aspects of Product Requirements are traceable into the Product Design. All variations in Product Requirements have equivalent variations in the Product Design.
- The Product Design satisfies the verification criteria appropriate to the specific design method used in creating it.

3. PROCESS DESCRIPTION

The Product Design Activity consists of the three steps shown in Figure DE.2.2.4-1.

3.1 PROCEDURE

Follow these steps for the Product Design Activity.

Step: Product Architecture Activity

Action Create design structures that characterize the internal organization of members of the product family.

Input

- Product Requirements
- Legacy Products

Result Product Architecture

Heuristics

- Create multiple design structures (each portraying a different perspective) for a product family.
- Ensure that the product family's Product Architecture applies to all members of that family.

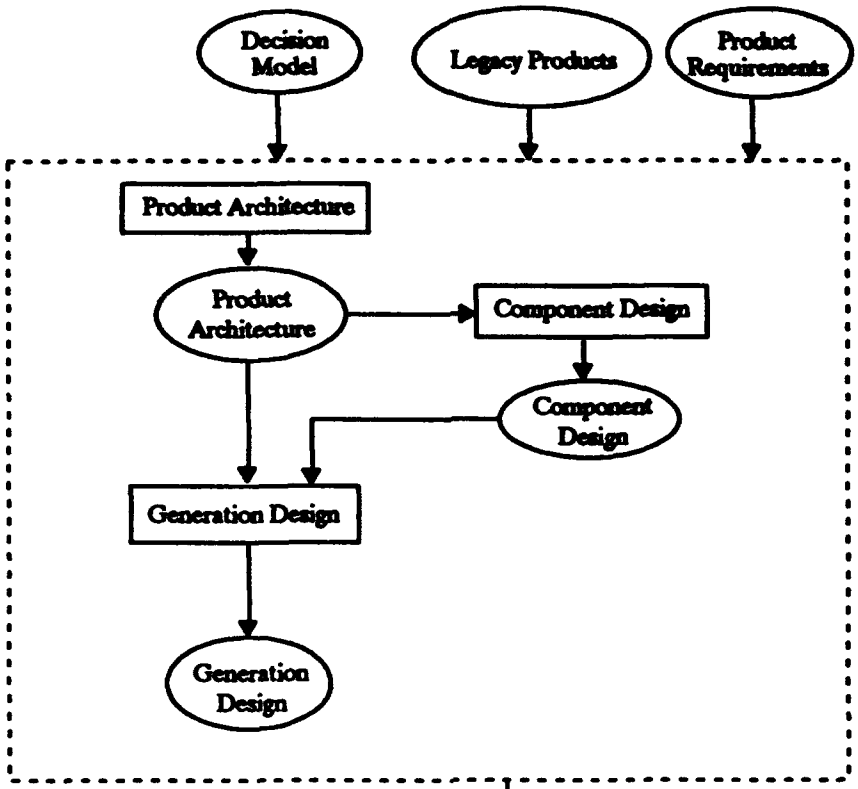


Figure DE.2.2.4-1. Product Design Process

Step: Component Design Activity

Action	Create a Component Design for each of a set of Adaptable Components that compose a product family as identified by the Product Architecture.
Input	• Product Requirements • Product Architecture • Legacy Products
Result	Component Design
Heuristics	Ensure that the Component Design satisfies relevant aspects of the Product Architecture and Product Requirements.

Step: Generation Design Activity

Action	Specify a precise procedure of how members of a product family are derived from Adaptable Components based on the decisions in an Application Model.
Input	• Decision Model • Product Architecture • Component Designs
Result	Generation Design
Heuristics	• Decide how the decisions for a product family determine the form and content of application engineering work products. • Specify the design by describing how Adaptable Components are selected, adapted, and composed according to the decisions in the work product family's Decision Model.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Decision Model is incomplete, ambiguous, or inconsistent.
Source	Decision Model Activity
Response	Describe the inadequacies in the Decision Model. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Decision Model.
Contingency	The Product Requirements are incomplete, ambiguous, or inconsistent.
Source	Product Requirements Activity
Response	Describe the inadequacies in the Product Requirements. Proceed with Product Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Design that satisfies the Domain Plan as closely as possible.

Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency	Suggestions are made for Product Design changes to exploit unforeseen opportunities, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.
Source	• Product Implementation Activity • Process Support Development Activity
Response	• Revise the Product Design. • Refer to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Product Design does not satisfy the Product Requirements.
Source	Domain Verification Activity
Response	Modify the Product Design to be consistent with the Product Requirements.
Contingency	The Product Design is incomplete, ambiguous, or inconsistent.
Source	Product Implementation Activity
Response	Refine the Product Design to correct inadequacies.

This page intentionally left blank.

DE.2.2.4.1. PRODUCT ARCHITECTURE ACTIVITY

1. GETTING STARTED

The Product Architecture Activity is an activity of the Product Design Activity for creating a Product Architecture. An architecture, for a given work product, is one or more design structures that define the internal organization of that work product from different perspectives. Similarly, a Product Architecture is a description of the internal organization of a product family. A Product Architecture includes the architecture of each of the work product families that make up the product family.A Product Architecture varies according to the decisions supported by the product family's Decision Model. A Product Architecture describes a standardized architecture for all members of a product family in a domain. By applying the decisions that characterize a particular product to the Product Architecture, a standardized architecture of that product is produced.

For each required application work product family (requirements, design, code, etc.), one design structure must identify a set of Adaptable Components. Application engineers compose instances of these components to create an instance of the work product. Depending on which design method domain engineer's follow, they may also create other structures which provide other views of the behavior or interrelationships of components. In all cases, the structures are in an adaptable form so that Application Engineering can use them to produce any member of the indicated product family.

1.1 Objectives

The objective of the Product Architecture Activity is to define an adaptable architecture for products that can be produced in Application Engineering. Product Architecture is the design of solutions to the problems that Product Requirements describe.

1.2 Required Information

The Product Architecture Activity requires the following information:

- Product Requirements
- Legacy Products

1.3 Required Knowledge and Experience

The Product Architecture Activity requires domain and software knowledge and experience in:

- The principles and use of an appropriate software design method (e.g., ADARTS [Software Productivity Consortium 1993])
- How systems in the domain are designed and documented following chosen design and documentation methods

- Typical engineering tradeoffs that must be balanced when designing systems in the domain
- The concepts and practice of metaprogramming (Campbell 1989)

2. PRODUCT DESCRIPTION

Name

Product Architecture

Purpose

The Product Architecture describes the internal organization of members of the application engineering work product family.

Content

The Product Architecture consists of design structures for each application work product. One of the structures must identify the components that make up each of the members of the work product family. Each structure consists of:

- A set of design elements
- A relation that associates elements

For example, a software requirements specification document is one type of application engineering work product. The domain engineers might choose sections of requirements documents as design elements, and "subsection" as the relation among these elements. This describes the structure of a software requirements specification document in enough detail to allow its composition from its constituent elements. The structures developed for a particular domain depend on the particular application work products and the design method used to produce them.

Although the Product Architecture for a particular product may contain multiple design structures, there must be one structure that describes the decomposition of the product into work assignments (e.g., modules, sections). The elements of this structure correspond to components that are to be implemented by Adaptable Components.

The only difference between the design structures specified in the Product Architecture and those specified in a conventional design is that the Product Architecture is parameterized and adaptable, so that it describes the family of products in the domain.

Form and Structure

The form for each structure of a Product Architecture is a textual or graphic network of elements and relations. This representation is then augmented with a suitable technology such as metaprogramming notation to parameterize the structure for adaptation to variations.

Examples DE.2.2.4.1-1, DE.2.2.4.1-2 , and DE.2.2.4.1-3 illustrate fragments of a Product Architecture for the TLC domain (i.e., fragments of the static software architecture for the product family, process structure architecture for the product family, and an architecture for a DOD-STD-2167A Interface Requirements Specification document work product family, respectively). Multiple structures are

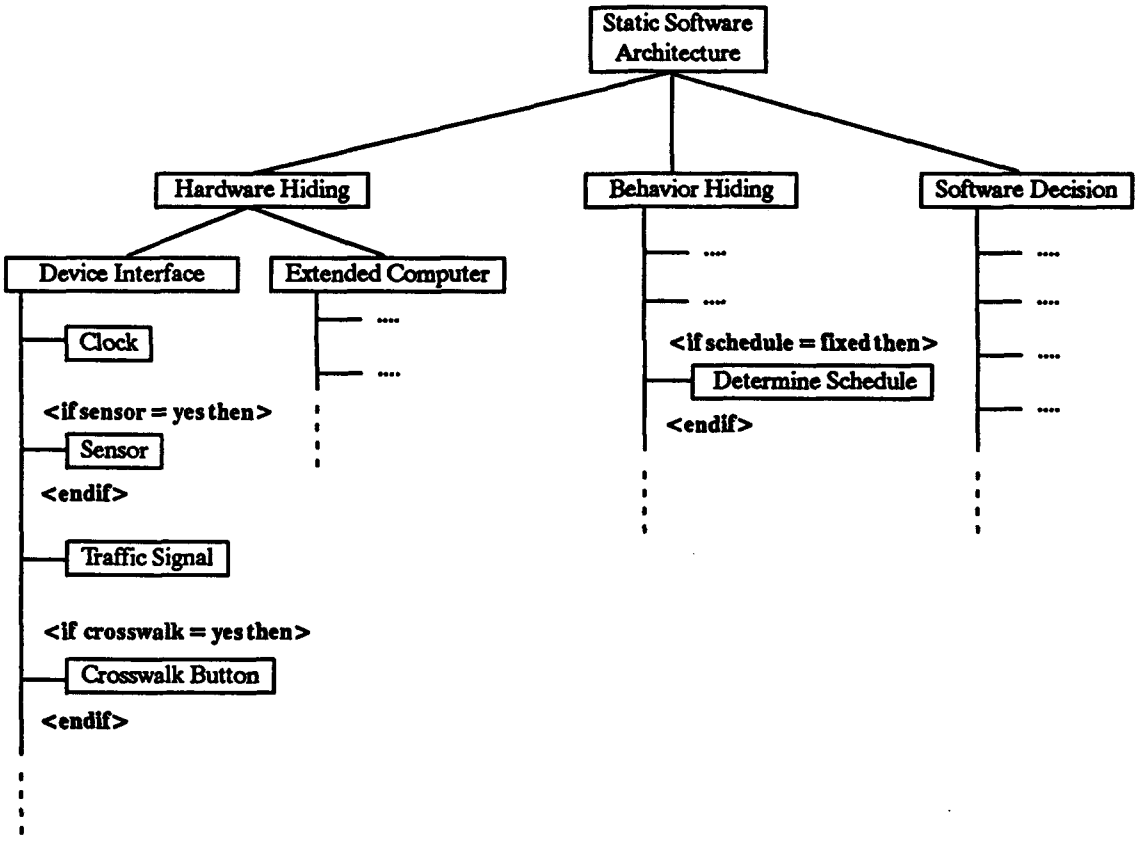
Product Architecture – Static Software Architecture of TLC product family

Instantiation Parameters

crosswalk	one of (yes, no)	{Pedestrian crosswalk push button. A value of yes means that the software information hiding architecture must include components to support the Pedestrian Crosswalk Push Button. A value of no means those components are not included.}
schedule	one of (fixed, dynamic)	{....}
sensor	one of (yes, no)	{....}
....		

Instantiation Constraints

None



Example DE.2.2.4.1-1. Fragment of TLC Product Architecture

needed to fully describe the architecture of the software and documentation for the product family. Notice that the Product Architecture of the document is expressed as an annotated outline where each numbered heading (e.g., 1. Scope) corresponds to a section of the work product. Preceding each architecture are the decisions that parameterize the respective architecture. The content of these architectures will vary depending on values chosen for the respective decisions. For example, in Example DE.2.2.4.1-3, Section 3.3 will be included only when parameter crosswalk is true. Otherwise, this section is omitted.

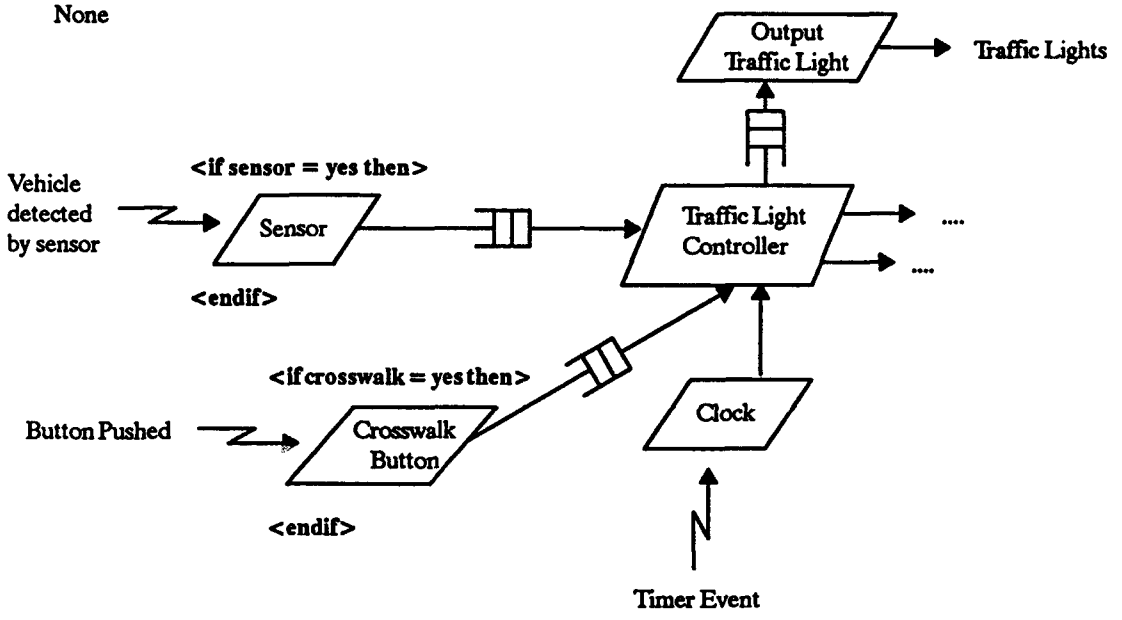
Product Architecture – Process Structure of the TLC product family

Instantiation Parameters

crosswalk	one of (yes, no)	{Pedestrian crosswalk push button. A value of yes means that the software information hiding architecture must include components to support the Pedestrian Crosswalk Push Button. A value of no means those components are not included.}
sensor	one of (yes, no)	{....}
....		

Instantiation Constraints

None



Example DE.2.2.4.1-2. Fragment of TLC Product Architecture

Verification Criteria

- Each Product Architecture structure satisfies the verification criteria established by the specific design method used in its creation.
- The Product Architecture defines all structures for the software and other work products required by the Application Engineering process.

3. PROCESS DESCRIPTION

The Product Architecture Activity consists of two steps shown in Figure DE.2.2.4.1-1.

3.1 PROCEDURE

Step: Identify Work Product Components

Action Develop a structure that describes, as a structured set of components, the internal organization of the products in the family.

Product Architecture – Interface Requirements Specification (IRS) work product family

Instantiation Parameters

crosswalk	one of (yes, no)	{Pedestrian crosswalk push button. A value of yes means that the IRS must include engineering requirements describing the pedestrian crosswalk push button capability of the TLC system. A no value means the IRS must omit these requirements.}
light_schedule	one of (fixed, dynamic)	{Identifier what traffic light sequence scheduling capability the IRS must describe.}
fixed_schedule	composite of	{The IRS must express the requirements for a traffic light sequence as defined by this parameter.}
....		

Instantiation Constraints

- Must provide values for fixed_schedule when light_schedule is 'fixed'
- ...

Interface Requirements Specification Annotated Outline

1. Scope
   ....

2. Applicable Documents
   ....

3. Interface Specification
   {This section specifies the requirements for those interfaces to which this IRS applies.}

3.1 Interface Diagrams
   {This paragraph identifies the interfaces to which this specification applies. One or more interface diagrams, as appropriate, will be provided to depict the interfaces.}

3.2 TLC_to_Clock interface
   ....

<if crosswalk = yes then>

3.3 TLC_to_Crosswalk_Pushbutton
   {This paragraph identifies the TLC_to_Crosswalk_Pushbutton interface and specifies the requirements for the interface and for the data transmitted across the interface.}

<endif>

....

4. Quality Assurance
   {This section shall always state "NONE."}

....

Example DE.2.2.4.1-3. Fragment of TLC Product Architecture

Input

- Product Requirements
- Legacy Products

Result

Product Architecture: Internal Organization

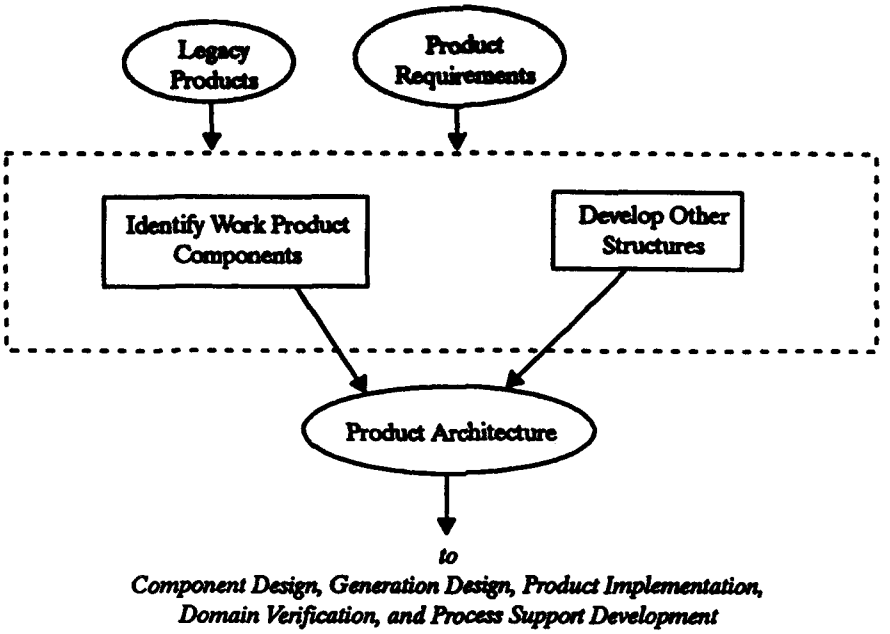


Figure DE.2.2.4.1-1. Product Architecture Process

Heuristics

- The Process Requirements define the required standards for the development of each work product. For software, Process Requirements define which design method is to be followed, which in turn determines the design structures required. Using ADARTS, there are three structures: Information Hiding, Process, and Dependency. For documentation, an annotated outline, in which each section and subsection is listed as a component, is normally sufficient as the design for the work product. For test and delivery support, a hierarchical structure is needed that characterizes each test case or delivery function as a component.
- Each work product is broken down, or decomposed, into a set of components so that the work may be performed independently by multiple component implementors. Using a chosen design method, a structure must be created that determines this decomposition. Using ADARTS, the Information Hiding Structure determines a suitable decomposition into components.
- Each structure is parameterized by a set of decisions that characterize how it is adapted to represent a particular system. A parameterized structure defines a family of structures. The set of structure families must be parameterized consistently, both with each other and with the Product Requirements. The content of each structure must be consistent with the Product Requirements.
- Analyze Legacy Products; look for recurring patterns in their design structures. Recurring patterns suggest portions of a structure that do not vary.
- Use the Product Requirements to determine what characteristics are common to all systems in the domain. These characteristics should correspond to common structures identified in the analysis of Legacy Products.
- Each structure must be adaptable to requirements and design variations among systems. Parameterize the structure to characterize required

variations. Use a metaprogramming notation to record the effects of variations. For example, a conditional statement referencing a variation indicates that a portion of a structure is included only for some systems.

Step: Develop Other Structures

Action Create any other structures required to define the work product fully.

Input
- Product Requirements
- Legacy Products

Result Product Architecture: Alternate Structures

Heuristics
- The chosen design method for software identifies other required design structures. Using ADARTS, these design structures are the Process Structure and the Dependency Structure. Hypertext-based documents would have an analogous alternative structure.
- Alternate structures impose constraints on the implementation of each component of the internal organization. The design method characterizes these constraints.
- Each structure must support a subset of the same variations. The internal organization determines how these variations affect each component. Component variations must account properly for variations in relevant parts of the alternate structures.

3.2 Risk Management

Risk The Product Architecture will not support all features or variations in Product Requirements.

Implication The Product Architecture is not a correct solution to the Product Requirements.

Mitigation Review the Product Architecture with developers of the Product Requirements and experienced designers. Establish traceability of all required features to elements of the architecture. Evaluate whether variations that characterize different products lead to proper architectural variations.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity

Response Describe the inadequacies in the Product Requirements. Proceed with Product Architecture, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Product Architecture that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source Domain Management Activity

Response Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.

4.2 Feedback From Product Consumers

Contingency Suggestions are made for Product Architecture changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source
- Product Implementation Activity
- Process Support Development Activity

Response
- Revise the Product Architecture.
- Refer to Domain Management for future planning.
- Reject the changes due to conflicts with the Domain Definition.

Contingency The Product Architecture does not satisfy the Product Requirements.

Source Domain Verification Activity

Response Modify the Product Architecture to be consistent with the Product Requirements.

Contingency The Product Architecture is incomplete, ambiguous, or inconsistent.

Source
- Product Implementation Activity
- Generation Design Activity

- Component Design Activity

Response Refine the Product Architecture to correct inadequacies.

This page intentionally left blank.

DE.2.2.4.2. COMPONENT DESIGN ACTIVITY

1. GETTING STARTED

The Component Design Activity is an activity of the Product Design Activity for creating a Component Design. The Product Architecture identifies a set of Adaptable Components that are required to implement the work products of a product family. A Component Design is a design specification for one of these Adaptable Components. A set of Component Designs defines a library of Adaptable Components that may be adapted and composed to implement the work products of the product family. Each component must be designed to satisfy relevant aspects of the Product Requirements and all design structures of the Product Architecture.

1.1 Objectives

The objective of the Component Design Activity is to produce a design for an Adaptable Component that satisfies applicable Product Requirements in accordance with its role in the Product Architecture.

1.2 Required Information

The Component Design Activity requires the following information:

- Product Requirements
- Product Architecture
- Legacy Products

1.3 Required Knowledge and Experience

The Component Design Activity requires domain and software knowledge and experience in:

- How components of systems in the domain are designed
- The principles and use of an appropriate design method
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name Component Design

Purpose

A Component Design is a specification for an Adaptable Component that can be used to construct an application or associated work product.

Content

Each Component Design represents a family of components. A Component Design consists of two parts:

- ***Adaptation Specification.*** The Adaptation Specification for an Adaptable Component describes the ways that the component can be tailored via a set of parameters. Each parameter has a name and type to indicate its range of variations. Constraints identify invalid combinations of parameter variations.

- ***Interface Specification.*** The Interface Specification describes the desired characteristics of the implementation of the component. The exact content of the interface specification is particular to the component type and the design method used. To describe the entire family, the interface specification is parameterized with respect to the variations in the Adaptation Specification.

Form and Structure

The Adaptation and Interface Specifications each include textual and tabular information. The form of an Adaptation Specification is the same for all types of components and includes the following information:

- *Name.* Name of the Adaptable Component.

- ***Instantiation Parameters.*** Adaptation parameters for the component, including the name, type, and description of each parameter.

- ***Instantiation Constraints.*** Constraints on the instantiation of the Adaptable Component (e.g., constraints on the legal combination of parameter values).

The interface part of Adaptable Components is different for software and documentation. The content of the software interface is specific to the design method(s) used to create the members of the family. The following types of information are examples: definitions of interface programs (names, parameters, parameter types, returned values), definitions of exported types, descriptions of the effects of interface programs, assumptions about the environment in which the software is to be used.

The interface for a documentation component does not require the same type of detailed information. It consists of a brief statement of the content of the component. Example DE.2.2.4.2-1 illustrates a fragment of a Component Design for the TLC domain. This design corresponds to one of the adaptable components identified in the Product Architecture shown in Example DE.2.2.4.1-1. The Adaptation Specification defines the adaptation parameters for this adaptable component. The Interface Specification is parameterized, where appropriate, in terms of these adaptation parameters.

Component Design -- Determine Schedule

This module determines the traffic light sequence schedule based on mode transition rules defined in the requirements document.

Adaptation Specification

Instantiation Parameters

schedule	list of composite of:
	start : (0:00 .. 23:59)
	end: (0:00 .. 23:59)
	name: identifier
	....

Instantiation Constraints

— There must be at least one schedule transition in the schedule list.

Interface Specification

Concrete Operations

determine_schedule_transition	Determines the traffic light sequence schedule that the TLC system should follow next and returns the appropriate indicator.
current_schedule	Returns the current schedule the TLC system following

....

Example DE.2.2.4.2-1. Fragment of TLC Component Design

Verification Criteria

- The Component Design satisfies the verification criteria established by the specific design method(s) used for its creation.
- The Component Design satisfies all structures of the Product Architecture.
- The value for each parameter in an Adaptation Specification either is derivable from the Decision Model or has a fixed, default value for all instances of the component family.

3. PROCESS DESCRIPTION

The Component Design Activity consists of two steps shown in Figure DE.2.2.4.2-1.

3.1 Procedure

Follow these steps for the Component Design Activity. Domain engineers perform these steps for each Adaptable Component defined in the internal organization of the Product Architecture.

Step: Define Component Adaptation Specification

Action Identify the variations that parameterize the Adaptable Component, and record constraints on legal combinations.

Input

- Product Architecture

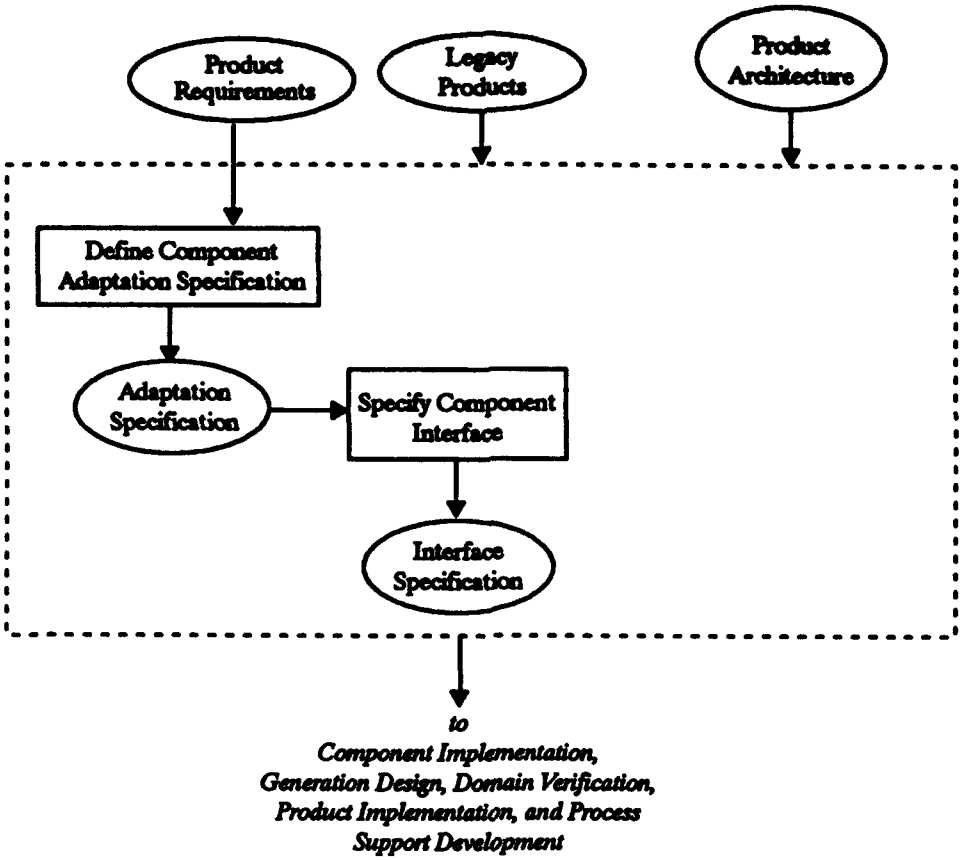


Figure DE.2.2.4.2-1. Component Design Process

- Product Requirements
- Legacy Products

Result

Component Design: Adaptation Specification

Heuristics

- Identify components in the Product Architecture that have the same form in all instances of the domain. These components have no associated variations (i.e., they are not adaptable). You must still provide an interface specification, but you may omit the Adaptation Specification.
- The adaptation specification of a component may be determined by identifying which roles the component must serve in the design, determining which adaptations are required so that it can fulfill these roles, and devising a set of parameters that can be used to indicate desired adaptations.
- Examine Legacy Products to determine variations. Concentrate on semantic, rather than syntactic, distinctions. However, unless you are willing to reengineer work products, you may need to define variations based on syntactic distinctions as well.

- Determine necessary component adaptations by analyzing the Product Requirements to see how the component must vary to satisfy relevant requirements. In practice, you should try to use both approaches.

- Decisions that parameterize components must derive from, but need not be, the decisions identified in the Decision Model. In general, there is a many-to-many relationship between Decision Model decisions and Component Design parameters. You may use whatever decisions most naturally specify variations among members of the family defined by an Adaptable Component.

Step: Specify Component Interface

Action	Specify the requisite properties for the implementation of each component.
Input	• Product Architecture • Component Design: Adaptation Specification
Result	Component Design: Interface Specification
Heuristics	The properties that you must specify depend on the type of component and the design method used. Parameterize each component interface with the decisions from the component's adaptation specification so that it describes all instances of the component.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Product Requirements are incomplete, ambiguous, or inconsistent.
Source	Product Requirements Activity
Response	Describe the inadequacies in the Product Requirements. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.
Contingency	The Domain Plan cannot be satisfied with available technical capabilities.
Source	Domain Management Activity
Response	Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Component Design that satisfies the Domain Plan as closely as possible.

Contingency	The practices and procedures specified in the Domain Plan are either ineffective or inefficient.
Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.
Contingency	The Product Architecture is incomplete, ambiguous, or inconsistent.
Source	Product Architecture Activity
Response	Describe the inadequacies in the Product Architecture. Proceed with Component Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.

4.2 Feedback From Product Consumers

Contingency	Suggestions are made for Component Design changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Component Design was completed.
Source	• Product Implementation Activity • Process Support Development Activity
Response	• Revise the Component Design. • Refer to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Component Design does not satisfy the Product Requirements.
Source	Domain Verification Activity
Response	Modify the Component Design to be consistent with the Product Requirements.
Contingency	The Component Design is incomplete, ambiguous, or inconsistent.
Source	• Component Implementation Activity • Generation Design Activity
Response	Refine the Component Design to correct inadequacies.

DE.2.2.4.3. GENERATION DESIGN ACTIVITY

1. GETTING STARTED

The Generation Design Activity is an activity of the Product Design Activity for creating a Generation Design. A Generation Design is a specification of production procedures that an application engineer uses to produce deliverable application engineering work products. A Generation Design defines a transformation (or mapping) from an Application Model that describes a product to the equivalent application engineering work products. For each application engineering work product, a Generation Design specifies how to select and adapt Adaptable Components according to decisions in an Application Model and to compose them according to the internal organization of that work product in the Product Architecture. The Generation Design Activity is performed for each work product that comprises the product family specified by the Product Requirements.

1.1 Objectives

The objective of the Generation Design Activity is to produce a specification for the production procedures that can be used to produce application engineering work products for a member of a product family through reuse of Adaptable Components. The specification establishes a correspondence between an Application Model and equivalent domain engineering work products that implement the intent of the model correctly.

1.2 Required Information

The Generation Design Activity requires the following information:

- Decision Model
- Product Architecture
- Component Designs

1.3 Required Knowledge and Experience

The Generation Design Activity requires domain and software knowledge and experience in:

- How work products of systems in the domain are designed
- The principles and use of the design method used for the Product Architecture
- The concepts and practice of abstraction-based reuse (Parnas 1976; Campbell 1989)

2. PRODUCT DESCRIPTION

Name

Generation Design

Purpose

A Generation Design is a specification for a production procedure for creating deliverable application engineering work products.

Content

A Generation Design relates the decisions from the Decision Model to the elements of a work product's internal organization defined in the Product Architecture. A Generation Design consists of three mappings:

- ***Architecture Mapping.*** The Architecture Mapping is a description of the relation between decisions in the product family's Decision Model and the decisions of the corresponding adaptable Product Architecture. This mapping describes how values for the Product Architecture's decisions are determined from values of decisions in the Decision Model. As a result, the Architecture Mapping defines the internal organization of a work product that describes a member of the product family based on decisions in the Decision Model (i.e., from an Application Model).

- ***Component Mapping.*** A Component Mapping is a description of the relation of each element of the organizational structure to an Adaptable Component that implements that element. This mapping defines how each component of a work product is to be produced.

- ***Decision Mapping.*** A Decision Mapping is a description of the relation between decisions in the product family's Decision Model and the instantiation parameters in the adaptation specification of a Component Design for each work product component. This relation describes how values for the instantiation parameters are determined from values of decisions in the product family's Decision Model.

Form and Structure

There is a Generation Design for each supported work product. The Architecture Mapping is represented as a statement for each instantiation parameter of the work product's Product Architecture. The statement contains a pairing between an instantiation parameter and an expression. The expression to determine the value to assign an instantiation parameter is described in terms of decisions in the product family's Decision Model. The expression may involve iteration over a group of decisions or conditional testing of one or more decisions.

The Decision Mapping representation is similar to the Architecture Mapping, except that the instantiation parameters come from the adaptation specification of the Component Design for the work product.

The Component Mapping is represented as a "use" statement. If the expression bracketing the use statement is True, then the use statement describes which Adaptable Component contains the needed implementation.

The expression is usually described in terms of decisions in the product family's Decision Model. However, if the Adaptable Component is always used, then an expression of True is sufficient to describe this situation.

Examples DE.2.2.4.3-1 and DE.2.2.4.3-2 illustrate fragments of a Generation Design for the TLC domain. These depict one way of representing the expressions discussed for Architecture, Component, and Decision Mapping. The decisions used the metaprogramming notation come directly from the Decision Model shown in Example DE.2.2.1-1. The parameters on the lefthand side of the "=" statements in the Architecture and Decision Mapping come from Examples DE.2.2.4.1-1, DE.2.2.4.1-3, and DE.2.2.4.2-1, respectively.

```
Generation Design – Static Software Architecture for the TLC product family

    Architecture Mapping
        crosswalk =  <if there is a street S that has a Crosswalk_Button = CB then> yes
                     <else> no
                     <endif>
        ....

    Component Mapping
        Determine Schedule
            use adaptable component Determine_Schedule_Transition
        ....

    Decision Mapping
        Determine_Schedule_Transition
            schedule =  ( <forall S in Fixed_Schedule.Schedule>
                            (  start = <S.Start_Time>,
                               end = <S.Stop_Time>,
                               name = <S.Name>,
                               .... )
                          <endfor>
                        )
        ....
```

Example DE.2.2.4.3-1. Fragment of TLC Generation Design

Verification Criteria

- The Generation Design specifies mappings that will produce application work products which exhibit the internal organization specified in the Product Architecture.
- The Generation Design specifies mappings that produce application work products which satisfy the Product Requirements (i.e., the mappings are consistent with Product Requirements variation).
- All variabilities allowed by decisions are properly represented as product variations.
- The effects of variabilities among work products are mutually consistent (i.e., all mappings are consistent).

```
Generation Design – Interface Requirements Specification work product family

    Architecture Mapping
        crosswalk =        <if there is a street S that has a Crosswalk_Button = CB then> yes
                           <else> no
                           <endif>
        light_schedule =   <if the Traffic_Light_Controller.Schedule is fixed then> fixed
                           <else> dynamic
                           <endif>
        ....

    Component Mapping
        ....

    Decision Mapping
        ....
```

Example DE.2.2.4.3-2. Fragment of TLC Generation Design

3. PROCESS DESCRIPTION

The Generation Design Activity consists of two steps shown in Figure DE.2.2.4.3-1.

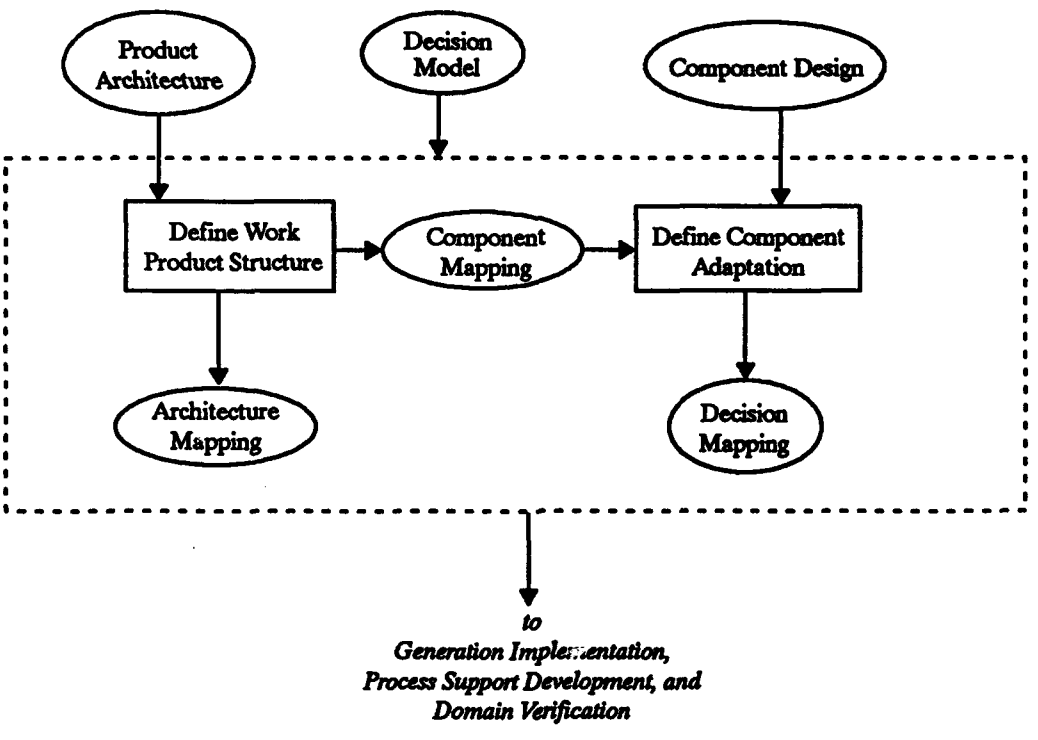


Figure DE.2.2.4.3-1. Generation Design Process

3.1 PROCEDURE

Step: Define Work Product Structure

Action Define how decisions in the Decision Model affect the structure of the work product.

Input
- Decision Model
- Product Architecture

Result Generation Design: Architecture and Component Mappings

Heuristics
- It is sufficient to define the work product structure as a mapping from the Decision Model to the internal organization of the Product Architecture for the work product. The internal organization defines the components that are required to implement the work product. This mapping determines which elements of the Product Architecture are implemented for a particular Application Model.
- The Product Architecture determines (conditionally and iteratively) how components of each work product are to be derived from Adaptable Components (i.e., the component mapping is provided implicitly by the Product Architecture). The Generation Design should not modify that mapping.
- Represent this mapping in metaprogramming notation associated with components in the Product Architecture. The mapping is defined in terms of decisions in the Decision Model and determines whether (one or more of) the associated component(s) should be included in the product created for a given Application Model. This mapping is formed by analyzing the Product Architecture and noting conditions that must be true if a particular component is to be included. If a component is always included in the product, metaprogramming notation is not required.
- Several Adaptable Components might be used to implement a single Product Architecture component, depending on decisions in the Application Model. In this case, use a conditional in the Component Mapping to qualify the association between the Product Architecture and Adaptable Components, thus indicating when a particular Adaptable Component is used.

Step: Define Component Adaptation

Action Define a mapping from the decisions in the Decision Model to adaptations of the Adaptable Components referenced by the Component Mapping.

Input
- Decision Model
- Component Design

- Generation Design: Component Mapping

Result Generation Design: Decision Mapping

Heuristics When a particular Component Design is to be used to implement a particular component in the Product Architecture, the variability of the Adaptable Component (i.e., its parameterization) must be realized in terms of decisions from the Decision Model. Define the value of each parameter (by name) as a derivation from Decision Model decisions.

3.2 Risk Management

Risk The Generation Design will not produce correctly-structured work products.

Implication Application Production will not produce acceptable application engineering work products.

Mitigation Derive work product structures from the Generation Design for Application Models of familiar systems and review the result with experienced engineers to determine whether the result is acceptable.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Decision Model is incomplete, ambiguous, or inconsistent.

Source Decision Model Activity

Response Describe the inadequacies in the Decision Model. Proceed with Product Architecture and document any assumptions made regarding the inadequate portions of the Decision Model.

Contingency The Product Requirements are incomplete, ambiguous, or inconsistent.

Source Product Requirements Activity

Response Describe the inadequacies in the Product Requirements. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Requirements.

Contingency The Domain Plan cannot be satisfied with available technical capabilities.

Source Domain Management Activity

Response Propose (alternative) revisions to the Domain Plan that better match available capabilities. Complete a Generation Design that satisfies the Domain Plan as closely as possible.

Contingency The practices and procedures specified in the Domain Plan are either ineffective or inefficient.

Source	Domain Management Activity
Response	Describe the ways in which the practices and procedures are either ineffective or inefficient. Propose revisions to the practices and procedures to make them more effective.
Contingency	The Product Architecture is incomplete, ambiguous, or inconsistent.
Source	Product Architecture Activity
Response	Describe the inadequacies in the Product Architecture. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Product Architecture.
Contingency	The Component Design is incomplete, ambiguous, or inconsistent.
Source	Component Design Activity
Response	Describe the inadequacies in the Component Design. Proceed with Generation Design, and document any assumptions made regarding the inadequate portions of the Component Design.

4.2 Feedback From Product Consumers

Contingency	Suggestions are made for Generation Design changes to exploit unforeseen opportunities. For example, a situation where substantial software is made available for use in the Domain Implementation that was not available when the Generation Design was completed.
Source	Generation Implementation Activity
Response	• Revise the Generation Design. • Refer to Domain Management for future planning. • Reject the changes due to conflicts with the Domain Definition.
Contingency	The Generation Design does not satisfy the Product Requirements.
Source	Domain Verification Activity
Response	Modify the Generation Design to be consistent with the Product Requirements.
Contingency	The Generation Design is incomplete, ambiguous, or inconsistent.
Source	Generation Implementation Activity
Response	Refine the Generation Design to correct inadequacies.

This page intentionally left blank.

DE.2.3. DOMAIN VERIFICATION ACTIVITY

1. GETTING STARTED

Domain Verification is an activity of Domain Engineering for ensuring the correctness, consistency, and completeness of domain engineering work products. Both formal and informal techniques may be applied to the domain engineering work products to verify these properties. Domain Verification is an independent verification activity performed separately from, and in addition to, the verification performed as part of each Domain Engineering Activity.

The Domain Verification Activity is motivated by the same concern that motivates IV&V in a conventional software development process; namely, that engineers involved in developing a work product cannot objectively judge the quality of that work product. Independent validation of the domain engineering product, from the perspective of client projects, is conducted in the Domain Validation step of the Project Support Activity.

Domain Verification establishes the correctness, consistency, and completeness of domain engineering work products. These terms have a precise meaning in the context of this activity. The concept of correctness is that of relative correctness. Similarly, the concept of completeness is that of relative completeness. A work product is said to be correct (complete) with respect to some criteria or to a more abstract representation of the entity the work product describes. For example, the Product Implementation can be said to be correct (complete) with respect to the Product Requirements. Consistency, on the other hand, is a term that applies to a collection of related work products (at the same level of abstraction) that form a whole. Two products are consistent when they exhibit the intended interrelationships. For example, the Product Architecture, Component Design, and Generation Design work products are strongly interrelated, and, therefore, mutual consistency is an important property for these work products.

1.1 Objectives

The objective of Domain Verification is to independently evaluate the quality of domain engineering work products.

1.2 Required Information

Domain Verification requires the following information:

- Domain Definition
- Domain Specification

- Domain Implementation
- Domain Plan: Practices and Procedures

1.3 Required Knowledge and Experience

The Domain Verification Activity requires domain and software knowledge and experience in:

- Appropriate software verification techniques
- How to systematically plan and perform software verification

2. PRODUCT DESCRIPTION

There are no Synthesis work products produced during Domain Verification.

3. PROCESS DESCRIPTION

The Domain Verification Activity consists of three steps shown in Figure DE.2.3-1.

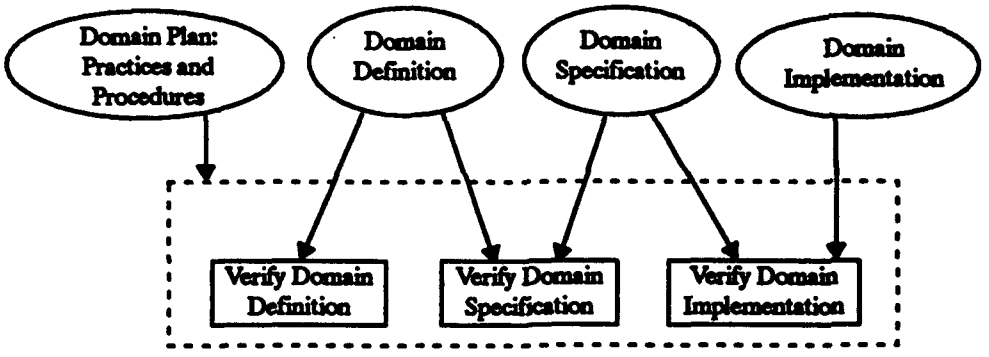


Figure DE.2.3-1. Domain Verification Process

3.1 Procedure

Step: Verify Domain Definition

Action	Verify the correctness, consistency, and completeness of the Domain Definition.
Input	• Domain Definition • Domain Plan: Practices and Procedures
Result	None
Heuristics	• Verify that the parts of the Domain Definition are correct and complete with respect to the guidance provided in their respective activity descriptions.

- Verify that the parts of the Domain Definition are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.
- Verify that the Domain Synopsis, Domain Glossary, and Domain Assumptions are mutually consistent.
- Use the verification criteria, established for the Domain Definition in its activity description, as guidance in verifying the Domain Definition.
- Use static analysis techniques (e.g., formal inspections, reviews, analysis tools) to verify the Domain Definition. These techniques are appropriate because the Domain Definition is typically represented in document form.

Step: Verify Domain Specification

Action Verify the correctness, consistency, and completeness of the Domain Specification.

Input
- Domain Definition
- Domain Specification
- Domain Plan: Practices and Procedures

Result None

Heuristics
- Verify that the parts of the Domain Specification are correct and complete with respect to the guidance provided in their respective activity descriptions.
- Verify that the parts of the Domain Specification are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.
- Verify that the Process Requirements, Product Requirements, and Product Design are consistent with the Decision Model. This means that these work products only reference decisions in the Decision Model and, conversely, all applicable decisions in the Decision Model are reflected in the work products.
- Verify that the Product Architecture, Component Design, and Generation Design are mutually consistent.
- Verify that the Product Design is correct and complete with respect to the Product Requirements.
- Verify that the Process Requirements is correct and complete with respect to the assumptions about the Application Engineering process in the Domain Definition. The Application Engineering process is normally not

explicitly described in the Domain Definition, but the Domain Definition will typically constrain what is an acceptable Application Engineering process.

- Verify that the Product Requirements and Product Design are correct and complete with respect to the representation of the Product Family in the Domain Synopsis and Domain Assumptions parts of the Domain Definition.
- Use the verification criteria, established for the Domain Specification work products in their respective activity descriptions, as guidance in verifying the Domain Specification.
- Use static analysis techniques (e.g., formal inspections, reviews, analysis tools) to verify the Domain Specification. These techniques are appropriate because the Domain Specification is typically represented in document form. If parts of the Domain Specification are represented in an executable form, the use of dynamic analysis techniques may be appropriate.

Step: Verify Domain Implementation

Action Verify the correctness, consistency, and completeness of the Domain Implementation.

Input
- Domain Specification
- Domain Implementation
- Domain Plan: Practices and Procedures

Result None

Heuristics
- Establish the criteria that you expect the Domain Implementation to meet before you try to verify it. Identify analysis that you can perform to ensure that the Domain Implementation is correct with respect to the Domain Specification. Your plan should minimally establish verification objectives and describe a strategy for meeting those objectives.
- Verify that the parts of the Domain Implementation are correct and complete with respect to the guidance provided in their respective activity descriptions.
- Verify that the parts of the Domain Implementation are correct with respect to any specific quality attributes required of them in the Practices and Procedures portion of the Domain Plan.
- Verify that the Process Support and Product Implementation are mutually consistent.
- Verify that the Component Implementation and the Generation Implementation are mutually consistent.

- Verify that the Process Support is correct with respect to the Process Requirements.

- Verify that documents and automation that make up the Process Support are engineered in a way that adequately addresses human factors concerns. For example, you should establish that the Application Engineering Environment portion of Process Support has the qualities of usability, adequate performance, and tolerance of user errors.

- Verify that the analyses that the Process Support allows to be performed on Application Models produce correct results.

- Verify that the Product Implementation is correct and complete with respect to the Domain Specification. The requirements for the Product Implementation are represented in the Product Requirements portion of the Domain Specification. The internal organization for the Product Implementation is represented in the Product Design portion of the Domain Specification.

- Verify that work products produced using the Process Support have expected properties. Do this by resolving the decisions of the product family's Decision Model, producing the work products corresponding to that model, and then verifying that the work products have the expected properties. Specifically:

  - Verify that the work products produced by the Process Support are correct and complete with respect to the Product Requirements and Product Design of the product family (appropriately instantiated with the decisions from the Decision Model).

  - Verify the usability and correctness of the Delivery Support. This should be established through direct inspection and by using the delivery support to install/deliver the Application Product.

  A good strategy for selecting work products to produce is to try to build all or part of Legacy Products that are within the intended scope of the domain.

- Use the verification criteria, established for the Domain Implementation work products in their respective activity descriptions, as guidance in verifying the Domain Implementation.

- Use conventional verification techniques that are appropriate to the task of verifying the Domain Implementation. Static analysis techniques (e.g., inspections) are appropriate for static representations of the Domain Implementation (e.g., Application Engineering User's Guide). Dynamic analysis techniques (e.g., testing) are appropriate for dynamic aspects of the Domain Implementation (e.g., automated support for specification, analysis, and product generation).

3.2 Risk Management

Risk	The criteria used to evaluate the domain engineering work products will be unduly influenced by the final content and form of the work products themselves.
Implication	The effectiveness of the verification effort will be reduced.
Mitigation	Define acceptable levels of correctness, completeness, and consistency for each domain engineering work product prior to examining it.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Definition is incorrect, inconsistent, or incomplete.
Source	Domain Definition Activity
Response	Precisely communicate how the Domain Definition is incorrect, inconsistent, or incomplete.
Contingency	The Domain Specification is incorrect, inconsistent, or incomplete.
Source	Domain Specification Activity
Response	Precisely communicate how the Domain Specification is incorrect, inconsistent, or incomplete.
Contingency	The Domain Implementation is incorrect, inconsistent, incomplete.
Source	Domain Implementation Activity
Response	Precisely communicate how the Domain Implementation is incorrect, inconsistent, or incomplete.

4.2 Feedback From Product Consumers

None

DE.3. DOMAIN IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Domain Implementation is an activity of Domain Engineering for implementing product and process support for application engineering projects in a business area. The Domain Implementation must satisfy the Domain Specification created by Domain Analysis. Product support consists of a set of production procedures and associated Adaptable Components that can be used to create standardized work products for members of the product family. Process support consists of procedures, documentation, and, optionally, automation that define a standard Application Engineering process.

1.1 Objectives

The objectives of the Domain Implementation Activity are to:

- Create a set of Adaptable Components and Generation Procedures as specified in the Product Design
- Create a standardized Application Engineering process as specified in the Process Requirements

1.2 Required Information

The Domain Implementation Activity requires the following information:

- Domain Specification
- Legacy Products

1.3 Required Knowledge and Experience

The Domain Implementation Activity requires domain and software knowledge and experience in:

- Technologies for creating, adapting, and composing Adaptable Components into systems and the verification of such Adaptable Components
- Documenting and providing automated support for Application Engineering processes
- How systems in the domain are built and sufficient expertise to create and document the software for these systems

2. PRODUCT DESCRIPTION

Name Domain Implementation

Purpose A Domain Implementation contains sets of Adaptable Components and associated production procedures that you can use to create standardized work products for members of a product family. The Domain Implementation also consists of the procedures, documentation, and, optionally, automation that define a standard Application Engineering process.

Content A Domain Implementation consists of two components:

- ***Product Implementation.*** A Product Implementation contains an adaptable implementation of a product family (see Section DE.3.1).
- ***Process Support.*** An application engineering infrastructure that supports the practice of Application Engineering by defining the procedures and standards by which application engineers develop applications (see Section DE.3.2).

Verification Criteria The Domain Implementation supports the product family identified in the Domain Specification.

3. PROCESS DESCRIPTION

The Domain Implementation Activity consists of the two steps shown in Figure DE.3-1.

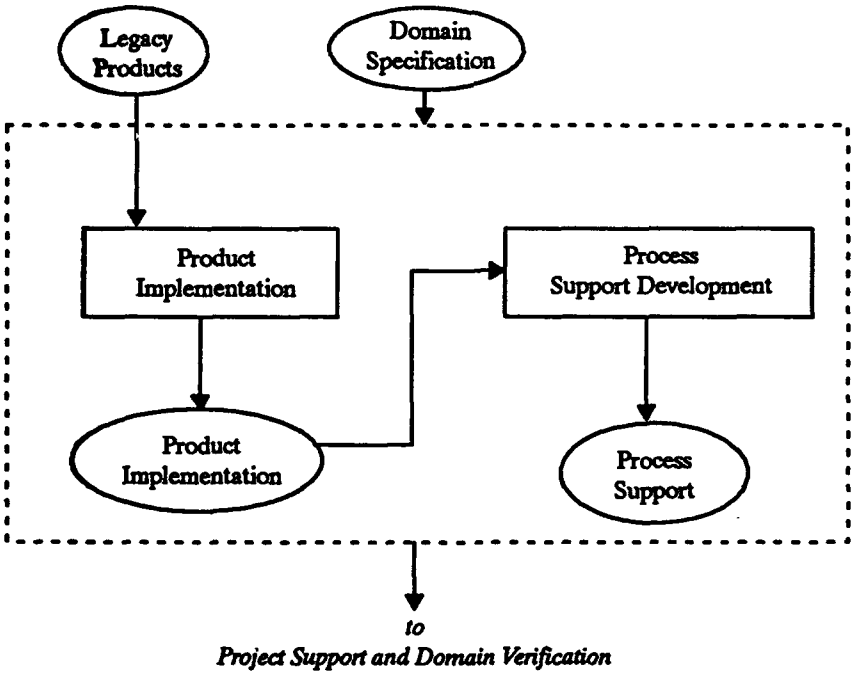


Figure DE.3-1. Domain Implementation Process

3.1 Procedure

Follow these steps for the Domain Implementation Activity.

Step: Product Implementation Activity

Action	Implement a product family.
Input	• Domain Specification • Legacy Products
Result	Product Implementation
Heuristics	• Derive the Product Implementation of a product family from appropriate Legacy Products. • Describe a mechanical procedure by which application engineers can select, adapt, and compose a deliverable application work product.

Step: Process Support Development Activity

Action	Create an application engineering infrastructure to support the standardized process by which application engineers develop applications.
Input	• Product Implementation • Domain Specification: Process Requirements
Result	Process Support
Heuristics	• Document the procedures and standards that application engineers follow to develop applications. • Optionally provide automated mechanisms which support the effective and correct performance of the Application Engineering process.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency	The Domain Specification is incomplete, ambiguous, or inconsistent.
Source	Domain Analysis Activity
Response	Describe how the Domain Specification is inadequate and suggest how it may be modified. Proceed with Domain Implementation as far as possible with the current Domain Specification.
Contingency	Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made

available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source Domain Analysis Activity

Response Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback From Product Consumers

Contingency The Domain Implementation is incorrect, inconsistent, or incomplete.

Source Domain Verification Activity

Response Request clarification of the intent of the Domain Specification, if necessary. Modify the Domain Implementation to satisfy the Domain Specification.

Contingency The support for the Application Engineering process is inefficient.

Source Project Support Activity

Response Revise the Domain Implementation based on Application Engineering experience.

DE.3.1. PRODUCT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

The Product Implementation Activity is the activity of the Domain Implementation Activity for creating a Product Implementation. A Product Implementation is an implementation a product family. A conventional implementation is an application and associated work products that solve a specific problem. Similarly, a Product Implementation is an implementation that is adaptable to decisions supported by the product family's Decision Model in order to solve any of a family of problems. A Product Implementation consists of Adaptable Components (e.g., code, documentation, and support for verification/validation) and procedures, as needed, for selecting, adapting, and composing these components. The Adaptable Components and procedures are used to create deliverable application engineering work products in accordance with an Application Model that describes the product.

1.1 Objectives

The objective of the Product Implementation Activity is to implement the Product Design. This implementation is used by application engineers to generate required work products for systems in the domain.

1.2 Required Information

The Product Implementation Activity requires the following information:

- Product Requirements
- Product Design
- Decision Model
- Legacy Products

1.3 Required Knowledge and Experience

The Product Implementation Activity requires domain and software knowledge and experience in:

- The design method used in specifying the Product Design
- Existing systems in the domain, including how they are designed, implemented, and verified, and what are their components and architectures

- Target language and platform capabilities
- The technologies for adapting and composing components into work products that make up an integrated product

2. PRODUCT DESCRIPTION

Name Product Implementation

Purpose A Product Implementation is an adaptable implementation of a product family. An application engineer must be able to derive members of a product family by adapting the Product Implementation mechanically based on the product family's decisions in an Application Model.

Content A Product Implementation consists of the following parts:

- ***Adaptable Components.*** An implementation of the product family's Component Design. Adaptable Components include software, documentation, and verification/validation components that are adapted based on the product family's Decision Model (see Section DE.3.1.1). Adaptable Components may be derived from Legacy Product. Collectively, these Adaptable Components form all of the necessary components for constructing, documenting, verifying/validating, and delivering a system.

- ***Generation Procedure.*** An implementation of a Generation Design for selecting, adapting, and composing Adaptable Components into deliverable work products that satisfy an Application Model (see Section DE.3.1.2).

  There is a Generation Procedure per product family. The Generation Procedure unifies a set of separate work product specific Generation Procedures. Each work-product-specific Generation Procedure corresponds to a particular set of Adaptable Components in a Product Implementation.

Verification Criteria The Product Implementation correctly constructs existing or envisioned systems from the domain.

3. PROCESS DESCRIPTION

The Product Implementation Activity consists of the two steps shown in Figure DE.3.1-1.

3.1 PROCEDURE

Follow these steps for the Product Implementation Activity.

Step: Component Implementation Activity

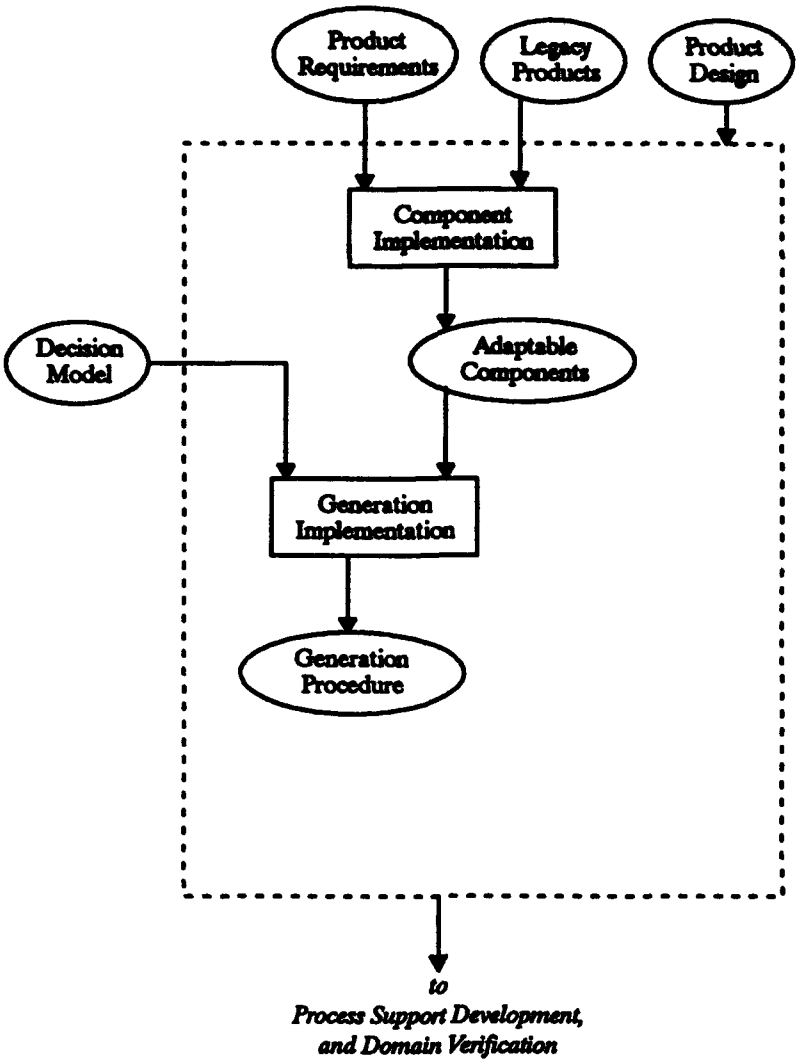


Figure DE.3.1-1. Product Implementation Process

Action	Create Adaptable Components for the product family.
Input	• Product Requirements • Product Design • Legacy Products
Result	Adaptable Components
Heuristics	• Derive Adaptable Components from Legacy Products.

- Ensure that the Adaptable Component satisfies and is consistent with relevant aspects of the Product Design and Product Requirements.

Step: Generation Implementation Activity

Action — Automate or document a mechanical procedure by which application engineers can derive deliverable application work products consistent with an Application Model.

Input
- Generation Design
- Decision Model
- Product Design: Product Architecture

Result — Generation Procedure

Heuristics — Ensure that the Generation Procedure satisfies and is consistent with relevant aspects of the Generation Design.

3.2 Risk Management

Risk — The Product Implementation will be inconsistent with Product Requirements.

Implication — Application work products will be generated that do not satisfy the Product Requirements.

Mitigation — When uncertainties arise, review the Domain Specification with domain analysts to clarify their intent. Review the Domain Implementation with other experienced engineers to identify omissions and inconsistencies. Derive test work products based on knowledge of existing or anticipated systems for review with experienced engineers.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency — The Domain Specification is incomplete, ambiguous, or inconsistent.

Source — Domain Specification Activity

Response — Describe how the Domain Specification is inadequate, and suggest how it may be modified. Proceed with Product Implementation as far as possible with the current Domain Specification.

Contingency — Unforeseen opportunities arise that cannot be exploited given the current Domain Specification, e.g., a situation where substantial software is made available for use in the Domain Implementation that was not available when the Domain Specification was completed.

Source	Domain Specification Activity
Response	Document the opportunities and the required changes to the Domain Specification.

4.2 Feedback From Product Consumers

Contingency	The Product Implementation does not satisfy the Domain Specification.
Source	Domain Verification Activity
Response	Request clarification of the intent of the Domain Specification if necessary. Modify the Product Implementation to satisfy the Domain Specification.

This page intentionally left blank.

DE.3.1.1. COMPONENT IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Component Implementation is an activity of the Product Implementation Activity for creating an Adaptable Component. A component is any work product fragment (e.g., software, documentation, or verification/validation artifact) produced during the Application Engineering process. An application engineering work product consists of a set of components. An Adaptable Component is a representation of a family of components that satisfies a Component Design (i.e., is adaptable to specified variations). The variability of an Adaptable Component enables application engineers to extract components to be used in creating work products for members of a product family.

1.1 Objectives

The objective of the Component Implementation Activity is to implement an Adaptable Component that satisfies a Component Design, consistent with relevant aspects of the Product Requirements and Product Architecture.

1.2 Required Information

The Component Implementation Activity requires the following information:

- Product Requirements
- Product Architecture
- Component Design
- Legacy Products

1.3 Required Knowledge and Experience

The Component Implementation Activity requires domain and software knowledge and experience in:

- Applicable standards and techniques for design, implementation, and verification of software components
- How to design, implement, and verify components of a work product family given a specification for the family
- The design and implementation of existing systems in the domain

2. PRODUCT DESCRIPTION

Name

Adaptable Component

Purpose

An Adaptable Component is a component (e.g., of software, documentation, verification/validation support) that is adaptable with respect to variations specified in the Component Design.

Content

An Adaptable Component is an implementation of a family of components. This family is defined by a Component Design, with support by portions of the Product Requirements and Product Architecture. The Component Design characterizes an Adaptable Component by specifying the permissible adaptations of the component, along with the desired characteristics of its implementation.

Form and Structure

An Adaptable Component is uniquely named and consists of two parts: an adaptability interface and a body.

A component family is characterized by a set of common capabilities and variations in those capabilities. The adaptability interface is a specification of a set of adaptation parameters that provide for the characterization and extraction of a particular instance of a component family.

The body is the sum of the potential implementations of all of the components in the family. The term "potential" is used because the parameters are sufficient to select any component family instance uniquely, but the particular implementation either may not be available or may be extracted from a representation of the family or relevant subfamily. This varies with the mechanism used for implementing adaptability in the Adaptable Component. Three common mechanisms for implementing an Adaptable Component are:

- ***Physical Separation.*** Represent members of the component set as physically distinct entities (e.g., in separate files on a computer).
- ***Target-Language Mechanisms.*** Use the language-specific facilities to represent different component set members. Ada generics, C++ templates, Interleaf effectivity control, and WordPerfect macro features are examples of target-language mechanisms for representing an Adaptable Component.
- ***Metaprogramming.*** Superimpose a language for handling variations on top of the language in which all members of the component set are expressed.

These may be used separately or in combination to implement a particular set of components.

The Booch Ada stack packages (Booch 1987) are an example of an Adaptable Component. There is a unifying concept of what a stack is and how it works. Different stack components are extracted based on decisions such as:

- ***Type.*** The type of data that can be put into the stack.

- ***Iteration.*** Whether an indexing mechanism should be available for moving forward and backward through the stack in addition to simply pushing elements onto and popping elements off of the top of the stack.

- ***Garbage Collection.*** Whether the stack should manage unused stack space dynamically for later use.

- ***Bounding.*** Whether the stack should be bounded in length.

The Booch packages use physical separation to implement 26 different types of stacks. This physical separation approach has the advantage of being simple. If a family has 10 instances, there are 10 implementations and each can be written and verified independently. Physical separation does not take advantage of similarities among the instances, however, nor does it make explicit how variabilities determine the content of each instance.

Ada provides generic packages as a standard facility that you can use to implement a code Adaptable Component whose instances differ only in the values of well-defined placeholders that can be substituted at compile time. The "type" variability of stack packages may be represented using a generic package. The placeholders are parameters defined in the adaptability interface of the Adaptable Component. This approach has the advantage of representing variabilities for the component family more compactly and uniformly; however, only a simple form of parameter substitution is supported.

A metaprogramming approach (Campbell 1989) uses a preprocessing mechanism to extract a concrete component from an Adaptable Component. This approach embeds target text fragments of a work product (e.g., code, documentation, verification/validation support) with a superimposed metaprogramming notation. The metaprogramming notation specifies how the target fragments are to be combined and adapted, based on the parameters in the adaptability interface of the Adaptable Component. Typical metaprogramming adaptations include:

- Substitution of parameter values

- Conditional inclusion of text fragments

- Repetition of text fragments

- Definition and in-line instantiation of parameterized fragments

This approach provides greater flexibility in representing a component family compactly but results in more complex descriptions. Since many implementations may be derived from a single description, domain engineers must both manually inspect that description and extract and verify representative instances. Example DE.3.1.1-1 illustrates a small fragment of a Component

Implementation for the TLC domain. This example contains a portion of the implementation for the Adaptable Component specified in Example DE.2.2.4.2-1.

```
Component Implementation – Determine_Schedule_Transition

    type mode is (                    -- an enumerated type describing the different traffic light schedules
        <forall s in schedule>
            <s.name>,
        <endfor>
            unknown
            );

    ----

    function determine_schedule_transition return mode is
        clock_time : c;
    begin
        c := get_clock_time;
    <forall s in schedule>
        if c >= <s.start> and c <= <s.end> then
            return <s.name>;
        elsif
    <endfor>
            ....
        endif;
    end determine_schedule_transition;
```

Example DE.3.1.1-1. Fragment of TLC Component Implementation

Verification Criteria

- The Adaptable Components implement their specifications as defined in the Product Architecture and Component Designs.
- The Adaptable Components produce the correct variations in concrete components.
- Behaviors or constraints imposed by Product Requirements or Product Architecture on the Adaptable Component are all supported.

3. PROCESS DESCRIPTION

The Component Implementation process consists of all activities necessary for implementing an Adaptable Component to its Component Design specifications, consistent with relevant parts of the Product Architecture and Product Requirements. This involves the design, implementation, and verification of the family of software, documentation, or test-support components. It may involve the reengineering of existing components of work products from previously built systems. The Component Implementation Activity consists of the three steps shown in Figure DE.3.1.1-1.

3.1 PROCEDURE

Follow these steps for the Component Implementation Activity.

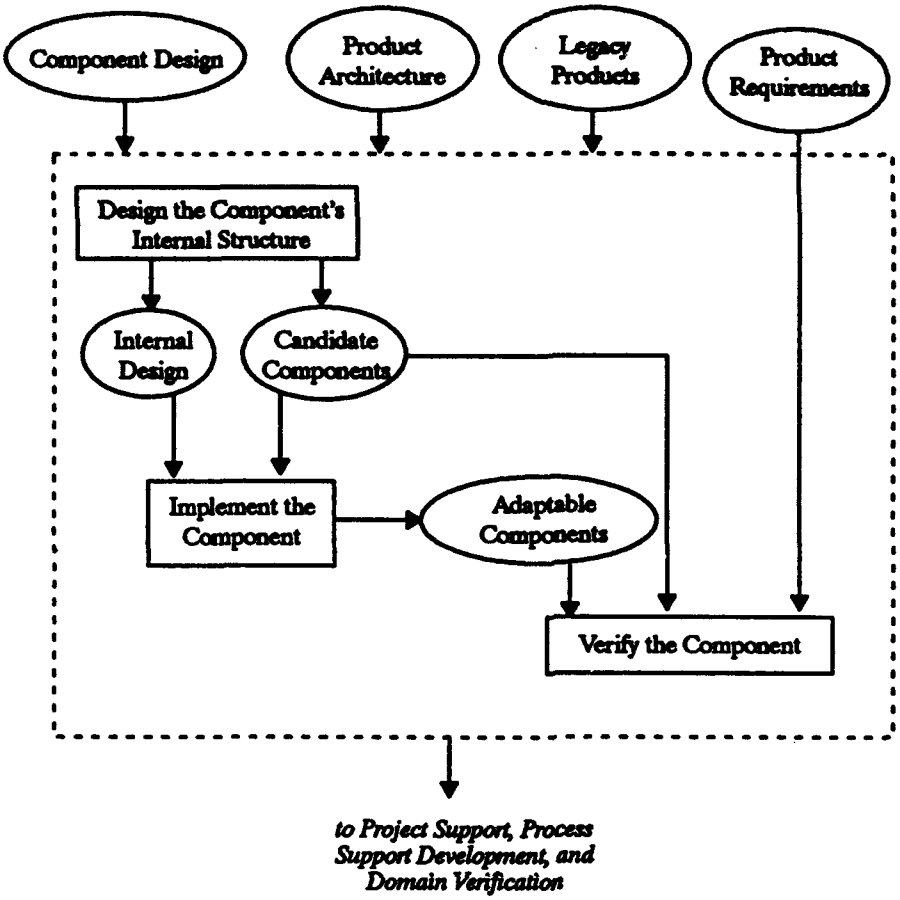


Figure DE.3.1.1-1. Component Implementation Process

Step: Design the Component's Internal Structure

Action Create an internal design of the structure and elements of the required Adaptable Component.

Input
- Component Design
- Product Architecture
- Legacy Products

Result
- Adaptable Component: Internal design
- Candidate Components

Heuristics
- The Adaptable Components internal design must satisfy its Component Design specification. The structures of the Product Architecture may impose additional requirements on the internal structure (e.g., for concurrency or level of performance) or define constraints on how other components are used in the implementation.

- Envision how to implement required members of the component's work product family. Create structures, according to the design method used, that characterize the required implementation. Parameterize each structure, if appropriate, with adaptability parameters that vary the structure, as required, for different members of the component family. Consider the required operations of the members.

- Determine whether a suitable component that approximates one of these desired instances is available from Legacy Products. Identify and evaluate the quality of each such component and designate it as a Candidate Component for further use. Determine which Adaptable Component variations are implicitly addressed by this selected Candidate Component.

- Consider whether other Adaptable Components in the Product Architecture can be used to implement all or part of this component family. Components that have complex functionality may be implementable as a composition of instances of other, more primitive components.

- Consider starting with the internal designs of identified Candidate Components. Portions of several candidate components may be used collectively to implement the Adaptable Component. These components may implement different variations that will be required for the family. Characterize which instance of the family corresponds to each component (i.e., by its parameter values). Consider whether each design is sufficiently well-engineered and representative of the family, or at the least of a subfamily, to provide substantial leverage in being refined to represent other variations and the family as a whole. Consider that they may represent different subfamilies that should be structured differently. See if their designs can be unified using variations. Be sure you can still derive each of the components with an acceptable structure.

- Use adaptation mechanisms (e.g., target-language mechanisms, metaprogramming) already present in the existing work products.

Step: Implement the Component

Action Elaborate the internal design to implement the Adaptable Component.

Input

- Adaptable Component: Internal design
- Component Design
- Product Architecture
- Candidate Components
- Legacy Products

Result Adaptable Component

Heuristics

- Fill in the internal structure with the details of the implementation. Keep in mind how the adaptabilities affect the content of the parts of the structure.
- If suitable Candidate Components were used in creating the internal design, then the implementations of those components can be useful as the starting point in implementing the Adaptable Component.
- If another Adaptable Components is to be used in implementing this Component, determine how the adaptability parameters of this Component can be mapped into the parameters of that Component so that its correct instance is derived for a particular instance of this family.
- Parts of the Adaptable Component might have to be engineered from scratch if all elements of the Adaptable Component's implementation cannot be obtained from existing Candidate Components or other Adaptable Components. These areas should be given much greater thought to ensure that you produce the correct content.
- Consider reengineering existing application engineering work products (e.g., Legacy Products) to increase their reusability. For example, you might want to replace arbitrary limits on data structure size with generic parameters. You should consider this if it will relax or remove constraints in your Decision Model. You should take into account any documentation or coding standards for the targeted project when you reengineer existing work products.

Step: Verify the Component

Action Verify that the Adaptable Component satisfies all relevant specifications.

Input

- Adaptable Component
- Component Design
- Product Architecture
- Product Requirements
- Candidate Components

Result Verified Adaptable Component

Heuristics

- Perform rigorous inspection of the Adaptable Component by other experienced engineers. The Component Design, as well as relevant parts of the Product Requirements and Product Architecture, should be verified as being satisfied.
- Derive representative instances of the Adaptable Component and test those instances in a conventional fashion to see if they operate correctly.

One part of this activity is the creation of test-case scenarios that can be used in regression testing of the Adaptable Component when it is modified in the future. These scenarios may be made adaptable to the same parameters as the Adaptable Component itself so that a scenario can be derived for a particular test instance.

- Rederive the Candidate Components that influenced the implementation of the Adaptable Component. The original and derived instances can then be compared to see if and how they differ and whether an equivalent result can be produced.

- Use of a Candidate Component may have been based on assurances that the component received with respect to certain desired properties such as correctness, reliability, certification, and trust (in the security sense). Note, however, that modification of the component can invalidate some of these assurances (i.e., certification and trust). It is important to verify that the desired properties are retained when the component is extracted from the resulting Adaptable Component.

3.2 Risk Management

Risk	Certain combinations of adaptability are not fully supported in the Adaptable Component.
Implication	Work products for some systems will not be derivable using the Component.
Mitigation	In verifying the Adaptable Component, use bounds-coverage techniques to identify a variety of adaptability combinations in deriving test instances.
Risk	The effort required to implement all specified adaptabilities for an Adaptable Component is not viable compared to that required to develop concrete components which support a single system development effort.
Implication	Only a subfamily of the Adaptable Component will be available for production of systems in the domain.
Mitigation	Implement the variations required for the current systems under development. The development of these variations may require less effort than developing all possible variations and can be refined as additional needs arise. The Adaptable Component can be evolved to a completed state over several development iterations of a system or systems.
Risk	Determining the value of existing components as a basis for the Adaptable Component will require too much effort (e.g., too many components to search through, too labor intensive to look through complicated components, too difficult to determine whether a component is correctly implemented).
Implication	Effort to evaluate and reengineer existing components exceeds the effort to create the Adaptable Component from scratch.

Mitigation If there are too many existing components to search through to find a good baseline component, limit the amount of effort dedicated to the search, and use the best approximation that results from the limited search.

If looking through complicated components is too labor-intensive, reduce the number of components that will be reviewed. If the component is overly complicated, relying on higher-level documentation (i.e., requirements, high-level design, or testing documentation) of the component as an indicator of its worth may be beneficial. Reviewing documentation on the existing component is likely to take less effort than reviewing code.

If determining the correctness of the component is difficult, then determining correctness from previous test documentation may be sufficient. Reliance on existing components may be greater if engineers are available who developed, or at least used, the existing components.

Risk Modifying the baseline component may invalidate assurances of quality that the component possessed (e.g., certification).

Implication Modifying a certified component will require that the resulting Adaptable Component must pass, once again, the tests required for assurance of given properties.

Mitigation

- Concentrate effort on areas of particular concern. If the given properties are less important for the component family as a whole, treat that particular member as a special case (i.e., a component subfamily in its own right). That is, if a component family contains several members, only one of which is certified, define two Adaptable Components, one whose component family contains the certified member and another which contains all the uncertified members.
- Try to retain the essential nature of the baseline component in the Adaptable Component so that proving assurance of given properties is not an expensive process.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Component Design is incomplete, ambiguous, or inconsistent.

Source Component Design Activity

Response Describe how the Component Design is inadequate, and suggest how it may be modified. Proceed with Component Implementation as far as possible with the current Component Design.

4.2 Feedback From Product Consumers

Contingency The Component Implementation does not satisfy the Component Design.

Source	Domain Verification Activity
Response	Request clarification of the intent of the Component Design, if necessary. Modify the Component Implementation to satisfy the Component Design.

DE.3.1.2. GENERATION IMPLEMENTATION ACTIVITY

1. GETTING STARTED

Generation Implementation is an activity of the Product Implementation Activity for creating a Generation Procedure. A Generation Procedure is a precise description of how to derive deliverable application engineering work products consistent with the decisions in an Application Model for a product family. A Generation Procedure may be automated or may take the form of a precise description that application engineers can mechanically follow to create work products.

1.1 Objectives

The objective of the Generation Implementation Activity is to create a Generation Procedure as specified by a Generation Design.

1.2 Required Information

The Generation Implementation Activity requires the following information:

- Generation Design
- Product Architecture
- Decision Model
- Component Designs

1.3 Required Knowledge and Experience

The Generation Implementation Activity requires knowledge and experience in:

- The notation used in specifying the Generation Design
- The technologies for adapting and composing components into work products

2. PRODUCT DESCRIPTION

Name Generation Procedure

Purpose This is a procedure that an application engineer uses to create deliverable application engineering work products for a member of a product family using Adaptable Components. This procedure is either implemented as a product generator or documented as a manual procedure.

Content The Generation Procedure is a procedural description for producing an application engineering work product that satisfies the mappings of a Generation Design. The Generation Procedure describes how to select appropriate Adaptable Components, how to apply decisions from an Application Model to adapt them, and how to compose them to create the work product in final form.

Form and Structure The form of the Generation Procedure depends on whether the procedure is automated or manual. The Generation Procedure is either implemented as an automated product generator or documented as a manual procedure to be followed by application engineers. If the manual form is chosen, then the Generation Procedure form is likely to resemble the form of a Generation Design. If the Generation Procedure is implemented in the form of a product generator, however, it will be a conventional software program.

Verification Criteria

- The Generation Procedure for a product family can be used to produce application engineering work products that exhibit the internal organization specified in the Product Architecture.
- The Generation Procedure for a product family can be used to produce application engineering work products that satisfy the Product Requirements.
- If a manual form is used, the Generation Procedure for a product family clearly describes how deliverable application engineering work products are constructed from Adaptable Components based upon decisions contained in an Application Model.

3. PROCESS DESCRIPTION

The Generation Implementation Activity consists of the two steps shown in Figure DE.3.1.2-1.

3.1 PROCEDURE

Perform one or both of the following two steps. The appropriate action depends on what automation you determine to have a significant payoff in Application Engineering.

Step: Document the Generation Procedure

Action Document some or all of the Generation Design.

Input

- Generation Design
- Product Architecture
- Decision Model
- Component Designs

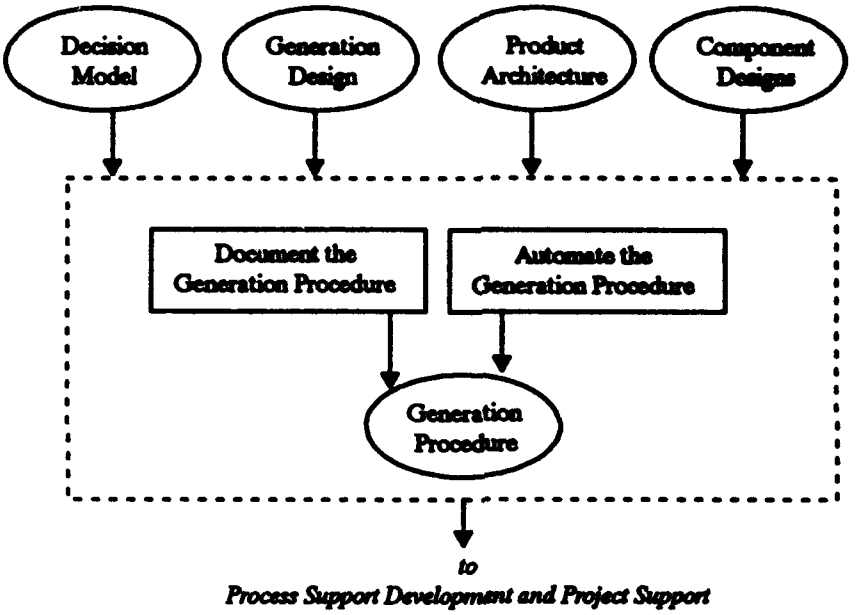


Figure DE.3.1.2-1. Generation Implementation Process

Result Generation Procedure

Heuristics

- The Generation Design is a precise description of the required Generation Procedure. Document the procedure in a form that is usable by application engineers.
- Include a description of how components are composed to form a work product consistent with its Product Architecture.
- Include a description of how to access the decisions in an Application Model for a product family. The Decision Model provides the organization of the decisions' conceptual schema.

Step: Automate the Generation Procedure

Action Develop automated tools that implement some or all of the Generation Procedure as defined in the Generation Design.

Input

- Generation Design
- Product Architecture
- Decision Model
- Component Designs

Result Generation Procedure

Heuristics

- If the Application Modeling Activity in Application Engineering is supported by automation, then the Generation Procedure must access the Application Model specification. If the activity is not automated, then there must be an automated mechanism for providing the decisions of the Application Model as input to the Generation Procedure. The Decision Model provides the organization of the decisions' conceptual schema.
- If a metaprogramming technology, such as described in the Component Implementation Activity (see Section DE.3.1.1), is used to implement the Adaptable Components, then the same metaprogramming technology is used to instantiate those components. Metaprogramming technology may also be useful in implementing portions of the Generation Procedure itself.
- Creating an automated Generation Procedure is a software development task. It requires the design of the required program, implementation to that design in a programming language, testing to verify that the resulting program implements the Generation Design correctly, and documentation so that the program can be correctly modified as the Generation Design changes.
- Tools such as the UNIX make facility may be useful in automating the procedure for composing adapted components into deliverable work products.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The Generation Design is incomplete, ambiguous, or inconsistent.

Source Generation Design Activity

Response Describe how the Generation Design is inadequate, and suggest how it may be modified. Proceed with Generation Implementation activity as far as possible with the current Generation Design.

Contingency Unforeseen opportunities arise that cannot be exploited given the current Generation Design, e.g., a situation where substantial software is made available for use in the Generation Implementation that was not available when the Generation Design was completed.

Source Generation Design Activity

Response	Document the opportunities and the required changes to the Generation Design.

4.2 Feedback From Product Consumers

Contingency	The Generation Procedure does not satisfy the Generation Design.
Source	Domain Verification Activity
Response	Request clarification of the intent of the Generation Design if necessary. Modify the Generation Procedure to satisfy the Generation Design.
Contingency	A manual Generation Procedure is difficult to use.
Source	Project Support Activity
Response	Investigate new forms for conveying the Generation Procedure to the application engineers.
Contingency	The Generation Procedure cannot be used in its current form in the Application Engineering process.
Source	Process Support Development Activity
Response	Revise the Generation Procedure (e.g., improve automation or upgrade documentation) so that it can be effectively used by application engineers.

This page intentionally left blank.

DE.3.2. PROCESS SUPPORT DEVELOPMENT ACTIVITY

1. GETTING STARTED

The Process Support Development Activity is the activity of Domain Implementation for creating the Process Support component of Domain Implementation. Process Support is the infrastructure that supports the practice of Application Engineering by defining the procedures and standards by which application engineers develop applications (i.e., the Application Engineering process). It optionally provides automated mechanisms which support the effective and correct performance of the Application Engineering process and associated use of the Product Implementation component of Domain Implementation.

1.1 Objectives

The objectives of the Process Support Development Activity are to:

- Document policies and procedures that institute a standard Application Engineering process and that guide its proper performance
- Determine the appropriate degree of automation that will support the Application Engineering process and construct the automated support

1.2 Required Information

The Process Support Development Activity requires the following information:

- Process Requirements
- Product Implementation

1.3 Required Knowledge and Experience

The Process Support Development Activity requires domain knowledge and experience in:

- Documenting, in a coherent and usable form, the use of conventions, policies, and procedures
- How to develop and communicate process standards in a concise and usable form
- Software production processes, methods, and practices

- Tools and technologies that can support an Application Engineering process (e.g., producing code/documents/tests, simulation/modeling)
- Developing user documentation and training courses for software development processes

This activity also requires the following additional expertise if you are planning to automate portions of the Application Engineering process:

- The principles and use of appropriate software development methods.
- Human-machine interface factors and related technology.
- Database technologies supportive of computer-aided software engineering.
- Host platform capabilities. The host platform is the hardware/software environment in which the automated portions of the Application Engineering process execute.
- Target platform capabilities. The target platform is the hardware/software environment in which the application produced by the Application Engineering process executes.

2. PRODUCT DESCRIPTION

Name Process Support

Purpose Process Support is a description and explanation of the policies and procedures by which application engineers produce a work product (the Application Engineering process) and automated support for efficient performance of the Application Engineering process.

Content Process Support consists of five parts:

- ***Application Engineering Process Standard.*** This establishes the policies and procedures that govern the Application Engineering process within the framework defined by the Process Requirements.
- ***Application Engineering User's Guide.*** A document that guides application engineers in how to perform the Application Engineering process to produce a product.
- ***Application Engineering Environment.*** The environment consists of the automated mechanisms that support the Application Engineering process.
- ***Application Engineering Environment Support Manual.*** This is a system administration manual describing how to install and maintain the Application Engineering Environment for a project. It provides appropriate information on any vendor-supplied software technologies contained in the environment.
- ***Application Engineering Training Courses.*** This material is used to train application engineers on how to perform the Application Engineering process and how to use Application Engineering Process Support.

Verification Criteria

- Members of the product family can be produced using the Application Engineering Environment by following the User's Guide.
- Process Support prescribes an effective and efficient Application Engineering process.
- Process Requirements are fully satisfied by Process Support.
- Process Support provides the ability to specify and generated the entire product family supported by Product Implementation.

2.1 Application Engineering Process Standard

Purpose

The Application Engineering Process Standard gives an overview of the standards, policies, and procedures that govern how application engineers should practice application engineering satisfactorily.

Content

The Application Engineering Process Standard documents the essentials of the Application Engineering process including what has to be done, why, and who completes the work. It also identifies and prescribes standards for form and content of application engineering work products.

Form and Structure

The form and structure of the Application Engineering Process Standard should adhere to your organization's current standards for documenting policies and procedures.

Verification Criteria

- The policies and procedures documented in the Application Engineering Process Standard are consistent with the Process Requirements.
- The policies and procedures documented in the Application Engineering Process Standard do not conflict with other applicable organizational or customer standards.

2.2 Application Engineering User's Guide

Purpose

The Application Engineering User's Guide provides a detailed description of how application engineers should perform to comply with the Application Engineering Process Standard. This guide expresses the decision-making process that application engineers follow for a domain.

Content

The Application Engineering User's Guide instructs application engineers on how to build applications that have particular characteristics and explains how to create, interpret, and evaluate Application Models properly. This guide also designates and explains the effective use of automated mechanisms that support the process.

Form and Structure

The User's Guide should conform to your organization's standards and guidelines for documentation.

Verification Criteria — The User's Guide describes how to build an application product.

2.3 Application Engineering Environment

Purpose — The Application Engineering Environment consists of all automated mechanisms, described in the User's Guide, that support creating and evaluating Application Models and generating and delivering equivalent application products. The Application Engineering Environment automates the mechanical portions of the process for increased consistency within a product and less opportunities for undetected error in producing a product.

Content — The Application Engineering Environment consists of both tools that are part of the host operating system and tools developed during this activity. Together, they support the Application Engineering process.

Form and Structure

- The tools adhere to the organization's standards and conventions for its software development environment.
- The view of the domain provided by the Application Engineering Environment must facilitate the tasks application engineers undertake when using it. The Application Engineering User's Guide describes these tasks in detail.

Verification Criteria

- All automated tools described in the User's Guide exist and behave as the User's Guide states. All Adaptable Components produced during Component Implementation Activities are accessible.
- Automated mechanisms contain no residual errors.

2.4 Application Engineering Environment Support Manual

Purpose — The Application Engineering Environment Support Manual describes how to install the automated mechanisms of the Application Engineering Environment.

Content — The instructions contained in this manual will be specific to your organization.

Form and Structure — The form and structure of the support manual should adhere to your organization's current standards for such work products.

Verification Criteria — Automated mechanisms can be installed on supported platforms without additional unspecified information or excessive effort.

2.5 Application Engineering Training Courses

Purpose — These courses are used to train application engineers on how to perform the Application Engineering process using the Process Support.

Content — The content is determined by how much expertise application engineers need in your organization to effectively perform Application Engineering.

Form and Structure

The form and structure of the training courses should adhere to your organization's current standards for such work products.

Verification Criteria

- The training course covers all topics application engineers need to know to effectively perform Application Engineering.
- The material in the training courses is consistent with the knowledge and experience in management and engineering of the anticipated attendees.

3. PROCESS DESCRIPTION

The Process Support Development Activity consists of all activities necessary to create appropriate Process Support consistent with the Process Requirements for the domain. In many respects, this activity involves work which is similar to that of a conventional software development project. You document the standards and procedures that establish how Application Engineering is to be practiced. You develop training courses and other support documentation that is used by the project support staff in helping projects use the domain product effectively. Furthermore, if you decide to automate some or all of the Application Engineering process (i.e., create an Application Engineering Environment), then you apply software development methods to accomplish that goal. The Process Support Development Activity consists of the five steps shown in Figure DE.3.2-1.

3.1 PROCEDURE

Follow these steps for the Process Support Development Activity. Perform these steps in the order listed but iterate through them until you are satisfied with the work product as a whole.

Step: Document the Application Engineering Process Standard

Action

Document the standard policies and procedures by which an application engineer develops applications in the domain.

Input

Process Requirements

Result

Application Engineering Process Standard

Heuristics

- The Process Requirements defines the essentials of the Application Engineering process. The Process Standard cannot conflict with the Process Requirements, but you will need to extend the Process Standard to provide a complete process description. The degree to which you plan to automate the process is a major factor in how you elaborate Process Requirements into a Process Standard.
- This document elaborates the essentials prescribed by the Process Requirements to establish the specific framework of the Application Engineering process. Your description of the process should define what has to be done, why it has to be done, and who completes the work. You should provide procedures that explain how to complete the process (the sequence of tasks or task steps that have to be performed), when the work is performed, and the criteria for measuring the quality of the work.

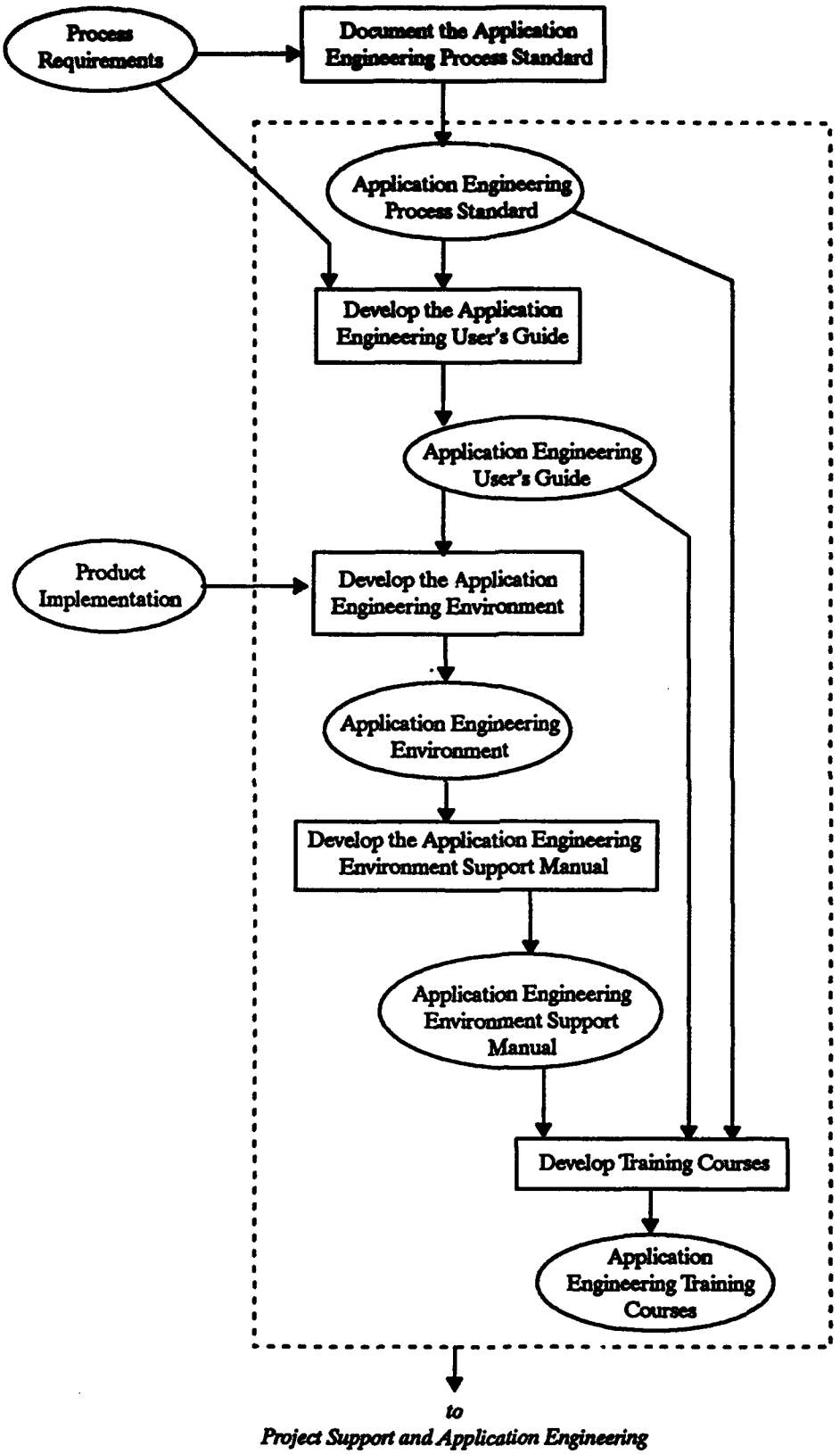


Figure DE.3.2-1. Process Support Development Process

- The purpose of standards is to promote a consistent approach in the development of application products. Your description of the standards should incorporate the standards used by your organization as well as cover detailed process requirements.

Step: Develop the Application Engineering User's Guide

Action Create a detailed guide for application engineers which instructs them on how to perform every aspect of Application Engineering including manual steps and effective use of any automated mechanisms.

Input

- Process Requirements
- Application Engineering Process Standard

Result Application Engineering User's Guide

Heuristics

- Describe all aspects of performing the Application Engineering process in the User's Guide. This description should be sufficient for experienced application engineers to understand and follow the process routinely (i.e., without assistance after initial training) and effectively. Within this description, explain effective use of the automated mechanisms provided by the Application Engineering Environment.
- Determine the degree to which the Application Engineering process is automated. Indicate additional aspects of the Application Engineering process to automate (beyond what Process Requirements and the Process Standard prescribe). Your guidance to the application engineer must reflect this decision. The choice of appropriate automation is governed by Process Requirements, economics, and human factors:
  - ***Process Requirements.*** The Process Requirements may dictate preferences on process and presentation automation mechanisms. For example, it may dictate display of certain information in a graphical form or the extent of automated support for Application Modeling. If Process Requirements imposes such preferences, you must either implement them or refer changes to the Process Requirements Activity.
  - ***Economics.*** You must determine whether reduced Application Engineering effort will be sufficient to justify the cost of automating. That cost may include the acquisition of software tools and hardware for application engineering projects, as well as the development resources (both time, people, and material) from your organization. Additionally, you must also consider the costs of maintaining and enhancing this automated support.
  - ***Human Factors.*** Manual procedures are usually labor-intensive and can result in many errors being introduced during any aspect

of Application Engineering, particularly in the Application Production phase. A project may lose much time trying to identify and correct errors, and then repeating the process from where the last error was introduced. Automation, whether specially-built or a commercial tool, can reduce the effort needed during labor-intensive activities. It can also help to reduce or eliminate errors in the process. Another benefit of using automation is to reduce training requirements for the application engineering staff. For example, a project may need more training to perform Application Modeling manually than if automated support were provided.

- Decide what expertise and experience application engineers are expected to have. Write the User's Guide with the perspective and assumptions that people with that expertise should have.
- Provide whatever domain knowledge application engineers will need to perform Application Engineering correctly and effectively. You should consider, as a minimum, paraphrasing the Product Requirements in a form usable by application engineers. This information will help application engineers understand the implications of decisions to be made in Application Modeling.
- Describe what the application engineer should do when confronted with a problem during Application Engineering. Include descriptions of common mistakes and known bugs (and corresponding work-arounds).
- Provide advice to the application engineer on how to build systems with particular characteristics (e.g., particular capabilities or performance). Explain the meaning and purpose of different Application Models that could be developed.

Step: Develop the Application Engineering Environment

Action — Design, implement, and verify the automated mechanisms needed to support the Application Engineering process.

Input
- Application Engineering User's Guide
- Product Implementation

Result — Application Engineering Environment

Heuristics
- The Application Engineering User's Guide specifies what aspects of the Application Engineering process are automated. Revise the User's Guide if this cannot be fully satisfied.
- Creating an Application Engineering Environment is a software development task. You must design an environment, implement that design in a programming language (or via equivalent commercially-available soft-

ware technology), and test it to verify that the resulting environment implements the Application Engineering User's Guide correctly.

- Reduce your up-front development costs by taking advantage of available technology to automate various activities within the infrastructure. For example, there are planning and scheduling tools for project management; object-oriented databases and user interface tools that can support specifying an Application Model; testing, prototyping, and environment simulation tools for validation; simulation and dynamic assessment tools for assessment; and metaprogramming and system generation tools for product generation. However, you must also consider what resources you will need to integrate these or other technologies into a coherent infrastructure.

- Consider, as a minimum, automating the specification task of the Application Modeling Activity and the Application Production Activity. These are the core of Application Engineering and provide the most direct benefits.

- Decide what code construction tools (e.g., compiler, linker, debugger) will be used by application engineers to construct Application Product Software. For code components, you must consider factors such as target hardware and operating system and, if different from the host environment, how the code will be tested (e.g., in a host-simulated target environment or directly in the target environment) and created in executable form for the target environment (e.g., cross-compilers).

- Consider how the Generation Procedure fits into this environment. From the perspective of Process Support, a Generation Procedure is a black-box that can apply the decisions resolved by an Application Model to Adaptable Components to create Application Product Software. If automated input of an Application Model is not supported, then additional effort is required from the application engineer to transform his Application Model into a form suitable for use by the Generation Procedure.

Step: Develop the Application Engineering Environment Support Manual

Action — Create a manual that defines the information, resources, and steps required to install the Application Engineering Environment for use by a project.

Input — Application Engineering Environment

Result — Application Engineering Environment Support Manual

Heuristics — Describe installation instructions for the automated mechanisms of the environment. These instructions are specific to your organization. However, consider, at a minimum:

- *Inventory.* This is a list of all materials (code, software utilities, documentation) to be provided to the Project Support Activity.

- ***Distribution Media.*** This describes the media and formats on which the automated mechanisms are provided to Project Support (e.g., tape, disk). It should also describe how to extract the contents from the medium.

- ***Build Procedures.*** These are instructions on how to build the automated environment from supplied source code.

- ***Software Resources.*** These describe software resources needed to build and execute the automated mechanisms (e.g., operating system, compiler, supporting utility programs, version numbers). These also identify the vendor and version of any commercially supported off-the-shelf software that supports portions of the environment.

- ***Hardware Resources.*** These describe the host platforms on which the environment can successfully execute (e.g., CPU, operating system).

- ***Limitations.*** These describe known discrepancies or unimplemented features in the delivered environment and any known workarounds.

Step: Develop Training Courses

Action Develop courses to train managers and application engineers in the effective use of the application engineering process and supporting mechanisms.

Input

- Application Engineering Process Standard
- Application Engineering User's Guide
- Application Engineering Environment Support Manual

Result Application Engineering Training Courses

Heuristics The content of these courses will be determined by how much expertise your organization believes application engineers need to practice Application Engineering effectively and correctly. You should consider, at a minimum, the following topics:

- How to build systems in your domain using the Application Engineering process.
- How to manage application engineering projects successfully.
- How to use the automated capabilities of the Application Engineering Environment.

3.2 Risk Management

Risk The Application Engineering process will be inconsistent with the Process Requirements.

Implication Application engineers may not be able to produce applications that satisfy customer requirements.

Mitigation Review the Process Standards with experienced managers and engineers to check viability. Review the design and use of automated mechanisms with experienced engineers to ensure that those mechanisms satisfy the Process Requirements and Process Support documentation.

Risk Automation will not address the major difficulties of engineering applications in a domain.

Implication The Application Engineering process will be too labor-intensive, error-prone, or difficult, increasing the cost required to build an application.

Mitigation

- Review other Application Engineering processes to see what areas were automated and the rationale for doing so.
- Review past Application Engineering process efforts from other, similar domains to see what areas of difficulties were encountered and how they were resolved.
- Measure and analyze the performance of the Application Engineering process to help identify deficiencies.

Risk The Application Engineering process will be hard to follow (i.e., vague, incomplete).

Implication Application engineers will have a difficult time developing applications. This may cause excessive use of the project support staff. It may also cause incorrect applications to be developed and increase the time required to deliver an acceptable product to the customer.

Mitigation Review the Process Support documentation with application engineers to see what areas of the process are incomplete, inconsistent, or ambiguous. Have them generate example work products, noting where they misinterpret or misuse the documentation.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 FEEDBACK TO INFORMATION SOURCES

Contingency The Process Requirements work product is incomplete, ambiguous, or inconsistent.

Source Process Requirements Activity

Response Describe specifically where the Process Requirements work product is inadequate and suggest improvements. Proceed with the implementation of Process Support as far as possible while the Process Requirements are being updated.

Contingency	The Generation Procedure cannot be used in its current form in the Application Engineering process.
Source	Generation Implementation Activity
Response	Describe how the Generation Procedure needs to be changed so that it will fit within the Application Engineering process.

4.2 Feedback From Product Consumers

Contingency	The Application Engineering process is difficult to use or is too labor-intensive.
Source	Project Support Activity
Response	Identify where the problems exist and discuss, with the application engineers, ways of reducing (or eliminating) these problems (e.g., through the use of automation).
Contingency	Suggestions are made to automate various aspects of the Application Engineering process.
Source	Project Support Activity
Response	Consider the economic and human factors to help you decide whether to pursue automating the suggested areas of the Application Engineering process.
Contingency	The Domain Specification is not satisfied.
Source	Domain Verification Activity
Response	• Correct errors as part of the next iteration of Process Support Development. • Refer capabilities that cannot be supported to Domain Analysis for revision.

DE.4. PROJECT SUPPORT ACTIVITY

1. GETTING STARTED

The Project Support Activity is an activity of Domain Engineering for validating Application Engineering Process Support and assisting projects in its use. Application Engineering Process Support is the application engineering name for the Domain Implementation. To ensure that the baselined Domain Implementation is usable and effective, Project Support independently validates it from the perspective of the product and process needs of the targeted application engineering projects. Project Support assists application engineers in effective use of the process and supporting materials, through delivery and installation, training, and consultation for the targeted project. Project Support trains application engineers in how to perform the prescribed Application Engineering process, using any accompanying automated support, and answers questions about the process, its documentation, and its automation. Based on issues, problems, and future needs identified by application engineers, Project Support coordinates feedback to the rest of Domain Engineering for improvements in the supported process or products of application engineering.

1.1 Objectives

The objectives of the Project Support Activity are to:

- Evaluate the effectiveness and quality of Domain Implementation for use by application engineering projects
- Provide customer support to application engineering projects in the understanding and use of Domain Implementation
- Provide a conduit by which the needs of application engineering projects can influence domain improvements and evolution

1.2 Required Information

The Project Support Activity requires the following information:

- Domain Definition
- Domain Implementation

1.3 Required Knowledge and Experience

The Project Support Activity requires domain and software knowledge and experience in:

- The methods, practices, and solutions of application development in the domain
- Installing and evaluating software products and their documentation
- Developing and teaching training courses
- Assisting engineers and managers in the use of process documentation and automation

2. PRODUCT DESCRIPTION

The Project Support Activity produces no work products. Instead, it is a service activity to application engineering projects within the domain.

3. PROCESS DESCRIPTION

The Project Support Activity consists of two steps shown Figure DE.4-1. The first, Domain Validation, is ongoing and must certify each baseline Domain Implementation as it becomes available. The second, Domain Delivery, is initiated at the beginning of each targeted application engineering project and continues until that project's termination.

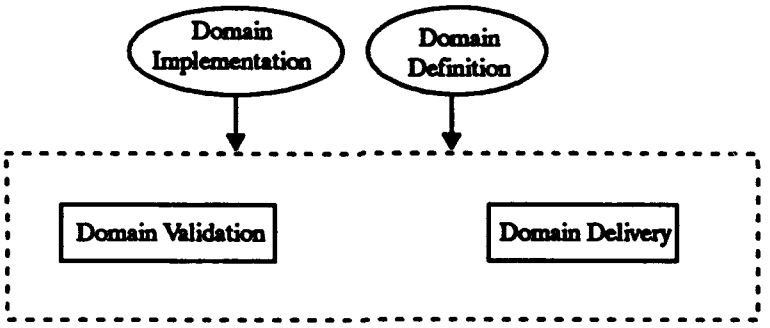


Figure DE.4-1. Project Support Process

3.1 PROCEDURE

Follow these steps for the Project Support Activity.

Step: Domain Validation Activity

Action	Certify that baselined, deliverable Domain Implementation will satisfy application engineering projects' needs, as specified in the Domain Definition (available in future releases).
Input	• Domain Definition • Domain Implementation
Result	None

Heuristics

- Review the Domain Plan and Domain Definition from the perspective of application engineering projects. Ensure that the product and process needs of projects are properly represented. Advise the rest of Domain Engineering on the realistic product and process needs of application engineering projects.
- Perform an independent evaluation of each baseline of the Domain Implementation as it becomes available. Evaluate whether it properly satisfies and balances the intended mix of general business objectives and specific application engineering project/customer needs.
- Perform independent validation, including extensive, scenario-based testing of the (integrated) Product Implementation and Application Engineering Environment portions of the Domain Implementation baseline. Evaluate the correctness and usability of the Application Engineering User's Guide and Application Engineering Environment Support Manual as they relate to use of the Product Implementation and Application Engineering Environment.
- Attempt to build typical products that reflect realistic project experience on existing systems in the domain. Identify capabilities or characteristics of those products that the Domain Definition accommodates but that are not attainable with the provided Domain Implementation baseline.
- Evaluate the impact of the Domain Implementation baseline on the efficiency and effectiveness of application engineering projects. Identify improvements in realistic and practical Domain Implementation usability.

Step: Domain Delivery Activity

Action Deliver Domain Implementation to an application engineering project, assist with its use, and identify needed product or process improvements (available in future releases).

Input Domain Implementation

Result None

Heuristics

- Initiate an instance of this activity upon creating each targeted application engineering project; continue this activity until the project terminates.
- Provide copies of process documentation (i.e., the Application Engineering Process Standard and the Application Engineering User's Guide) to the engineers and managers of the application engineering project.
- Install the Application Engineering Environment (and subsequent upgrades), including the Adaptable Components from the Product Implementation, for project use and check it for proper operation. Follow installation procedures documented in the Application Engineering Environment Support Manual.

- Use Application Engineering Training Courses to provide formal instruction to application engineers (including project managers) in the proper and effective use of the Application Engineering process and the associated Process Support and Product Implementation work products. Explain use of the Application Engineering Process Standards and Procedures for understanding of the process in-the-large and the Application Engineering User's Guide for understanding and performing the process in-the-small. Explain the use of the Application Engineering Environment as described in the User's Guide.

- Provide consultation services to application engineers as they perform Application Engineering. Consulting requires extensive domain knowledge to answer application engineers' questions accurately and fully. Consultants should be knowledgeable in all aspects of Domain Implementation. There also needs to be a core of expert consultants who are sufficiently familiar with other domain engineering work products to provide complete, detailed, in-depth information, rationales, and assistance when complex problems are encountered by a project. In small organizations, the entire domain engineering team may be called on as a consulting resource.

- In response to the delivery services provided, application engineers will identify problems, improvements, and future needs that Domain Engineering should consider for possible action. Some of these ideas will relate directly to meeting customers' needs while others will relate to how efficiently application engineers can use the process and associated domain assets. Properly record and communicate these ideas and their motivations to the rest of Domain Engineering as feedback from application engineering. This is a key part of Project Support and is essential to continual project and market responsive improvement and evolution of a domain.

3.2 Risk Management

Risk	The needs of a particular application engineering project will conflict with a simple interpretation of prescribed standards and procedures.
Implication	The project will be forced to work in conflict with that interpretation and to be less effective and efficient.
Mitigation	Try to interpret standards and guidelines flexibly so that they best fit the needs of each project. Be aware of variations in the Process Support, particularly in environment installation, that support different needs. Tailor consultation and training to each particular project's needs.
Risk	Changes in the circumstances of a project may conflict with the previous interpretation of prescribed standards and procedures.
Implication	The project will be forced to work around obsolete support and will be less effective and efficient than necessary.

Mitigation Reconsider the support given to a project whenever circumstances change significantly. Be prepared to adjust the environment, training, and consulting advice to fit current needs better.

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency Application engineers are having difficulty using the Application Engineering process or Domain Implementation to develop applications.

Source Process Support Development Activity

Response
- Suggest better ways to the project for performing the process within the prescribed standards.
- Document the nature of the difficulties and suggest improvements in the prescribed process or in its documentation, automation, or training.

Contingency Particular, noncommon customer requirements cannot be expressed in an Application Model.

Source
- Domain Definition Activity
- Process Requirements Activity
- Process Support Development Activity

Response
- Identify unrecognized domain variations that application engineers need.
- Suggest to the project how it can best work around current limitations.

Contingency Projects cannot build applications that provide required behavior or that satisfy required constraints on resource usage (e.g., time, space, reliability).

Source
- Domain Analysis Activity
- Domain Implementation Activity

Response
- Document the requirements or constraints that cannot be satisfied.
- Suggest to the project how to best remedy the behavior or resource usage.

4.2 Feedback From Product Consumers

None

This page intentionally left blank.

AE. APPLICATION ENGINEERING OVERVIEW

1. GETTING STARTED

Application Engineering is a Synthesis process for creating and supporting an application product that satisfies specified customer requirements. A product is represented by a set of associated work products that result from analysis of those requirements. Application Engineering is characterized by a comprehensive life-cycle process for the management, analysis, production, and support of the members of a product family. This is similar in purpose to a conventional software development process, but is tailored to the problems and the needs of projects in a particular business area. Such a business-area focus allows for the systematic reuse of standardized work products within and among projects in that business area.

Domain Engineering identifies a set of characteristic decisions for a business area that determine how a product can be tailored to meet particular needs. It also provides standardized work products in a form that supports tailoring to those decisions. Application engineering concentrates on the analysis of customer requirements to resolve those decisions. The result is a model of a corresponding product that can be evaluated according to those requirements. When a model is found to be acceptable, it is used to drive the generation of tailored work products that implement the model. After the resulting work products are verified to the model, they are delivered to customers for further evaluation and use.

1.1 Objectives

The objectives of Application Engineering are to:

- Understand the needs of customers, balancing concerns of cost versus value, to produce a product that fulfills those needs most effectively
- Organize and direct resources for the production and support of the product
- Produce software and documentation that support the delivery and use of the product
- Leverage standardized process and work products for a domain to produce required results effectively, predictably, and profitably

1.2 Required Information

Application Engineering requires the following information:

- Application Engineering Process Support
- Customer requirements

1.3 Required Knowledge and Experience

Application Engineering requires domain, business, and software knowledge and experience in:

- The problems that the products in the domain are intended to solve and the engineering tradeoffs to be considered in creating a viable solution
- Understanding and interpreting customer requirements and developing applications that satisfy those requirements
- The management, production, and delivery of software work products
- How to use the Application Engineering Process Support to develop an application
- The information required by the Application Engineering process employed by the project

2. PRODUCT DESCRIPTION

Application Engineering creates four work products:

- *Project Plan.* The Project Plan establishes standard practices and procedures and defines tasks for incremental development with milestones and resource allocations.
- *Application Model.* The Application Model describes a deliverable system in terms of requirements and engineering decisions.
- *Application Product.* The Application Product consists of an application and associated user documentation work products. It is derived mechanically from the Application Model to provide a capability specified by customer requirements.
- *Delivery Support.* Delivery Support includes testing, installation, and training materials. It is derived mechanically to be consistent with the Application Model to support the delivery of the Application Product.

3. PROCESS DESCRIPTION

The Application Engineering process defined here is prototypical in the sense that an objective of Domain Engineering is to define such a process tailored to the needs of a domain. This Application Engineering process consists of the four activities shown in Figure AE-1.

3.1 Procedure

Step: Project Management Activity

Action	Plan, monitor, and control project resources to deliver a product.
Input	Customer requirements
Result	Project Plan

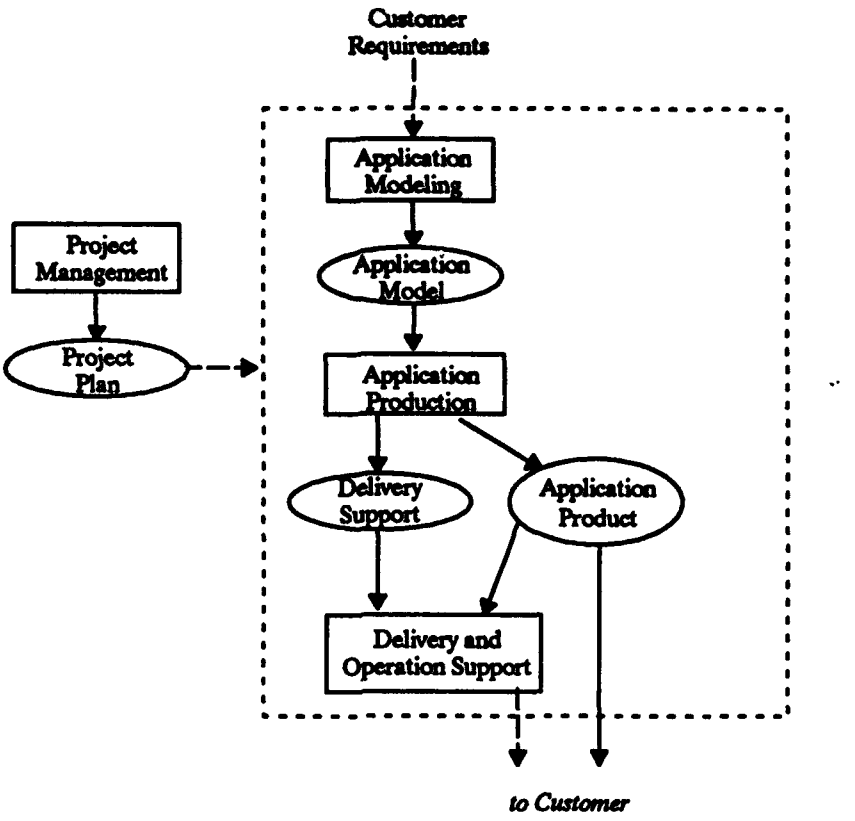


Figure AE-1. A Prototypical Application Engineering Process

Heuristics

- Institute policies and procedures in accordance with the Application Engineering Process Standard of Application Engineering Process Support. Allocate time and budget for personnel to receive needed training in use of Application Engineering Process Support. Domain Engineering Project Support staff will deliver, including installing automation, and support effective use of Application Engineering Process Support.

- Create a plan that reflects the process defined and supported by Application Engineering Process Support. Revise the plan to reflect unexpected progress or delays.

- For more accurate monitoring and control of the effort, set specific, measurable objectives and schedule repeated short iterations through the process to achieve them. Build the required product incrementally and review interim capabilities with the customer and users in order to maximize opportunities for feedback that can ensure timely delivery of a valid result.

- Coordinate development and performance of the plan with Domain Management both to exploit any forthcoming domain development and to schedule domain improvements (in process or product) needed by this project.

Step: Application Modeling Activity

Action Resolve decisions to specify a required product, based on an analysis of customer requirements, and evaluate it with respect to customer or technical constraints on an acceptable solution.

Input Customer requirements

Result Application Model

Heuristics

- The User's Guide of Application Engineering Process Support defines an application modeling process for products within a domain. This process provides a framework, notation, and mechanisms for:

  - *Specification.* Analyzing customer requirements and expressing them in an Application Model as a set of decisions that describe an Application Product which should satisfy those requirements.

  - *Validation.* Evaluating the functional adequacy of a modeled Application Product by analyzing the static (consistency and completeness) and dynamic (execution behavior in a simulated environment) characteristics implied by the Application Model.

  - *Assessment.* Evaluating relevant properties of alternative Application Models to determine, qualitatively and quantitatively, which will result in a system that best satisfies the customer's operational and product quality requirements.

  The Application Engineering Environment provides associated automated support for this process.

- Perform the prescribed application modeling process to create an Application Model that expresses customer requirements. Extend the model, as necessary, to reflect engineering decisions needed to specify a complete product. Review your understanding of those requirements, as expressed in the model, with customer (and user) representatives.

- Use provided mechanisms to specify, validate, and assess alternative Application Models and review the results with customer representatives. If one of the provided mechanisms is simulated execution, review use of the simulated application with customer and user representatives.

- Based on your perception of the risks, rapidly build an approximate (i.e., complete but inaccurate) model of the complete product or partial models of poorly understood aspects of the product. For parts of the customer requirements that are incomplete or unclear, create alternative (partial) models that can be compared in review and evaluation with the customer.

- If the implications for the Application Product of aspects of an Application Model are not clear, seek clarification and elaboration from Domain Engineering Project Support staff. Project support can determine how decisions (individually and in combination) affect each of the work products that will make up the Application Product.

- When some aspect of customer requirements cannot be expressed accurately in an Application Model or acceptable properties cannot be attained for the product, seek Domain Engineering assistance. For particular problems, there may be workarounds or alternative ways of describing particular facets of the product.

Step: Application Production Activity

Action Create a standardized product and accompanying delivery support work products in accordance with an Application Model.

Input Application Model

Result

- Application Product
- Delivery Support

Heuristics

- Application Engineering Process Support provides a mechanical, possibly automated, process for creating an Application Product and Delivery Support, given a valid Application Model representing customer requirements and engineering decisions.

- For each work product that makes up the Application Product and Delivery Support, there is a manual (documented in the User's Guide) or automated (implemented in the Application Engineering Environment) generation procedure, driven by the Application Model, for constructing a tailored version of the work product.

- In some cases, there may be facilities in Application Engineering Process Support for special component implementation. These facilities support the direct construction of highly variable parts of particular work products without violating the integrity of the Application Model. Such modifications generally entail greater effort and risk of error, both initially and when requirements change.

Step: Delivery and Operation Support Activity

Action Deliver the Application Product to the customer, support its use, and evaluate its effectiveness.

Input

- Application Product
- Delivery Support

Result None

Heuristics

- Delivery Support provides the mechanisms needed to deliver an Application Product to the customer. Delivery includes installing the Application Product into its operational environment and training users in its operation. Procedures for accomplishing this are part of the User's Guide of Application Engineering Process Support with additional specifics included as part of Delivery Support.
- Application engineers provide consulting assistance as needed for users to become effective with the Application Product. This encompasses both initial training and subsequent troubleshooting if unexpected behaviors are detected.
- Operation support includes analyses of the effectiveness of the Application Product for users. Application engineers should identify and document any problems encountered and any additional or changed needs revealed as the Application Product is used.

3.2 Risk Management

None

4. INTERACTIONS WITH OTHER ACTIVITIES

4.1 Feedback to Information Sources

Contingency The standardized product family is inadequate to support the needs of particular customers.

Source Domain Engineering

Response Describe why the standardized product family is deemed inadequate, and suggest how it can be improved.

Contingency The standardized Application Engineering process is inefficient or leads to less-than-ideal results for this project.

Source Domain Engineering

Response Describe why the standardized Application Engineering process is deemed inefficient or inadequate, and suggest how it can be improved.

Contingency The Application Engineering Process Support provided is incomplete or deficient.

Source Domain Engineering

Response Describe the inadequacies in the Application Engineering Process Support. Indicate which sections are incomplete or deficient. These may include:

missing work product families, an incomplete User's Guide, errors in the User's Guide, improperly described Adaptable Components, malfunctioning generation procedure(s), and bugs in interface software provided by Domain Engineering.

4.2 Feedback From Product Consumers

Contingency The customer requests new or modified capabilities for the system.

Source Customer

Response

- Build a new version of the system.
- Reject the suggestions as out of scope.
- Ask Domain Engineering to make necessary changes in the domain.

Contingency The customer identifies system anomalies, changes to the target environment, inadequate system performance, or inadequate reliability.

Source Customer

Response

- Correct system anomalies, accommodate changes to the target environment, tune the system.
- Reject changes as out of scope.
- Ask Domain Engineering to make necessary changes in the domain.

This page intentionally left blank.

APPENDIX: MATURITY ASSESSMENT AND FUTURE EVOLUTION

This is the first release of the Reuse-Driven Software Processes Guidebook. This release describes two reuse-driven software processes that are oriented toward different stages of reuse capability from the perspective of the Consortium's RCM. A complete Reuse-Driven Software Processes Guidebook would describe a continuum of reuse-driven software processes from which an organization, at any stage of reuse capability, could derive the definition of a suitable process for that organization.

As a whole, the Consortium considers the opportunistic process description in Part Opp of this guidebook to be at the exploratory level in the following maturity scheme. The leveraged process description in Part Lev is approaching the developmental level in the following maturity scheme:

- ***Exploratory.*** Many sections are immature, incomplete, or missing; incomplete sections are limited in detail.
- ***Developmental.*** All sections are present but some are incomplete; complete sections have been used on at least one pilot project or case study.
- ***Functional.*** All sections are complete and have been independently validated in use on several pilot projects.
- ***Production.*** All sections are complete, validated, and constitute a mature representation of organizational policies and concerns.

To advance beyond functional-level maturity, the guidebook must be refined to meet particular standards for production use in a particular organization. Quality improvements can continue based on organizational experience with using it.

The Consortium will continue to refine the guidebook until all its sections reach functional-level maturity. This goal assumes pilot project participation by government and industrial organizations. Such participation is essential to improving the depth and quality of the guidebook.

At this stage, the Consortium considers the guidebook to be usable in the following ways:

- On pilot and low-risk production projects
- Under the guidance of a technologist, acting as a transfer agent, who can interpret, elaborate, and fill in guidebook material sufficiently for other participants to be effective in performing Synthesis
- With Synthesis-novice managers and engineers as primary participants, to supply essential domain expertise and to be trained in Synthesis practices

The maturity scheme also applies to the individual sections in Part Opp and Part Lev of the guidebook. Tables App-1 and App-2 show the current maturity of each section in Part Opp, Opportunistic Synthesis, and in Part Lev, Leveraged Synthesis, respectively.

Table App-1. Maturity Scheme for Part Opp, Opportunistic Synthesis

Section	Maturity
OV. Overview	Exploratory
DE. Domain Engineering Overview	Exploratory
DE.1. Domain Management	Exploratory
DE.2. Domain Analysis	Exploratory
DE.2.1. Domain Definition Activity	Exploratory
DE.2.2. Domain Specification Activity	Exploratory
DE.2.2.1. Decision Model Activity	Exploratory
DE.2.2.2. Product Requirements Activity	Exploratory
DE.2.2.3. Process Requirements Activity	Exploratory
DE.2.2.4. Product Design Activity	Exploratory
DE.2.2.4.1 Product Architecture Activity	Exploratory
DE.2.2.4.2 Component Design Activity	Exploratory
DE.2.2.4.3 Component Design Activity	Exploratory
DE.2.3. Domain Verification Activity	Exploratory
DE.3. Domain Implementation Activity	Exploratory
DE.3.1. Product Implementation Activity	Exploratory
DE.3.1.1. Component Implementation Activity	Developmental
DE.3.1.2. Generation Implementation Activity	Exploratory
DE.3.2. Process Support Development Activity	Exploratory
DE.4. Project Support Activity	Exploratory
DE.4.1 Domain Validation Activity	Omitted
DE.4.2 Domain Delivery Activity	Omitted
AE. Application Engineering Overview	Exploratory
AT. Advanced Topics	Omitted

Table App-2. Maturity Scheme for Part Lev, Leveraged Synthesis

Section	Maturity
OV. Overview	Exploratory
DE. Domain Engineering Overview	Exploratory
DE.1. Domain Management	Exploratory
DE.2. Domain Analysis	Exploratory
DE.2.1. Domain Definition Activity	Developmental
DE.2.2. Domain Specification Activity	Exploratory
DE.2.2.1. Decision Model Activity	Developmental
DE.2.2.2. Product Requirements Activity	Developmental
DE.2.2.3. Process Requirements Activity	Exploratory
DE.2.2.4. Product Design Activity	Exploratory
DE.2.2.4.1 Product Architecture Activity	Exploratory
DE.2.2.4.2 Component Design Activity	Exploratory
DE.2.2.4.3 Component Design Activity	Exploratory
DE.2.3. Domain Verification Activity	Exploratory
DE.3. Domain Implementation Activity	Exploratory
DE.3.1. Product Implementation Activity	Exploratory
DE.3.1.1. Component Implementation Activity	Developmental
DE.3.1.2. Generation Implementation Activity	Exploratory
DE.3.2. Process Support Development Activity	Exploratory
DE.4. Project Support Activity	Exploratory
DE.4.1 Domain Validation Activity	Omitted
DE.4.2 Domain Delivery Activity	Omitted
AE. Application Engineering Overview	Exploratory
AE.1. through AE.4.	Omitted
AT. Advanced Topics	Omitted

This page intentionally left blank.

LIST OF ABBREVIATIONS AND ACRONYMS

ADARTS	Ada-based Design Approach for Real-Time Systems
AE	Application Engineering
AT	Advanced Topics
DE	Domain Engineering
DOD	Department of Defense
ESP	Evolutionary Spiral Process
IV&V	Independent Verification and Validation
Lev	leveraged
Opp	opportunistic
OV	Overview
RCM	Reuse Capability Model
SSS	System/Segment Specification
STARS	Software Technology for Adaptable Reliable Systems
Syn	Synthesis
TLC	Traffic Light Control Software System

This page intentionally left blank.

GLOSSARY

Abstract component	A family of components characterized by a defining abstraction and the decisions that are needed to distinguish among the members of the family (or to extract a concrete component).
Abstraction	A description of a collection of things that applies equally well to any one of them. Also, a concept that denotes the common properties of a family.
Activity	A step of a process for producing and/or evaluating work products to satisfy objective(s) supporting that process. An activity comprises other steps.
Adaptable Component	A Domain Engineering work product that implements a Component Design. See Abstract component.
Application	The hardware, software, and manual procedures that characterize a system.
Application Engineering	An iterative process for the design and development of a product that satisfies specified customer requirements. Its work products are an Application Model and an Application Product. For the leveraged Synthesis process, see Application Modeling (Activity), Application Production (Activity), Delivery and Operation Support (Activity), and Project Management (Activity).
Application Engineering Environment	Automated support provided for a prescribed Application Engineering process. See Process Requirements and Process Support and Infrastructure.
Application Engineering Process Support	A Domain Implementation, from the perspective of an Application Engineering project.
Application Model	A set of resolved requirements and engineering decisions, as specified by a Decision Model, that (partially) determine an instance of a family of systems. See Application Modeling Notation.

Application Modeling (Activity)	The Application Engineering activity that produces a validated Application Model sufficient to derive a production-quality Application Product that satisfies customer requirements. See Specification (Activity), Validation (Activity), and Assessment (Activity).
Application Modeling Notation	A notation for expressing an Application Model such that a complete Application Model is sufficient to distinguish uniquely any system of a domain. See Process Requirements.
Application Product	Software artifacts, including code and documentation, produced by the Application Production activity to satisfy customer requirements.
Application Production (Activity)	The Application Engineering activity that produces an Application Product, as specified by an Application Model, and Delivery Support. See Generation (Activity) and Special Components Implementation (Activity).
Architecture	A set of design structures that characterize a system and each associated artifact (i.e., work products).
Assessment (Activity)	The Application Modeling activity that produces analyses of the degree to which alternative Application Models satisfy the operational (e.g., performance, reliability) and product quality requirements of the customer.
Business area	A coherent market characterized by (potential) customers possessing similar needs.
Business-area knowledge	Information that characterizes the market for a domain including: • Current and future customer and contract profiles • Projected growth in contracts (or sales) • Value and marketability of features • Market analyses
Business-area organization	An organization whose mission is the production and delivery of products for customers in a specified business area.

Business objectives	The needs of a business-area organization that determine the scope and extent of a domain.
Candidate Components	A set of previously-built components that have been judged as qualified for potential use as raw material in the engineering of Adaptable Components.
Commonality	A characteristic of a domain that corresponds to a similarity among members of the associated family of systems. See Variability.
Component	A work-product fragment whose production is a managed work assignment. See Abstract component.
Component Design	The element of a Product Design that defines the design of a component identified in a Product Architecture.
Component Design (Activity)	The Domain Analysis activity that creates a Component Design.
Component Implementation (Activity)	The Domain Implementation activity that creates Adaptable Components, as specified by a Component Design.
Constraint	A limitation on decision(s).
Customer	The person(s) or organization(s) that specify the requirements, accept, and authorize payment for a product.
Customer requirements	A description of the capabilities, as understood by customers, required of a system and any constraints on the engineering of that system.
Decision	A choice among allowable alternatives.
Decision constraint	A set of constraints on the resolution of interdependent decisions.
Decision group	A logical grouping of decisions in the Application Modeling Notation that allows the application engineer to separate concerns when describing a system.
Decision Model	The element of a Domain Specification that defines the abstract form (concepts, decisions, and structure) of an Application Modeling Notation.
Decision Model (Activity)	The Domain Analysis activity that creates a Decision Model.

Decision specification	A specification of the set of decisions that suffice to distinguish among members of a family.
Delivery and Operation Support (Activity)	The Application Engineering activity that delivers an Application Product to customers and supports its use.
Delivery Support	Software artifacts produced by the Application Production activity to support the Delivery and Operation Support activity for delivery of an Application Product to customers.
Dependency constraint	A relationship specifying how decisions made by an application engineer limit subsequent decisions.
Derived requirements	Requirements that indicate characteristics specific to particular systems in the domain based on the decisions that an application engineer expresses in an application model.
Design structure	A set of relationships among a set of components that represent some characteristic of the aggregate. Examples for a program are dependency structures that define a program's static structure and process structures that define its dynamic behavior.
Document	A documentation component, including textual and graphical artifacts.
Domain	A product family and an associated production process supporting a product line.
Domain Analysis (Activity)	The Domain Engineering activity in which domain knowledge is studied and formalized as a Domain Definition and a Domain Specification. The expertise in a business area is formalized to create standards for problem descriptions and corresponding solutions. See Domain Definition (Activity), Domain Specification (Activity), and Domain Verification (Activity).
Domain Assumptions	The element of a Domain Definition that defines the guiding assumptions and justifications of domain scope and extent.

Domain Definition	An informal description of the scope, extent, and justification for a domain. See Domain Synopsis, Domain Glossary, Domain Assumptions, and Domain Status.
Domain Definition (Activity)	The Domain Analysis activity that creates a Domain Definition.
Domain Delivery (Activity)	The Project Support activity that assists Application Engineering projects in the effective use of Application Engineering Process Support.
Domain Engineering	An iterative process for the design and development of (1) a product family and (2) an Application Engineering process for producing members of that family. See Domain Management (Activity), Domain Analysis (Activity), Domain Implementation (Activity), and Project Support (Activity).
Domain evolution	Revision of a Domain Definition, Domain Specification, and associated Application Engineering Process Support to reflect changes in domain scope.
Domain Glossary	The element of a Domain Definition that defines the terminology of a domain.
Domain Implementation	A Product Implementation and Process Support that satisfies a Domain Specification.
Domain Implementation (Activity)	The Domain Engineering activity that creates support for Application Engineering projects in the form of a Domain Implementation. See Product Implementation (Activity) and Process Support Development (Activity).
Domain knowledge	Knowledge and expertise characteristic to a domain: • Relevant scientific theory and engineering practice • Capabilities and uses of existing systems • Past system development and maintenance experience and work products • Potential developments in related or supporting technology • Potential changes in customer needs

Domain Management (Activity)	The Domain Engineering activity that plans, monitors, and controls the activities and resources of a Domain Engineering organization and which coordinates domain development and evolution with client Application Engineering projects.
Domain objectives	An element of the Domain Plan Master Plan that defines the goals of domain development.
Domain Plan	Schedules, budgets, assignments, and progress evaluations for the management of a Domain Engineering organization.
Domain Specification	A specification of a standardized Application Engineering process and product family for a domain. See Decision Model, Product (Family) Requirements, Process Requirements, and Product (Family) Design.
Domain Specification (Activity)	The Domain Analysis activity that creates a Domain Specification.
Domain Status	The element of a Domain Definition that specifies the current scope, extent, and viability of a domain relative to its objectives.
Domain Synopsis	The element of a Domain Definition that is an informal description of a domain.
Domain Validation (Activity)	The Project Support activity that evaluates the quality and effectiveness of Application Engineering Process Support from the perspective of Application Engineering project needs.
Domain Verification (Activity)	The Domain Analysis activity that evaluates a Domain Implementation to determine compliance with the corresponding Domain Definition and Domain Specification.
Entrance criteria	Conditions that must be met before an activity can be started.
Exit criteria	Conditions that must be met before an activity can be considered successfully completed.
Family	A set of things that have enough in common that it pays to consider their common characteristics before noting specific properties of instances.

Feasibility	The degree to which an objective is amenable to solution with predictable resources and risk.
Feedback	Information communicated by the consumer of a work product to its producer regarding issues in the correctness, quality, and viability of the product.
Generation (Activity)	The Application Production activity that applies a Generation Procedure to an Application Model, Adaptable Components, and Special Components to produce software work products.
Generation Design	The element of a Product Design that specifies a Generation Procedure (i.e., the mapping from a Decision Model and Product Architecture to work products for an application).
Generation Design (Activity)	The Domain Implementation activity that creates a Generation Design.
Generation Implementation (Activity)	The Domain Implementation activity that creates a Generation Procedure.
Generation Procedure	The definition of a procedure for selecting, adapting, and composing components to create a work product.
Goal	A specific, time-related, measurable target.
Implicit requirements	Requirements that indicate characteristics that are common to all systems in a domain. (*COMMENT*: These are referred to as implicit because they are implicit to an Application Model [i.e., there are no decisions in the Decision Model that affect them]).
Infrastructure	Those mechanisms or attributes of an Application Engineering Environment that are not determined by the Domain Specification. See Process Support Development (Activity).
Instantiation	Creating a thing from a representation of an abstraction denoting a set of such things.
Iterative process	A process in which completion occurs only after repetition of producing and using activities results in refined work products.

Legacy Products	The element of a Domain Definition that is a collection of work products (or portions thereof) that are potentially useful raw material for developing other Domain Engineering work products. Legacy Products are derive from existing systems in the domain.
Life cycle	A sequence of distinct states of an entity beginning with its initial conception and ending when it is no longer available for use.
Metaprogram	A description of an abstract component that is sufficient, given a set of resolved decisions, to instantiate a corresponding concrete component.
Metaprogramming	A method for creating abstract components and extracting concrete components from them. See Instantiation.
Metaprogramming notation	A notation for defining and instantiating metaprograms.
Method	Guidance and criteria that prescribe a systematic, repeatable technique for performing an activity.
Methodology	An integrated body of principles, practices, and methods that prescribe the proper performance of a process.
Model	A representation of a thing from which analysis provides approximate answers to designated questions about the thing itself.
Module	A software component that consists of design, code, documentation, and test artifacts.
Objective	The intended or desired result of a course of action.
Plan	A designation of tasks and resource allocations for accomplishing a specified objective.
Process	A (partially) ordered set of steps, intended to accomplish specified objective(s).
Process engineering	The construction of a process appropriate to accomplish the objectives of an organization or project.

Process Requirements	The element of a Domain Specification that defines an Application Engineering process and concrete forms (syntax) of an associated Application Modeling Notation.
Process Requirements (Activity)	The Domain Analysis activity that creates Process Requirements.
Process Support	Standards and procedures, in the form of documents and supporting automation, that institute a standard Application Engineering process, as specified by a Process Requirements. See Application Engineering Environment.
Process Support Development (Activity)	The Domain Implementation activity that creates Process Support.
Product	The aggregation of all work products resulting from a process or activity.
Product Architecture	See Product (Family) Architecture.
Product Architecture (Activity)	The Domain Analysis activity that creates a Product Architecture.
Product Design	See Product (Family) Design.
Product Design (Activity)	The Domain Analysis activity that creates a Product Design.
Product Family	A family of products. See Family.
Product (Family) Architecture	The element of a Product Design that is a specification of the (adaptable) architecture of the products for a system (possibly as a set of components).
Product (Family) Design	The element of a Domain Specification that defines how an Application Model which satisfies the Decision Model determines the structure and content of an Application Product and Delivery Support. This includes the criteria by which components are selected and adapted to create fragments which are then composed into complete work products.
Product (Family) engineering	The development and evolution, consistent with a Domain Definition and Decision Model, of Product Requirements, Product Design, and Product Implementation corresponding to a product family.

Product (Family) Implementation	The implementation of a Product Design as sets of Adaptable Components and Generation Procedures.
Product (Family) Requirements	The element of a Domain Specification that defines the requirements of systems in a domain relative to a Decision Model.
Product Implementation	See Product (Family) Implementation.
Product Implementation (Activity)	The Domain Implementation activity that creates a Product Implementation. See Component Implementation (Activity) and Generation Implementation (Activity).
Product line	A collection of (existing and potential) products that address a designated business area.
Product Requirements	See Product (Family) Requirements.
Product Requirements (Activity)	The Domain Analysis activity that creates a Product Requirements.
Program	(1) An aggregation of software components that, when integrated with hardware, operates as a unit. (2) A directed, funded effort to acquire, develop, or maintain a product(s).
Project	An undertaking requiring concerted effort, which is focused on developing and/or maintaining a specific product. Typically, a project has its own funding, cost accounting, and delivery schedule.
Project Management (Activity)	The Application Engineering activity that plans, monitors, and controls Application Engineering process execution and provides feedback to Domain Engineering on desired product family and Application Engineering process modifications.
Project Plan	Schedules, budgets, assignments, and status evaluations for the management of an Application Engineering project.
Project Support (Activity)	The Domain Engineering activity that validates Application Engineering Process Support, delivers it to application projects, and supports its use. See Domain Validation (Activity) and Domain Delivery (Activity).

Reuse Library	The portion of an Application Engineering Environment that provides access to Adaptable Components.
Risk	A potential for incurring undesirable results.
Specialization	Constraining an abstraction to denote a subset.
Specification	A complete, precise description of the verifiable properties required of a work product.
Specification (Activity)	The Application Modeling activity that analyzes customer needs to produce an application model. The Application Model expresses requirements and engineering decisions that describe a system intended to satisfy those needs.
Step	Either an activity or an unelaborated action.
Structural constraint	A constraint that limits the number of instances of a decision group in a valid Application Model
Subdomain	The denotation of a subfamily of systems for which a corresponding domain denotes the larger family.
Subfamily	A subset of the members of a family that have some set of common characteristics not shared by any members of the family outside that subset. See subdomain.
Synthesis	A methodology for the construction of software systems as instances of a family of systems that have similar descriptions. Its primary distinguishing features are: • Formalization of domains as families of systems that share many common features, but which also vary in well-defined ways • System building reduced to resolution of requirements and engineering decisions that represent the variations characteristic of a domain • Reuse of software artifacts through mechanical adaptation of components to satisfy requirements and engineering decisions

	• Model-based analyses of described systems to help understand the implications of system-building decisions and evaluate alternatives
System	A collection of hardware, software, and people that operate together to accomplish a mission.
Task	A work assignment (i.e., subject to management accountability) to accomplish a specified objective.
Test scenario	A test component, which includes test procedures and test data artifacts.
User	The person(s) or organization(s) that will use the system for its intended purpose when it is deployed in its environment.
Validation	The evaluation of work products to determine whether they satisfy customer needs.
Validation (Activity)	The Application Modeling activity that produces analyses of the degree to which alternative Application Models satisfy the functional requirements of the customer.
Variability	A characteristic of a domain that corresponds to features that distinguish among members of the associated family of systems. See Commonality.
Verification	The evaluation of a work product to determine whether it meets its specification.
Viability	The degree to which benefits of a feasible undertaking dominate the costs of its performance, taking into consideration risk-induced uncertainties.
Work product	Any configuration-managed artifact.

REFERENCES

Balzer, Robert, and Neil Goldman 1979 — *Principles of Good Software Specification and their Implications for Specification Languages.* USC/Information Sciences Institute.

Booch, Grady 1987 — *Software Components with Ada.* Menlo Park, California: Benjamin-Cummings.

Borgida, Alexander 1985 — Features of Languages for the Development of Information Systems at the Conceptual Level. *IEEE Software* 1:63–72.

Burkhard, Neil 1992 — *Domain Engineering Validation Case Study—Synthesis for the Air Traffic Display/Collision Warning Monitor Domain,* SPC-92050-CMC. Herndon, Virginia: Software Productivity Consortium.

Campbell, Grady H., Jr. 1989 — *Abstraction-Based Reuse Repositories,* REUSE_REPOSITORIES-89041-N. Herndon, Virginia: Software Productivity Consortium.

Campbell, Grady H., Jr., Stuart R. Faulk, and David M. Weiss 1990 — *Introduction to Synthesis,* INTRO_SYNTHESIS_PROCESS-90019-N. Herndon, Virginia: Software Productivity Consortium.

Davis, Alan M., Edward H. Bersoff, and Edward R. Comer 1988 — A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering* SE-14:1453–61.

Department of Defense 1988 — *Military Standard: Defense System Software Development* (DOD-STD-2167A). Washington, D.C.: Department of Defense.

Dijkstra, E.W., O.J. Dahl, and C.A.R. Hoare, eds. 1972 — "Notes on Structured Programming." In *Structured Programming,* 1–82. London,England: Academic Press.

Eward, Mary, and Steven Wartik 1994 — "Introducing Megaprogramming at the High School and Undergraduate Levels." *Seventh SEI Conference on Software Engineering Education.* San Antonio, Texas. To be published in January 1994.

Heninger, Kathryn L. 1980 — Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Transactions on Software Engineering* SE-6: 2–13.

Heninger, Kathryn, J. Kallander, David L. Parnas, and John Shore 1978 — *Software Requirements for the A-7E Aircraft*. Memorandum Report 3876. Washington, D.C.: Naval Research Laboratory.

Humphrey, Watts S. 1989 — *Managing the Software Process*. Reading, Massachusetts: Addison-Wesley Publishing Company.

Kent, William 1978 — *Data and Reality*. North Holland, Amsterdam.

O'Connor, James, and Catharine Mansour 1992 — *Introducing Systematic Reuse to the Command and Control Systems Division of Rockwell International*, SPC-92020-N. Herndon, Virginia: Software Productivity Consortium.

Parnas, David L. 1976 — On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* SE-2:1–9.

Software Productivity Consortium 1991a — *Synthesis Guidebook*, SPC-91122-MC. Herndon, Virginia: Software Productivity Consortium.

1991b — *Systematic Reuse: The Competitive Edge*, SPC-91047-N. Herndon, Virginia: Software Productivity Consortium.

1992a — *Domain Engineering Guidebook*, SPC-92019-CMC, version 01.00.03. Herndon, Virginia: Software Productivity Consortium.

1992b — *Process Engineering with the Evolutionary Spiral Process Model*, SPC-92079-CMC. Herndon, Virginia: Software Productivity Consortium.

1992c — *Reuse Adoption Guidebook*, SPC-92051-CMC. Herndon, Virginia: Software Productivity Consortium.

1993 — *ADARTS Guidebook*, SPC-91104-MC, version 03.00.09. Herndon, Virginia: Software Productivity Consortium.

Wartik, Steven, and Rubén Prieto-Díaz 1992 — Criteria for Comparing Reuse-Oriented Domain Analysis Approaches. *International Journal of Software Engineering and Knowledge Engineering* 2, 3: 403–431.

Winograd, Terry 1979 — Beyond Programming Languages. *Communications of the ACM* 22: 391–401.