AD-A237 156

1 June 1991 Final

SLAYING THE SOFTWARE DRAGON ... A Look at How Software Engineering, the Ada Programming Language and Process Maturity Are Changing Software Development n/a

Col David R. Dick

Hq USAF/SCX
Washington D. C. 20330 n/a

The Armed Forces Communications and Electronics Association
International Headquarters n/a
4400 Fair Lakes Court
Fairfax, Va 22033-3899

Final project paper for AFCEA Senior Fellowship

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED A

The cost of software today consumes over ten percent of the DoD budget. Software costs, complexity, and size continue to rise because of the ever increasing dependence of weapons and general purpose systems on computers. Software engineering, the Ada programming language, and efforts to determine the performance and risk of software development organizations by measuring process maturity represent key initiatives by the DoD to improve the quality, reliability, and maintainability of the software DoD buys. This study examines and describes each of these areas based on current literature and interviews with officers of 37 companies engaged in software development for command, control, and communications systems.

Software engineering; Ada programming language; Process maturity; Software process maturity model 70 n/a

UNCLASSIFIED UNCLASSIFIED UNCLASSIFIED UL

The [illegible]
that [illegible]
[illegible]
[illegible]

[illegible]

[illegible]

(2)

"SLAYING THE SOFTWARE DRAGON"

A Look at How Software Engineering, the Ada Programming Language, and Process Maturity Are Changing Software Development.

Col David R. Dick, USAF
Senior Fellow
The Armed Forces Communications and Electronics Association

June 1, 1991

ABSTRACT

The cost of software today consumes over ten percent of the DoD budget. Software costs, complexity, and size continue to rise because of the ever increasing dependence of weapons and general purpose systems on computers. Software engineering, the Ada programming language, and efforts to determine the performance and risk of software development organizations by measuring process maturity represent key initiatives by the to improve the quality, reliability, and maintainability of the software DoD buys. This study examines and describes each of these areas based on current literature and interviews with officers of 37 companies engaged in software development for command, control, and communications systems.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT)

TABLE OF FIGURES

Chapter 1
INTRODUCTION

"Software is like entropy. It is difficult to grasp, weighs nothing, and obeys the Second Law of Thermodynamics; it always increases."[1]

Readers of this paper are probably well aware of the degree that computers and software have become the staples of modern society. Virtually everything we touch, do, or imagine is improved by the magic of computers. But for many, underlying this phenomenon is the realm of software; unseen, untouched, and often misunderstood. Software carries great assumptions: assumed to always work, to cost little, and to be capable of doing almost anything the capacity of the computer will allow. The reality is, of course, different.

The First "Software War"

Within the Department of Defense (DoD) the growth of computers to support increasing complexity and effectiveness of weapons, command and control, communications, logistics, administrative, and a myriad of other systems was vividly displayed during Desert Shield/Desert Storm. In an interview with Defense News, Lieutenant General Billy Thomas, Deputy Commander for Research, Development, and Acquisition, Army

[1]Norman R. A. Augustine, Augustine's Laws, New York: Viking Penguin Inc., 1986, p 118.

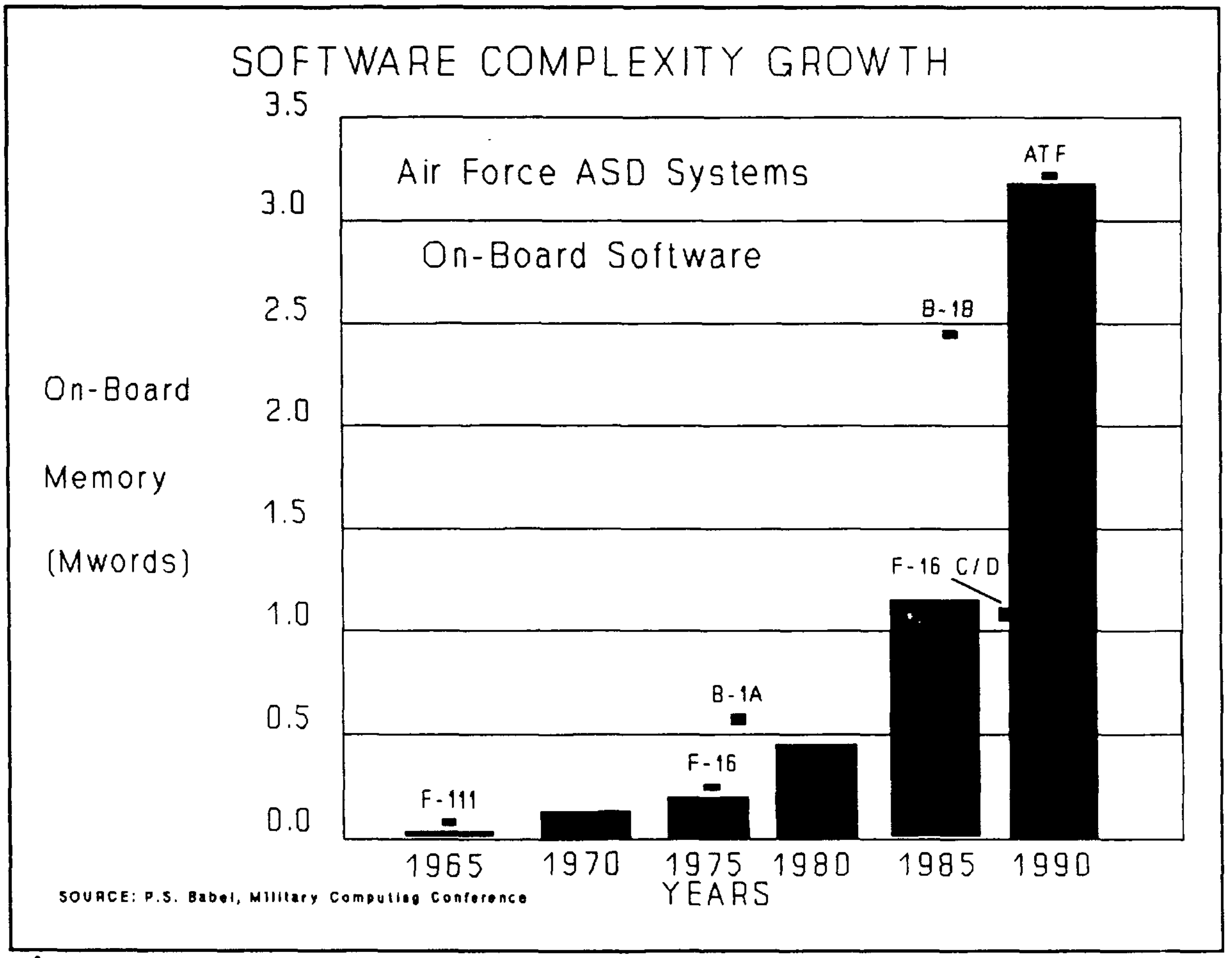


Figure 1

Material Command, called Desert Storm our first software war and said that "software is the crux of our weapon systems, and becoming more so." He further pointed out that unless the DoD devotes more attention to software development, the US military will find itself against adversaries with smarter weapons.[2]

These gains are a result of overwhelming improvements in computer hardware technology and tremendous increases in the size and complexity of software. An often used but nevertheless

---

[2]"Army Encourages Software Integration," Defense News, April 29, 1991, p 10.

appropriate example is the difference between the Vietnam era F-4 aircraft with virtually no software on board to an F-16D with about 236,000 lines of code (Figure 1). This growth has in turn raised concerns about the reliability, quality, and cost of the software the DoD buys.

Software Costs

The DoD's software costs in 1990 were estimated to be $31B, or about ten percent of the entire DoD budget. This cost has grown from about $9B in 1985 and approximately $3B in 1974. The 1990 figure represents almost one third of the total software revenue in the United States, making the military the nation's largest consumer of software.[3]

Software costs have risen disproportionally as a factor in system procurement. Ten years ago Barry Boehm, in his book Software Engineering Economics, stated, "...today the computer system that we buy as 'hardware' has generally cost the vendor about three times as much for the software as it has for the hardware."[4] This trend continues, except that the relationship

[3]From a speech by Edward L. Lafferty, Mitre Technical Director, to the Tri-Ada 90 Conference, Baltimore, Maryland, December 4, 1990.

[4]Barry Boehm, Software Engineering Economics (Prentice-Hall, Inc), 1981, p 17.

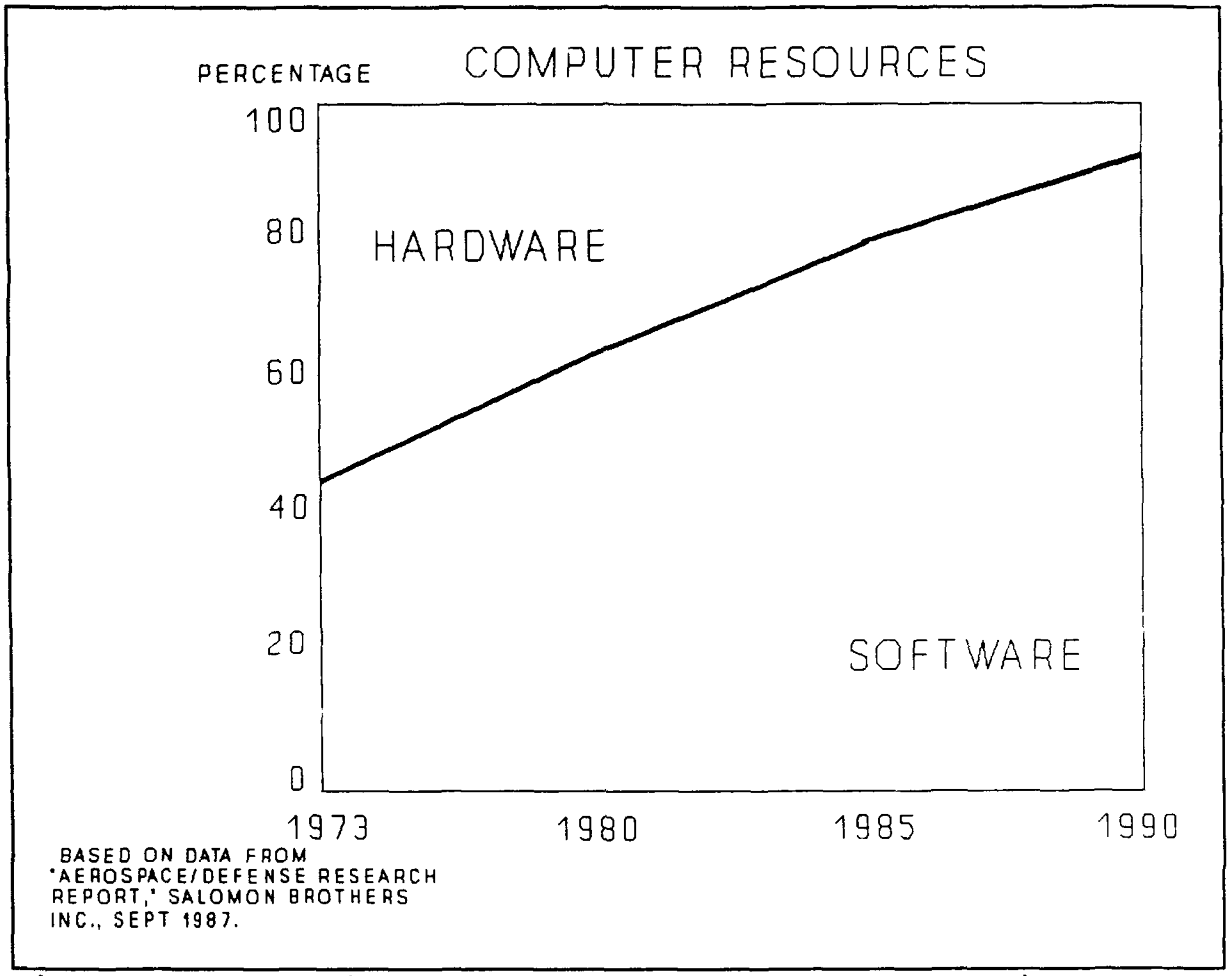


Figure 2: Comparison of Software to Hardware Costs in Computer Systems

is even more pronounced as seen in Figure 2.[5]

The Software Crisis

DoD's growing investment in software occurred at the same time a "software crisis" has been acknowledged to exist in this country. This crisis reflected problems that have contributed to

[5]Taken from a chart in AFCEA's Professional Development Center Course 139D, "Management of Ada-Based Systems."

major cost and schedule overruns and crisis characterized by the virtual shutdown of AT&T's long distance network on January 15, 1990. Software problems have been an area of intense study by the Congress, the DoD, and industry.[6] Recently, a series of articles in the Washington Post[7] described this nation's insatiable hunger for software in everything from medical equipment and automobiles to military systems. One of the conclusions from the series was that our ability to produce the amount and quality of software needed is essentially stagnant.[8]

DoD Initiatives to Improve Software Development

DoD has taken a number of initiatives to improve the software acquisition process and force economies on software throughout its life cycle. The principle initiative within DoD itself was the development of the Ada programming language. Designed initially as the language of choice for all critical

---

[6]Over 30 studies have been written on the "Software Crisis." Two of these are especially valuable in understanding many of the problems associated with software as it applies to the DoD. These studies are the Report of the Defense Science Board Task Force on Military Software, Defense Science Board/Office of the Under Secretary of Defense for Acquisition, September 1987, and Bugs in the Program, Problems in Federal Government Computer Software Development and Regulation, Committee on Science, Space, and Technology, U.S. House of Representatives, Subcommittee on Investigations and Oversight, September 1989.

[7]"The Software Snarl," The Washington Post, December 9 - 12, 1990.

[8]Evelyn Richards, "Society's Demands Push Software To Upper Limits," The Washington Post, December 9, 1990, p A24.

weapon system software, its use has been expanded as the preferred language for virtually all software applications. Associated initiatives included the establishment of the Ada Joint Program Office (AJPO), the Software Engineering Institute (SEI), and the Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office. The AJPO, as its name implies, was established to manage the introduction of Ada and, in particular, effect technology transfer into major Defense programs through the use of Ada. The STARS program and the SEI were created to advance software engineering technology and accelerate the use of software engineering techniques in the development of DoD systems.

The Study

This paper focuses on three areas; software engineering, Ada, and the maturity of the software development process that are central to these initiatives. These subjects are interrelated and cannot be treated independently. They must be seen as a set of interlocking principles and techniques that have evolved to make software and systems that are more reliable, cost effective, and responsive to the needs of the ultimate user.

The subject of software engineering provides a framework that describes much of the effort underway to move the business of software development from what has long been considered a "black art" to a structured process governed by the principles of science. This discussion also helps understand the Ada programming language which was created to support and facilitate software engineering principles.

Ada is examined first in terms of the central features of the language. But, most importantly, its acceptance and use is explored in light of increased pressure by the DoD and Congress to mandate its use for all computer software developed for the DoD barring certain exceptions.

Finally, process maturity as it is defined and measured by the SEI ties the picture together. The SEI model of process maturity and the determination of a software organization's placement and improvement in the context of the model provide a look at how facets of total quality management (TQM) are being introduced into software development. The discussion of process maturity measurement is particularly relevant because it is beginning to be widely used as a determinant of risk and performance in the evaluation and award of competitive procurement contracts.

The paper examines these subjects and the results of DoD's initiatives. The views of many in industry and government provide the assessment. The paper is intended to provide someone not familiar with the issues and background of software development an understanding of some of the problems and management techniques.

Chapter 2
METHODOLOGY

The study was conducted in two phases. The first was a review of much of the available literature on software engineering and Ada, including technical reports, articles, books, and policy documents from OSD and the services. A bibliography is contained at the end of this paper.

The second phase, and by far the most important and unique, consisted of a series of interviews with 37 companies who are major suppliers of software for the DoD, primarily in applications supporting telecommunications and command and control systems. A few of the interviews also examined software developed for commercial purposes. These interviews provided information in three areas of concern:

(1) Software engineering practices and use within the company.

(2) How the DoD mandate for the use of the Ada programming language helps or hinders software development and what is the future for Ada.

(3) Efforts within the company to improve the process of software development.

A list of questions was drawn up to guide the interview process and is contained in Appendix 1. However, the interviews were generally unstructured, information was collected on any perspective that the individual being interviewed felt needed to be aired. Those interviewed were generally at the project manager level, although, they ranged from senior programmer to vice president.

Chapter 3
SOFTWARE ENGINEERING

"The tar pit of software engineering will continue to be sticky for a long time to come. One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate and complex of man's handiwork. The management of this complex craft will demand our best use of new languages and systems, our best doses of common sense, and a God-given humility to recognize our fallibility and limitations."

Frederick P. Brooks, Jr
The Mythical Man Month[9]

During the interviews for this project, those interviewed repeatedly pointed to improved quality and reliability of software as the most important factors in their company's software development. In every case but one, these improvements were attributed to the adaption of software engineering concepts. When asked to describe which concepts were the most important, several answers were given. For some, software engineering was a standardized process involving "best current practices" or checklists that promote standards in coding, naming conventions, documentation, and tool sets. In other cases, software engineering meant peer reviews, seminars and courses in software engineering, and the collection of metrics to better define and assess the process of software development. The common theme throughout was that those who practiced software engineering felt their company's credibility in the marketplace was improved

[9]Frederick P. Brooks, Jr, The Mythical Man-Month, Addison-Wesley Publishing Co, 1975.

because they not only produced better quality software, but were also more responsive to their customers, and over the long term lowered costs. For most, the introduction of software engineering had occurred within the last two to five years.

Software Engineering Defined

Software engineering is not new. It has existed as an identifiable discipline for about twenty years.[10] The term was originally coined to focus attention of a NATO workshop on software production in 1968. Since then, it has come to identify management practices, software tools, and design activities for software development. The Society of International Electronic and Electrical Engineers (IEEE) defines software engineering as a systematic approach to the development, operation, maintenance, and retirement of software.[11] Another definition put forth by a leading expert on software development, Dr. Barry Boehm, argues that software engineering must also accommodate human and economic concerns as well as programming mechanics.[12]

---

[10]Mary Shaw, "Prospect for an Engineering Discipline of Software," IEEE Software, November 1990, p 15.

[11]Ted G. Lewis and Paul W. Oman, "The Challenges of Software Development," IEEE Software, November 1990, p 10. In contrast to its definition of software engineering, IEEE defines software development as the process by which user needs are transformed in design, design is implemented in code, and code is tested, documented, and certified for use.

[12]Barry W. Boehm, Software Engineering Economics, (Prentice-Hall, Inc.), 1981, p 10.

For the uninitiated, the concept of software engineering embodies a confusing mix of abstract ideas and terms that are applied across a continuum stretching from requirements definition to software maintenance.

The "Software Problem"

The advancement of software engineering is largely due to the increases in the size and complexity of software. These changes in size and complexity have made the problems inherent in the software development and support processes much more evident. The nature of these software related problems are economic, managerial, and technical, and they involve the production, maintenance, and use of systems. A view that these issues can be lumped together as the "software problem" and dealt with as a single issue is overly simplistic and has been proven not to work.[13]

Software development for most of the approximately forty years that it has been practiced has been considered akin to an art form. The building of software is basically a product of the human mind. It is a labor intensive endeavor; one that does not reduce easily to tools, mass production, or standard parts.[14] Traditionally, a person trained in writing software was

---

[13]Mary Shaw, Beyond Programming-in-the Large, SEI-86-TM-6, May 86, p 4-6

[14]Evelyn Richards, "Society's Demands Push Software To Upper Limits," The Washington Post, December 9, 1990, p A24.

considered capable of producing code for any project no matter how large or complex. This situation has been compared to a person who recently built a dog house in his back yard being considered equally capable of building a custom house or for that matter a skyscraper.[15]

Software today is commonly in the hundreds of thousands or millions of lines of code with disparate parts written by hundreds of people often over a period extending for several years. The real complexity comes in melding these parts together. People with little understanding or knowledge of how other parts were developed must make the whole program work within a specified schedule and cost.

This complexity fuels a key concern expressed during many of the interviews. Government and corporate management are often baffled about how best to manage and create structure for something that can not be seen or touched. A classic example is use of the waterfall model to define the software life cycle (Figure 3). This approach has been traditionally employed in the development of computer systems. It is successful for hardware development. The focus of the waterfall model is on documentation. Design milestones assume there is a clear understanding of the user's requirements at the outset of a program. However, often in the development of systems where software plays a significant role the requirements are not well

[15]This comparison is used by Eileen Quann in her course Management of Ada Based Systems, AFCEA Professional Development Center, Course # 139.

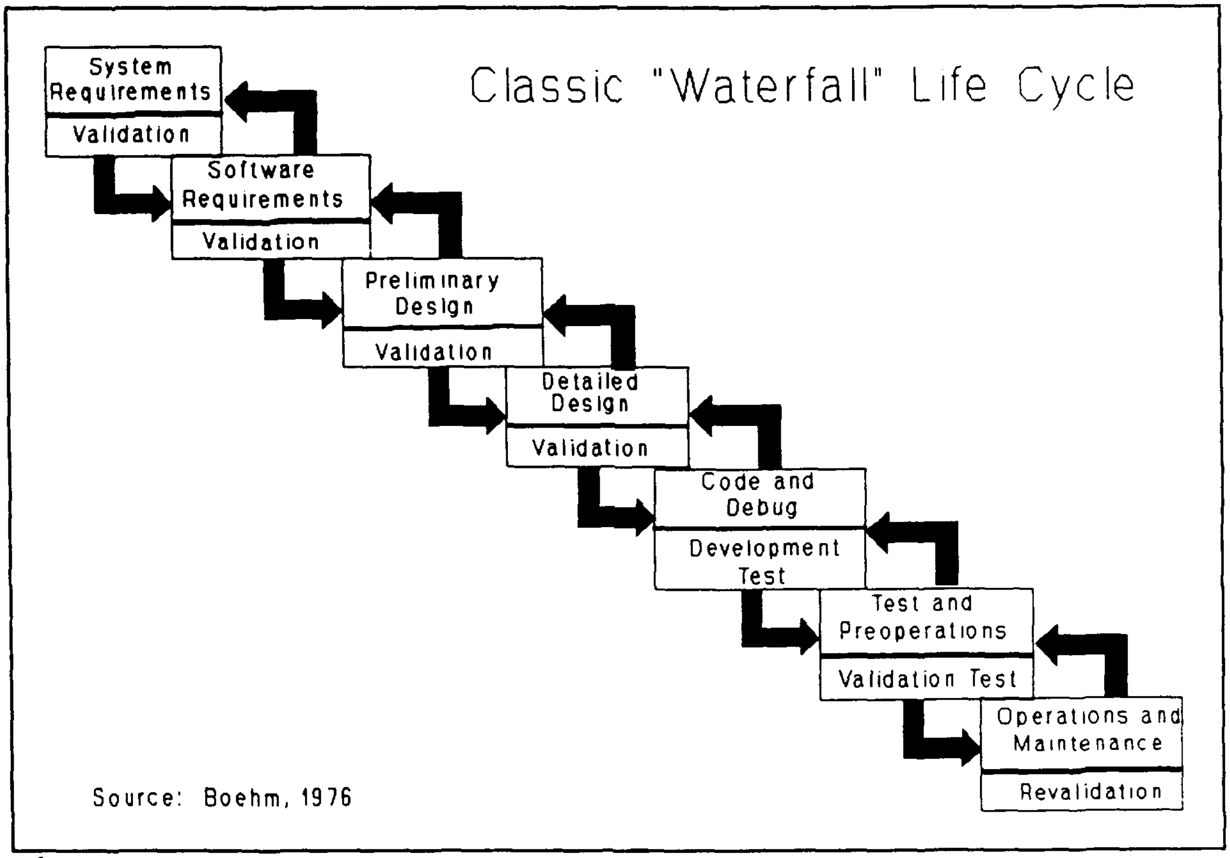


Figure 3

understood. Frequently, the user doesn't know what he wants nor does he have an appreciation of what the capability being provided can do, but is counting on the flexibility software provides to evolve the system.

The need to "get productive" and demonstrate progress pushes developers to start coding before requirements and design are adequately defined. The result is that most problems are not discovered until the testing phase near the end of the program.

This is cited as a leading reason software is late and over cost.[16] It can cost six to one hundred times more to fix errors found during the testing and maintenance phases of a program than during the requirements definition and design phases.[17]

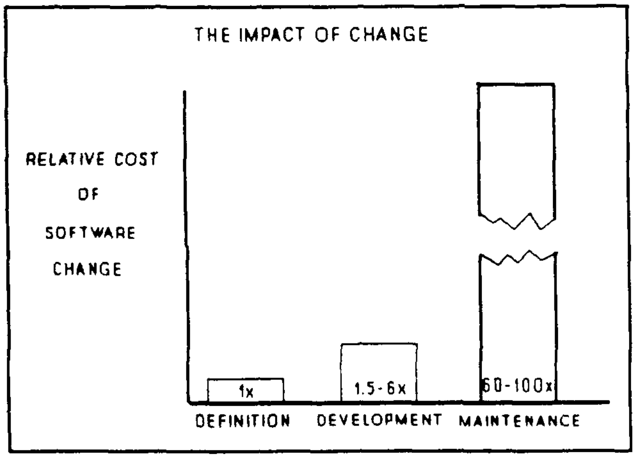
The "Engineering" in Software Engineering

Software engineering brings science to the art of software development in the same sense that Mary Shaw describes any accepted engineering practice as, "emerging from the commercial practice by exploiting the results of a companion science."[18] Her depiction of this evolution is shown in Figure 5. Many argue

---

[16]U.S. Congress, House, Committee on Science, Space, and Technology, Subcommittee on Investigations and Oversight, Bugs in the Program Problems in Federal Government Computer Software Development and Regulation, September 1989, p 9.

[17]From a briefing by Eugene Bingue to the Software Technology Support Center, Hill AFB, Utah, conference "The New Era - Software Technology," 16 - 17 April 1991, Salt Lake City, Utah (see accompanying figure). These estimates have been supported by several of the interviews with corporate managers.



[18]Mary Shaw, Prospects for an Engineering Discipline of Software, CMU/SEI-90-TR-20, September 1990, p 14.

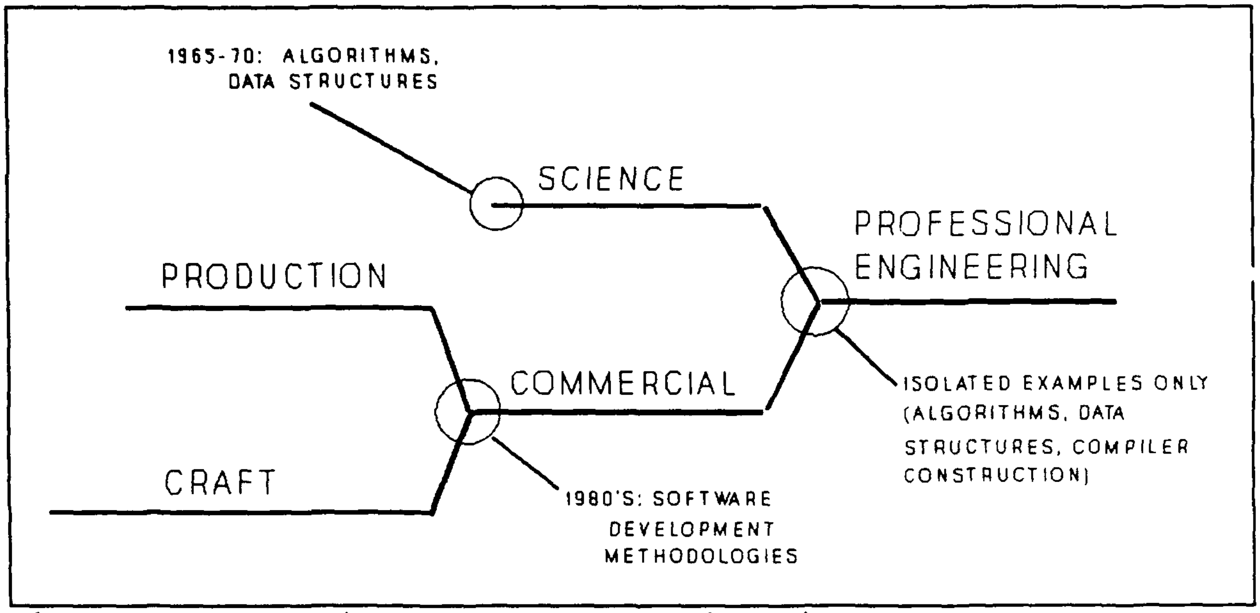


Figure 5: Evolution of Software Engineering

that software engineering is far from being a true engineering discipline because the scientific precepts are not yet mature. Shaw agrees, stating that software engineering is in some cases craft and in some cases commercial practice. But she also claims that science is beginning to contribute and in isolated cases professional engineering is taking place.[19] Another industry expert further explained that the change from art to science has become popularized but that in his opinion most software development is still about 80% artistic.[20]

Elements, Goals, and Principles of Software Engineering

Software engineering consists of elements, goals, and

[19]ibid, p 16.

[20]Discussion with BGen Dennis Brown (USAF, Ret), Vice President, Information Systems Group, Martin Marietta Corporation.

principles that must be used throughout the software life cycle if the end game of reliable, maintainable systems is to be realized. Each of the elements of software engineering as shown in Figure 6 must be in balance.[21] Too often, according to many of those interviewed, the primary focus is on picking or developing tools before the methodology had been sorted out when ideally the reverse should be true.

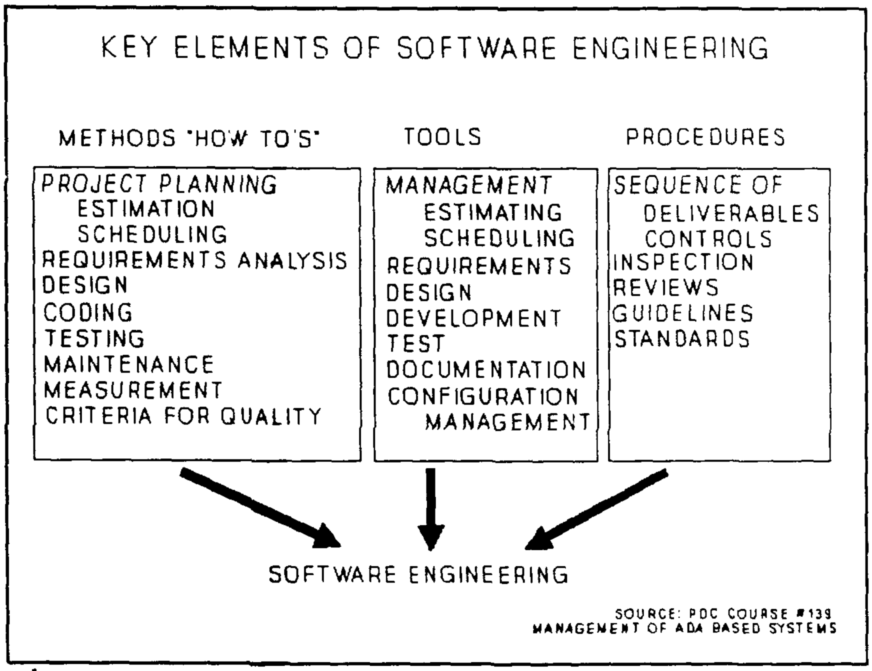


Figure 6

Reliability, efficiency, modifiability, and understandability are the goals of software engineering. Reliability means that software will perform correctly and consistently for all input data under all circumstances. If external problems occur, reliable software will detect the problem and degrade gracefully. Efficient software will optimize the use of time and space. Response will occur within the required time limits and the software fits within memory

[21]Adopted from a lecture by Eileen Quann in AFCEA Professional Development Center Course #139, "Management of Ada Based Systems."

constraints. Software needs to be as efficient as required to do the job, not as efficient as possible. Modifiability is the ability to make changes that have localized or controlled effect. High modifiability is the result of a consistent, well organized design and controlled complexity. It results in systems that are easier to integrate and test, easier to maintain, and easier to enhance. Understandability relates to the clarity and logic of the software that comes from consistent techniques for design and documentation. Understandability results in systems that are easier to code, easier to verify, and easier to maintain.

Software engineering goals are achieved through a set of principles which contribute to the effective production of software. These principles include: abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability. Definitions of each can be found in Appendix 2.

The Importance of Requirements Definition

The application of these principles focuses attention on requirements definition as a fundamental aspect of development. In the three basic phases of the software life cycle; definition, development, and maintenance, the first, or definition phase, often determines the success and cost of the next two. One study showed that sixty to eighty percent of the problems in software systems' design were attributed to inaccurate requirements definitions.[22] The importance of requirements definition as it translates into system design was summarized by one industry executive in Figure 7.[23] A commonly held view in industry is

GOOD SYSTEM DESIGN + GOOD PROGRAMMING = GREAT PRODUCT

GOOD SYSTEM DESIGN + BAD PROGRAMMING = GOOD PRODUCT

POOR SYSTEM DESIGN + GOOD PROGRAMMING = BAD PRODUCT

POOR SYSTEM DESIGN + BAD PROGRAMMING = TERRIBLE PRODUCT

Figure 7: Importance of Requirements Definition

that too often solutions are attempted when the problems are not understood.

Almost universally, an iterative design process is necessary to crystalize what the true system requirements are. Iterative design, which includes techniques of rapid prototyping and evolutionary acquisition, provides a user the opportunity to see and use versions of the final product and make changes as the system evolves throughout its development. The Spiral Life Cycle Model created by Barry Boehm is a popular example of this

---

[22] B. H. Boar, Application Prototyping: A Requirements Definition Strategy for the 80's, 1984

[23] The executive, Bob Rowe from the Boeing Aerospace Corporation, said he keeps this chart on his wall, author unknown, to constantly remind himself and anyone who walks in the importance of requirements analysis and up-front systems engineering.

approach (Figure 8).[24] Several industry and government

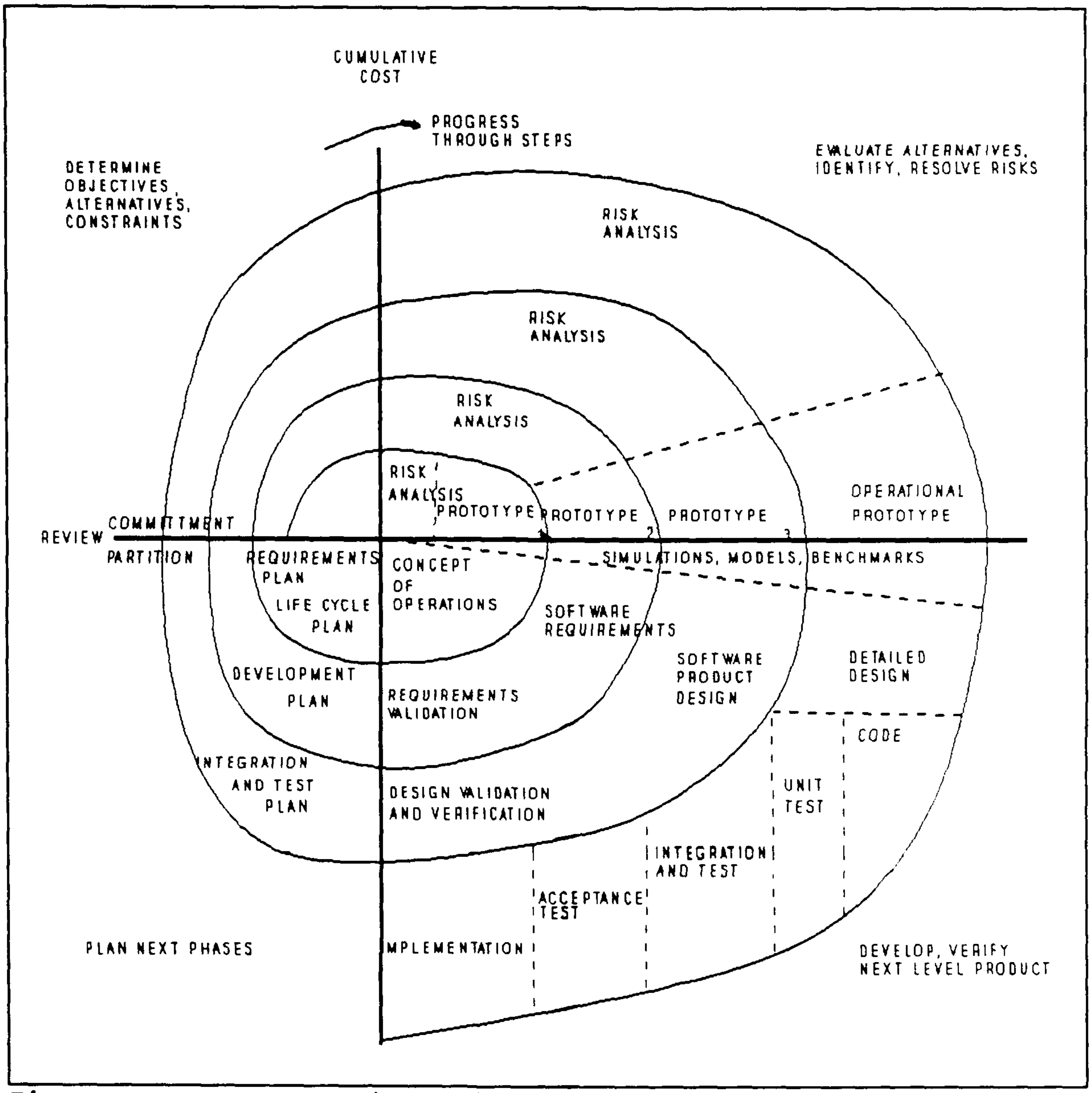


Figure 8: Boehm's Spiral Life Cycle Model

sponsored studies have concluded that an evolutionary approach to both the requirements definition phase and the acquisition phase

---

[24]Barry Boehm, "A Spiral Model of Software Development and Enhancement", IEEE <u>Computer</u>, May 1988, p 61-72.

provides a greater probability of fielding a useful command and control capability sooner and with a higher degree of user satisfaction than if procured via a traditional (waterfall) approach[25]. Many of those interviewed stated that iterative development of requirements, testing of those requirements, and the construction of systems by incremental development have only recently begun to see wide spread acceptance[26].

Good software engineering and requirements definition are independent of the programming language used. However, one language, the Ada programming language, was specifically designed around software engineering constructs. As such Ada forces more effort to be expended in the requirements definition and design phases of a program, as will be seen in the next chapter.

[25]J. H. Garner, LCDR, Evolutionary Acquisition Revisited, A C2 Industry Study, Manuscript, AFCEA, January 15, 1991.

[26]Defense Science Board/Office of the Under Secretary of Defense for Acquisition, Report of the Defense Science Board Task Force on Military Software, September 1987, p 33.

Chapter 4
ADA

"Now the whole earth used only one language, with few words.... Then they said, "Come, let us build ourselves a city with a tower whose top shall reach the heavens (thus making a name for ourselves), so that we may not be scattered all over the earth. "The Lord said, 'They are just one people, and they all have the same language. If this is what they can do as a beginning, then nothing that they resolve to do will be impossible for them. Come, let us go down, and there make such a babble of their language that they will not understand one another's speech.' Thus the Lord dispersed them from there all over the earth, so that they had to stop building the city."

Genesis 11:1-8

The first thing one learns when studying Ada is that for its supporters and detractors alike, arguments for its use take on almost religious overtones. It is not that Ada is viewed as a good or bad technology, for it is almost universally agreed that Ada is a powerful programming language whose constructs of reusability, portability, information hiding, and strong typing superbly support the introduction of software engineering for most software development. The questions are whether Ada yields the cost savings it was designed to do, whether it is the "one" programming language for all uses as some proponents argue, and whether DoD should put all of its software eggs in one basket as many feel is happening.

Development of the Ada Programming Language

Ada's development was motivated by the aforementioned software crisis and the move toward larger, more complex systems.

It was developed by the DoD in the late 1970s to reduce software life cycle costs resulting from the proliferation of more than 300 languages[27] that were then in use. The costs of trying to develop and maintain software over a life cycle that could stretch beyond twenty years had become uncontrollable. Not only were a vast variety of tools and expertise required, but also substantial difficulties were encountered trying to move application programs between computer systems.

Ada was built to be reliable, maintainable, and efficient.[28] Reliability means that errors are caught before the program is run (accomplished through compiler and interface checking), that it encourages good programming practices, and that it makes for ease of reuse. Maintainability is primarily due to the ease with which Ada can be read (even though it is harder to write). Finally, efficiency is tied to how quickly a program compiles and runs.

Ada is much more than another programming language, it is a robust and proven technology especially designed for well engineered software systems. The language supports modern software engineering principles, risk reduction, and several development paradigms including functionally oriented and object

---

[27]GAO/IMTEC-89-9, Programming Language Status, Costs and Issues Associated with Defense's Implementation of Ada, March 1989, p 2. Other sources have put this number as high as 450 languages and dialects, of which about half were assembly languages (Understanding the Adoption of Ada: Results of an Industry Survey by M. Carlson and G. N. Smith, SEI-90-SR-10, May 1990, p 25).

[28] From AFCEA Professional Development Center Course #139, Management of Ada Based Systems.

oriented design methodologies. The language has proven to be suitable for many dissimilar application areas, ranging from commercial data processing to Artificial Intelligence (AI).[29]

Ada Is the Law

"Ada," its advocates like to say, "not only makes good programming sense, it's the law." This statement arises from the FY 91 Appropriations Bill requiring that:

> "...after 1 June 1991, where cost effective, all Department of Defense software shall be written in the programming language of Ada, in the absence of a special exemption by an official designated by the Secretary of Defense[30]."

In drafting the language, the House Appropriations Committee cited a number of reasons why enforcing the Ada mandate was important. These included: training economies of scale arising from a common language, Ada's constructs as building blocks for disciplined software engineering, its internal checking which inhibits errors in large systems, and its design which facilitates and encourages reuse of already built and tested program parts. However, the fundamental reason for this legislation was Ada's encouragement of software engineering .... the application of engineering discipline being seen as the only currently feasible way to control software cost escalation in

---

[29]J. L. Diaz-Herrera, Artificial Intelligence and Ada, March 1991, p 13.

[30]FY 91 Appropriations Bill, Section 8092

ever larger and more complex systems.[31]

The legislation also reflected the Appropriations Committee's views that failure to fully incorporate Ada had become a weak link in fielding new systems and that DoD's mandate was not being enforced effectively. This law was intended to improve and accelerate the use of Ada and add teeth to existing DoD policy. In fact, the law went beyond the original mandate and forces Ada to be used (or be specifically exempted for use) in general purpose computer applications as well as those in weapon systems.

The caveat of cost effectiveness in the Congressional language, which some view as a loophole around the need to use Ada, was inserted because it was felt that not all applications can be done effectively in Ada. How cost effectiveness will be defined has become an issue that will have to be clarified by new DoD policy. This policy, by its very nature, will send a signal to industry just how serious DoD is about keeping Ada for the long term.

The DoD Mandate for Ada

The requirement to use Ada for DoD software existed long before this recent legislation. Ada became a DoD standard in

---

[31]Appropriations Committee Report to the FY 91 Appropriations Bill, p 45.

1980. Two directives published in 1987[32] established Ada as the common programming language for Defense computer resources used in intelligence systems, for the command and control of military forces, or integral to weapon systems. Other languages were authorized only where DoD did not have to develop or maintain the software for the life of the system (Commercial Off The Shelf software, COTS) or if a waiver could be obtained for reasons of cost effectiveness or performance. This mandate turned out to be less effective than originally hoped, for a number of reasons.

Difficulties Bringing Ada On Board

The perceived and real problems of introducing Ada have not been insignificant. The cost of new compilers and other tools necessary for software development in Ada had to be borne by either the contractor or the program office. Both viewed themselves as under pressure to cut costs for new programs in an era of declining budgets and fixed price contracts. Compilers and tools did not exist for several of the computer systems developers either wanted or were directed to use, resulting in delays and increased costs while these were developed. Some existing compilers and tools were viewed with suspicion because

[32]DoDD 3405.1, Computer Programming Language Policy, April 2, 1987, and DoDD 3405.2, Use of Ada(tm) in Weapon Systems, March 30, 1987.

of early claims of performance that proved not to be true.[33]

Equally significant, the language was considered complex and hard to learn, and there were few experienced programmers in Ada. Because universities had yet to treat Ada as little more than an elective for graduate level computer scientists, industry had to invest in training and pay the high costs for the initial climb up the learning curve. These factors, coupled with either weak or no implementation guidance from the Services, gave the impression that waivers were granted routinely or in some cases that the requirement to use Ada had been set aside.

Renewed Commitment by DoD

In 1990, the Services strengthened their commitment to Ada. The Air Force and the Army both issued amplifying policies that reaffirmed Ada as the standard programming language for software development and established the approval authority for waivers at the service headquarters level: in the Air Force the Deputy Assistant Secretary (Communications, Computers and Logistics) and in the Army the Director of Information Systems for Command,

---

[33]The September 1987 Report of the Defense Science Board on Military Software noted that Ada's complexity contributed to slow maturation of its compilers and tools. As a result, Ada compilers executed slowly in comparison to those for other languages. This was viewed as a result of the compilers doing more checking and was treated as something that should improve as engineering refinements occur. The report also noted that the code generated by Ada compilers was not yet highly optimized but that there were no technical obstacles to achieving optimized code for applications written in Ada.

Control, Communications, and Computers. The Navy's policy requiring Ada was strengthened in September, 1990, when the Assistant Secretary of the Navy for Research, Development and Acquisition became the approval authority for all Ada waivers. The effect was to toughen the process through which waivers are granted and, in general, make waivers more difficult to obtain.

Other Recent Changes

Other changes have evolved as well. Many of the early to mid 80s' problems in execution speed, large memory requirements, and long compilation times associated with Ada occurred because of the relative immaturity of Ada compilers. Over the past five years Ada compilers and tools have improved and now have been proven for virtually all commonly used developing environments (Figure 9). As Ada use has increased, many of the problems associated with early Ada performance have been corrected as matters of course.

Use of Ada outside of the DoD has also demonstrated the language's versatility. The FAA and NASA have selected Ada for major projects and have written polices for the use of the language similar to DoD's. The FAA is using Ada for development of its Advanced Automation System which is to upgrade the national airspace management system. This program alone will develop over two million lines of Ada code. Similarly, NASA's space station program is being done in Ada.

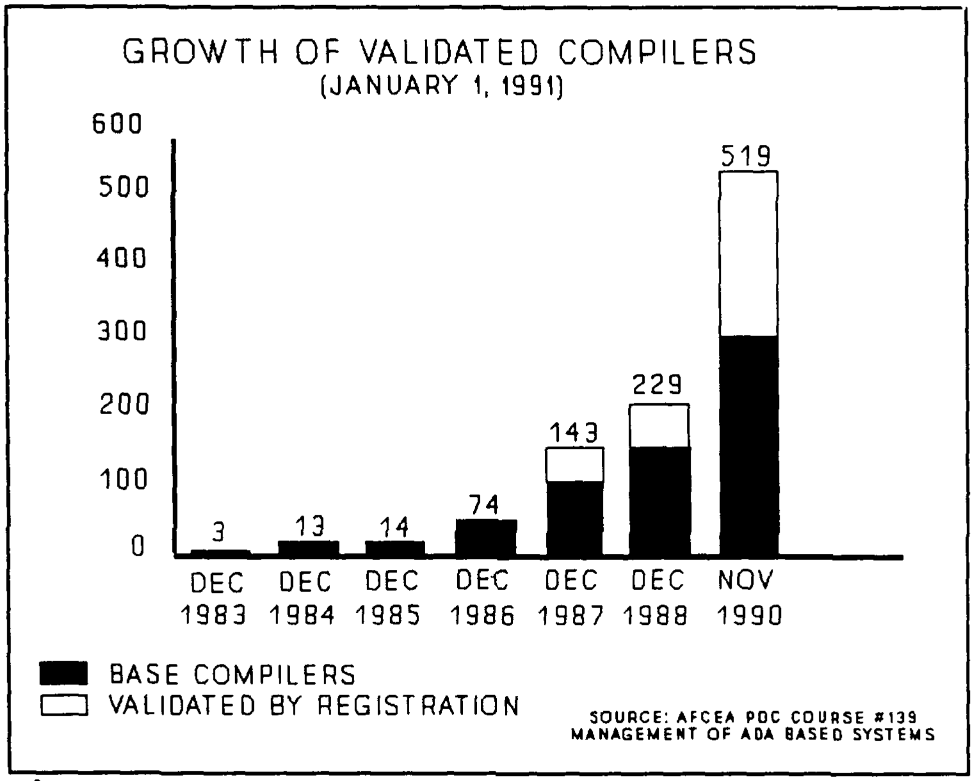


Figure 9

Industry Acceptance

Privately, many companies report that the prospect of fewer and fewer waivers, the Defense Appropriations Act language, non-DoD and commercial growth in support of Ada, and new efforts focused at applying total quality management to software development have forced rethinking of their attitudes regarding Ada. Many who were experimenting with Ada either commercially or as part of independent research and development felt that this experience was important to making or keeping them competitive in the defense market. An equally important factor for some is the extensive use of Ada overseas, particularly in Europe[34]. One company mentioned that it viewed its admittedly small Ada experienced staff as a particularly valuable resource since it should enhance their ability to compete or team with other companies in the overseas market.

The impact of these changes has affected the perceptions of many in industry about the future of Ada. Many companies have initiated Ada projects on their own because of an advocate in the company or as a result of a DoD project. Several cases exist where projects were developed in both Ada and another more traditional language for comparison. Motorola, for example, built a test system for cellular telephone switches in Ada that was equivalent to a previous system developed in C language. Motorola found dramatic improvements in both quality and productivity which they attributed to Ada's facilitation of software engineering principles. Other companies have said that the supportability and maintainability of Ada offers advantages that they have not found with other languages. A study of commercial uses of Ada by the AJPO[35] found that integration and testing was significantly reduced, error rates were lower, the required development resources were lower, and the reuse of

---

[34]Ada has gained widespread acceptance in Europe. The reasons are attributed to the high interest in advanced technology, the proportionally greater number of program new starts, and a greater willingness to use new programming languages. Additionally, NATO has mandated the use of Ada and there is no waiver process.

[35]The Ada Joint Program Office, Tracking the Use of Ada in Commercial Applications; Case Studies and Summary Report, 9 January 1988.

existing code was improved. These and other results dispelled the fear of Ada that many have held.

According to a study by Focused Ada Research, 61 million source lines of code have been developed or are being developed. The market in Ada now approaches one and one-half billion dollars annually[36].

Where is Ada Going?

Not surprisingly, apprehension remains that the DoD commitment to Ada is short lived. Many in government and industry have expressed concern that the impact on program costs and schedules of a forced wholesale transition to Ada will be too great. The advent of newer fourth generation and object oriented programming languages also create the view that Ada is being left behind as technology pushes ahead with newer ideas and methods. It needs to be pointed out, however, that while Ada lacks several of the features necessary to be a object oriented programming language, it superbly supports object oriented design[37]. Other

---

[36]From a briefing by Don Reifer, Reifer Consultants Inc., at the Tri-Ada 90 conference in Baltimore, Md., December 3-7, 1990. Other sources, including a market research report by the investment firm of Branch Cabell and Company in Richmond, Virginia, put the Ada market in the US in 1989 around $1B. The Branch Cabell report estimated that the Ada market could grow to between $2.4B and $9B by 1995 depending on how actively vendors pursue commercial markets with Ada technology. They predict a 20% annual compounded growth rate despite present and anticipated cuts in the Defense budget.

[37]An excellent discussion of Ada and object oriented development is contained in the December 1990 issue of SIGNAL in an article titled "Object-Oriented Development Aids Prototyping and

technical concerns are the limitations of Ada's interfaces to open system components such as x-windows and Structured Query Language (SQL) as well as inadequate handling of decimal arithmetic for Management Information System (MIS) applications. These are viable concerns which the standards community with heavy DoD participation is striving to solve as is the AJPO through its efforts in Ada technology insertion.

Another answer to these technical concerns is a program known as Ada 9x. 9x represents the year in the 1990s that a revision to the Ada standard will take effect. The goal is 1993. The intent of the program is to select those changes that improve the usability of the Ada language while minimizing the disruptive effects of changing the standard[38]. Of primary importance will be to preserve Ada's reliability, safety, and maintainability. A substantial number of technical requirements have been proposed, some have been accepted by a panel of "distinguished reviewers" and others are still being studied. The inclusion of these new requirements in 9x will strengthen Ada's position in the marketplace.

Cultural Influences

The resistance to change is not just technical but largely

Delivery" by Dr. Marilyn Andrulis.

[38]Office of the Under Secretary of Defense for Acquisition, Ada 9x Project Report, Ada 9x Requirements (Draft), December 1990.

cultural. The software development community in this country has evolved primarily around the assembly, FORTRAN, COBOL, and C programming languages, resulting in vast amounts of code and skills that exist in these languages. For example, it is estimated that there are roughly 100 billion[39] source lines of COBOL code in existence and use today. In contrast to Ada, these languages have been in use for 30 years on the average and in many ways reflect the "art and elegance" of software programming (i.e. the ability to get the machine to do what is desired in the least amount of time and space).

By comparison, Ada forces greater structure and requires more time in the design phase of a program as seen in Figure 10.[40] The longer time spent in design can create the impression that more time and money are being spent with less results. This situation is particularly troublesome when design reviews are fixed and adequate time was not allowed for design. However as Figure 10 shows, the longer design phase means less time is usually spent later in coding, integration, and testing.

Another significant reason for the cultural resistance to Ada is its continued lack of acceptance by academic institutions. Universities do not build or maintain large systems but

---

[39]This number is based on estimates provided by several in industry who pointed to the large COBOL installed base and skills as a reason for the slow acceptance of Ada for MIS applications.

[40]From a briefing by Don Reifer, Reifer Consultants Inc., at the Tri-ADA 90 conference in Baltimore, Md., December 3-5, 1990. His briefing summarized conclusions derived in an analysis of 146 Ada projects from the US, Europe, and Asia.

concentrate primarily on small, single use, throwaway software packages. Universities also typically have limited funds for computers,

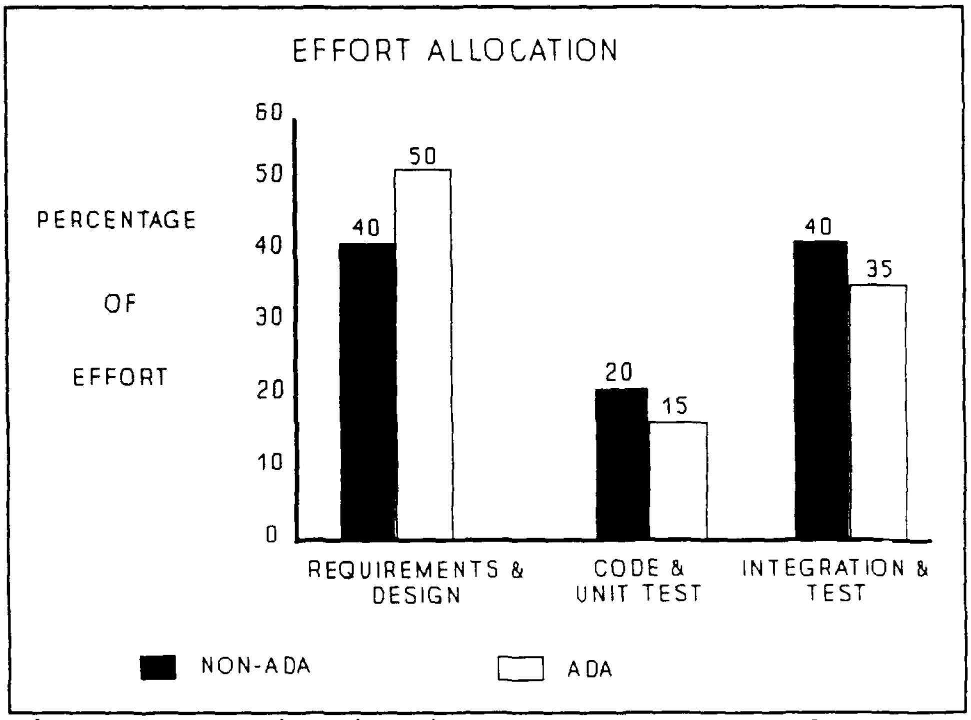


Figure 10: Distribution of Resources Related to Effort

and costs to transition to Ada have traditionally been higher than for development environments for other, more widely used languages. Student demand for Ada courses has been small and faculties generally are not familiar with Ada. Faculties teach only what they know and use. Ada is considered a "DoD language," which in some sectors adds to its unpopularity. Most importantly, industry has not demanded Ada skills. In engineering and applied skills, colleges do not teach what they perceive industry does not support. In turn, industry depends on what universities teach to ease the up front investment and transition costs. Many in industry feel Ada can not achieve wide spread acceptance until it is taught widely in college.

The Future

Two statements sum up much of what has been said. The first, voiced by a senior Air Force official is that "Ada is the military standard for software development .... software engineering principles and discipline are important, but industry must employ Ada if they are to satisfy DoD needs[41]." The second, from an industry representative, is that Ada hype has painted Ada as fitting the DoD/Aerospace niche rather than the high quality niche it deserves[42].

These views imply Ada use is growing and will continue to grow at a steady rate as a result of forces in both the government and commercial sectors. Nevertheless, it is clear Ada has not yet reached the critical mass necessary to assure the acceptance it deserves. New compilers and tools, increased emphasis on software engineering, quality and reliability in software development, and the new standards created by the 9x community are the hope for the future.

In a popular article on software engineering entitled "No Silver Bullet," Fred Brooks says of Ada,

> "I predict that a decade from now, when the effectiveness of Ada is assessed, it will be seen to have made a substantial difference, but not because of any language feature, not indeed because of all of them combined. Neither will the new Ada environments prove to be the cause of the improvements. Ada's greatest contribution will be that

---

[41]Discussion with Maj Gen Albert J. Edmonds, Assistant Chief of Staff Command, Control, Communications, and Computers, Hq USAF, on November 26, 1990.

[42]Discussion with Mr. Douglas W. Waugh, Software Technology, Federal Sector Division, IBM, on February 14, 1991.

switching to it occasioned training programmers in modern software design techniques."[43]

[43]Frederick P. Brooks, Jr., "No Silver Bullet, Essence and Accidents of Software Engineering," IEEE Computer, April 1987, p 14.

Chapter 5
PROCESS MATURITY

"Quality comes not from inspection, but from improvement of the process."

W. Edwards Deming[44]

One problem that has hounded the software development community has been how to predict with any reasonable certainty the outcome of a project. As has been shown, this is vitally important to the DoD given the investment and oversight associated with software projects. Obviously, the tenants of software engineering are a major force in reducing risk and improving the predictability of software, and Ada is a powerful tool that encourages and facilitates software engineering. However, as most of those interviewed expressed, there is no easy solution. Efforts within a company directed toward improving quality and productivity were most frequently cited as how predictability was being achieved and improved. Generally speaking, these efforts were an outgrowth of broader programs within those companies to introduce Total Quality Management (TQM).

Concerned with this issue, the Air Force approached the SEI in 1987 to develop a method to evaluate contractor proposals for software development. Based on work by Watts Humphrey and others, the SEI published a means to characterize the capability

---

[44] Mary Walton, The Deming Management Method, 1986

of software development organizations.[45] The result was a software process maturity framework that provides the DoD and software organizations a way to assess their own capabilities and identify the most important areas for improvement. This framework is based on the software process and the principle of software process management.

Software Process and Process Management

The software process focuses on the idea that 1) the process of producing and evolving software can be defined, managed, measured, and improved, and 2) the quality of software is largely governed by the quality of the process to create and maintain it. It considers the relationships of the required tasks, the tools and methods, and the skills, training, and motivation of the people involved.[46]

Software process management applies process engineering concepts, techniques, and practices to monitor, improve, and control the software process.[47] Software process management assumes that the development process is under statistical control (meaning that if work is repeated in roughly the same manner it

---

[45]Watts S. Humphrey, Characterizing the Software Process: A Maturity Framework, CMU/SEI-87-TR-11, June 1987.

[46]Watts S. Humphrey, David H. Kitson, Tim C. Kasse, The State of Software Engineering Practice: A Preliminary Report, CMU/SEI-89-TR-1, February 1989, p 5.

[47]ibid.

will produce approximately the same result). If it is, then better results depend on improving the process. If it is not, then no progress is possible until statistical control is achieved.[48]

These views of the software process and process management led to the development of a process maturity model and an measurement methodology. Software process maturity is measured though a questionnaire and interviews that determine where a given organization's process resides on the model. The measurement is in the form of an assessment or an evaluation depending on the context. Each is explained in detail later. Another essential part of this structure is a management system for actually implementing the priority actions needed to improve the organization.[49]

The SEI Software Process Maturity Model

The software process is defined by SEI's process maturity model. Five levels exist in the model (Figure 11). Each level reflects a reduction of risk in software development and a corresponding increase in the productivity and quality of the

---

[48]Watts S. Humphrey, Characterizing the Software Process: A Maturity Framework, CMU/SEI-87-TR-11, June 1987, p 1.

[49]ibid., p 2.

Level	Characteristic	Key Challenges	Results
5 Optimizing	Improvement fed back into process	Still human intensive process Maintain organization at optimizing level	Productivity & Quality
4 Managed	(Quantitative) Measured process	Changing technology Problem analysis Problem perspective	
3 Defined	(Qualitative) Process defined and institutionalized	Process measurement Process analysis Quantitative quality plans	
2 Repeatable	(Intuitive) Process dependent on individuals	Training Technical practices - reviews, testing Process focus - standards, process groups	
1 Initial	(Ad hoc/chaotic)	Project management Project planning Configuration management Software quality assurance	Risk

Figure 11: SEI Software Process Maturity Model

outcome for any given project.[50] The result is a commensurate improvement in the predictability of that outcome as well.

Level 1 (Initial) represents a software process that is generally labeled as ad hoc or chaotic and embodies an unpredictable and poorly controlled development environment. Usually, it is the entry level for most organizations. Formal project controls may or may not exist, but if they do the

[50]The discussion of the process maturity model is adapted from a seminar by Judah Mogilenski, Contel Federal Systems, at the AFCEA 1991 Military/Government Computing Conference and two technical reports by the SEI: The State of Engineering Practice: A Preliminary Report, by W. S. Humphrey, D. H. Kitson, and T. C. Kasse (CMU/SEI-89-TR-1, February 1989) and Characterizing the Software Process: A Maturity Framework, by W. S. Humphrey (CMU/SEI-87-TR-11, June 1987).

management mechanisms to insure controls are followed are lacking. Projects at this level are frequently characterized by large cost and schedule overruns. Organizations at this level do have projects that succeed, but usually as a result of a dedicated team rather than the capability of the organization.

Organizations that achieve level 2 (Repeatable) can demonstrate basic project controls of project management, management oversight, product assurance, and change control. These organizations have a high degree of repeatability to similar past projects but face high risks when taking on new challenges. The increased risk reflects frequent quality problems and lack of a structure for improvement.

Level 3 (Defined) defines a process in which standards are institutionalized and the process is well documented and formalized. Organizations at this level have the foundation for examining the process and how to improve it. The outcome of projects can be forecast accurately across a broader range of activities. Movement to the defined level depends on the establishment of a Software Engineering Process Group (SEPG) which is a group of software professionals within the organization specifically chartered to focus on software process improvement.

The ability to measure results, set goals for both quality and productivity, and achieve those goals reflects level 4 (Managed). Predictability at this level has improved to a high degree for projects as a whole and for each step along the way.

Level 5 (Optimizing) allows bottlenecks and weaknesses in the process to be identified and used to focus improvement. Ideally, data gathering at this stage is automated. This is a proactive stage where the process is continuously modified based on analysis and the introduction of new technology as it contributes to further process improvement. Level 5 is called "optimizing" instead of "optimized" because the software development process at this level is always changing and evolving.

Measuring Maturity and Process Improvement

The maturity of an organization's software development process is defined in terms of one of the levels in the model. A structured set of yes-no questions is used to determine an organization's maturity level. The questions cover six to ten prior projects. The areas evaluated include organizational and resource management, the software engineering process and its management, and the tools and technology used in the software engineering process. A trained team validates the answers by seeking concrete evidence of yes responses.

Software process assessments and software capability evaluations are the measurement instruments used (Figure 12). Although similar in form, the two are distinguished by the purpose each serves.

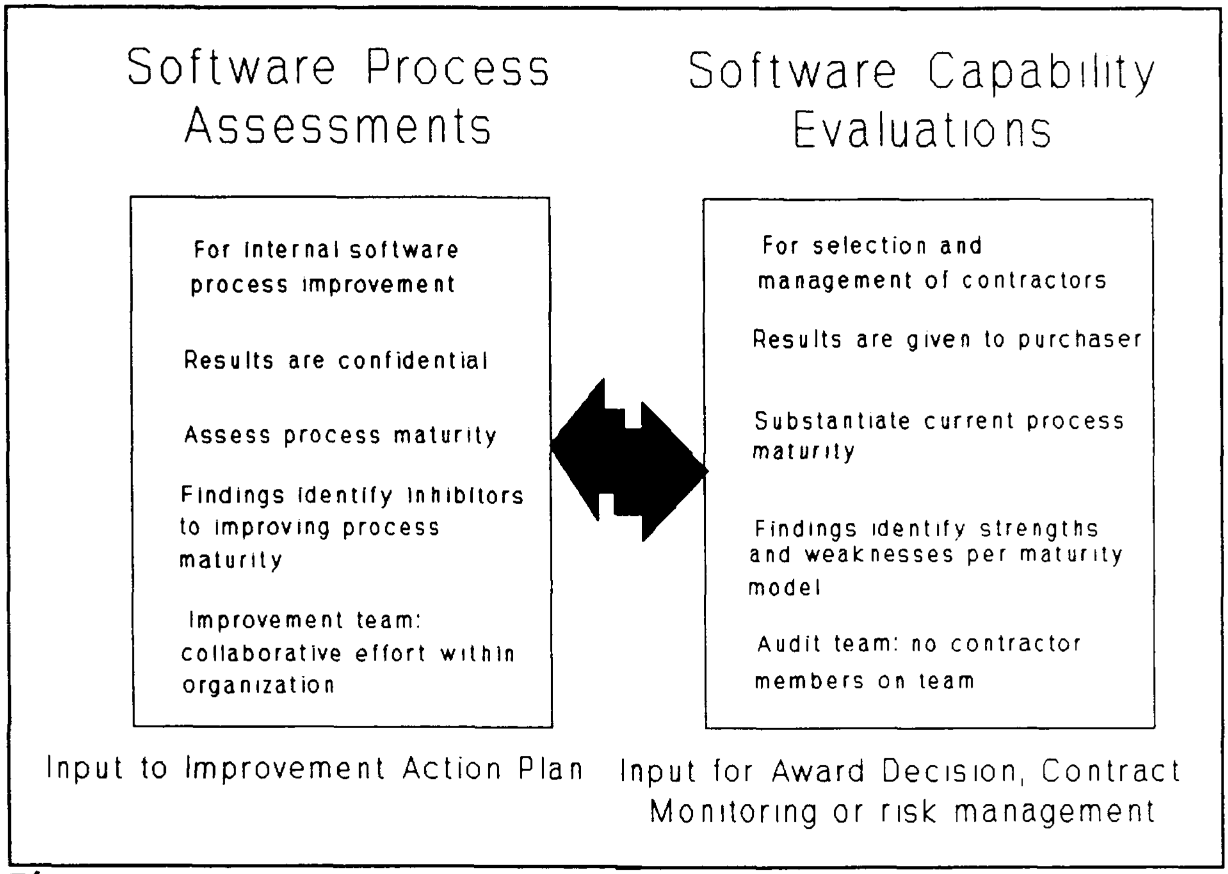


Figure 12

The Software Process Assessment

The software process assessment allows a company or organization to understand its own software engineering practices and identify areas for improvement. It is an in-house appraisal of the organization's current software process done at the request of senior management. The assessment is accomplished by the organization itself, usually assisted by either the SEI or one of nine SEI certified vendors. Two important aspects of the assessment program are key to its success. First, the results are confidential; restricted to the company or organization that requests the assessment. More significant, however, is that the assessment requires the commitment and active participation of senior management. Assessments are exhaustive and costly in terms of time and personnel (four team members must be provided for about seven days to train and complete the assessment plus complete access to all software professionals in the organization must be granted during the two or three days the assessment takes).

Regardless of the extent or quality of the assessment, it accomplishes little unless a company makes the investment to improve the process once the results are known. In this sense, the assessment provides top management with a structured list of strengths and prioritized improvement areas needed to improve its software development capability. The development of a Software Improvement Action Plan and a SEPG to implement the plan are essential elements of this follow-up. The importance attached to this program was voiced by many in industry who stated that assessments are becoming fundamental tools to sustaining their company's position in the DoD software market.

The Software Capability Evaluation

A software capability evaluation is similar to an assessment except that it is externally driven. It is performed by the government to determine the expected productivity, quality, and risk of an upcoming or existing project. It is one of the factors a source selection committee can use in the award of a competitive procurement contract.

Software capability evaluations are starting to be used more widely throughout the DoD; particularly in the award of large, complex projects. Capability evaluations are now required by the Naval Air Development Center for software contracts in excess of one million dollars. The Army's Communications-Electronics Command has drafted a policy requiring that bidders have evaluations if the software for a project is expected to cost more than 10 million dollars. The Air Force's Electronic Systems Division (ESD) is considering a requirement for a process improvement plan for contractors who have been evaluated with a maturity level less than three.[51]

ESD recently advertised a Command Center Omnibus contract in the Commerce Business Daily stating:

> "...the software process capability of the responding contractors will play a major role in the source selection decision. ESD will use the Software Capability Evaluation method developed by the SEI to recognize the current software process capability of responding contractors. Those contractors without a verifiable software process improvement program leading to accomplishment of the 'Level 5' SEI-defined maturity level and an SEI-defined maturity level of at least 'Level 3' will be viewed as high risk in the source selection decision."[52]

Partly because of the potentially widespread use of the capability evaluations, the SEI is currently making some minor revisions to the maturity model and the measurement methodology

---

[51]From a briefing at the SEI on their software process program, SEI Visitors Day, February 21,1991.

[52]Commerce Business Daily, January 3, 1991, p 2.

to improve performance and risk prediction. The revised maturity model will identify key issue areas and key practices required at each maturity level. A revised questionnaire will sample the key practices and provide areas of focus for the assessment/evaluation teams. These changes are expected to be published in the summer or fall of 1991.[53]

Results of Process Measurement

Process measurement provides a good barometer of where industry is today in software development. The results of assessments accomplished through the end of 1989 show that 74% of the organizations assessed fell into level 1, 22% were level 2, and remaining 4% fell into level 3.[54] Only a select few, highly specialized software development organizations approached the requirements necessary for levels 4 or 5. SEI indicates that those companies who have invested in the assessments have shown increases in the maturity level with corresponding improvements in quality and productivity. These results are supported by cost performance data.[55] Most of these companies are defense related

---

[53]Conversation with Dr. William Curtis, SEI, May 23, 1991.

[54]Tutorial: Improving the Software Process, Tutorial Proceedings, AFCEA 1991 Military/Government Computing Conference and Exposition, Section 2, slide 23.

[55]From a discussion with Mark Paulk of the SEI. The SEI collects data from assessments on a continual basis. However, this data is highly confidential and is not releasable except in the very generalized, aggregated sense that is reflected here.

but more and more commercial software developers have recognized the value of this methodology and have begun to accept it as a way to gain an advantage in a highly competitive market.

Cultural Factors in Process Improvement

The most dramatic and difficult changes in process improvement are again cultural rather than technical. The concept of process improvement is abstract, not something for which a classic return on investment calculation is readily available. Furthermore, process improvement competes for resources with other "tangible" mission critical functions. Process improvement and the resulting impact on quality and reliability are totally a function of commitment from the top down. To be effective the commitment must become an ongoing cycle of improvement and reassessment (Figure 13). An action plan, continued investment, and a long term,

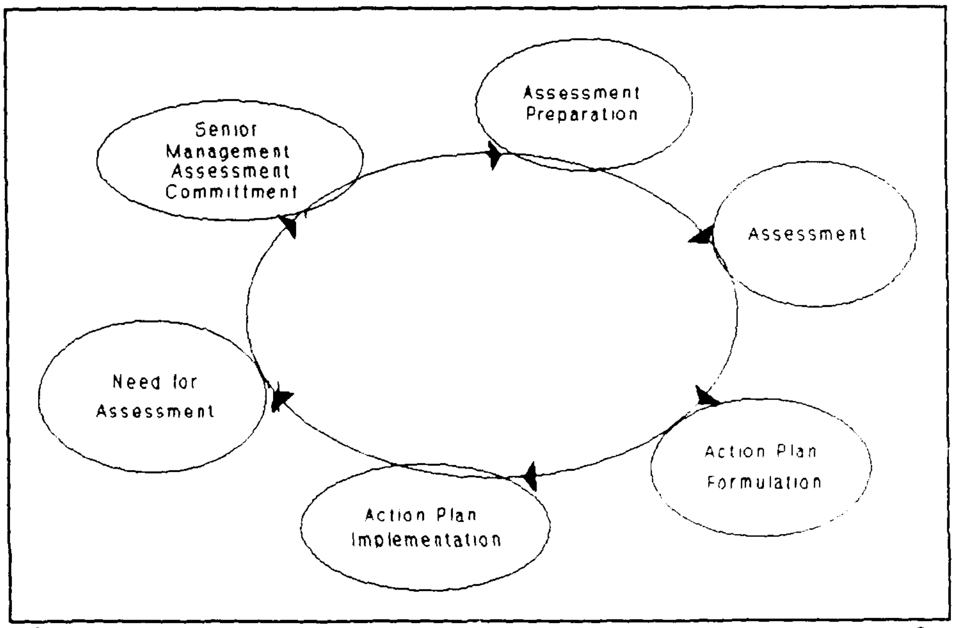


Figure 13: The Software Process Improvement Cycle

constantly evolving strategy are necessary to reach and sustain the top level of process maturity.

Chapter 6
SUMMARY/OBSERVATIONS

The software industry is a strategic industry essential to the national defense. The national security strategy of the US relies on technological superiority and improvements in the reliability and quality of fielded software are vital elements in maintaining that superiority. Efforts to reduce software costs through increased maintainability, reuse, and portability are equally vital as software's importance continues to grow. Software engineering, Ada, and process maturity are the mechanisms to achieve these goals.[56]

A New Era

This study provides the impression that the software industry is beginning a new era. The problems of the past have been recognized and there is widespread movement toward strengthening productivity and quality in software development. Although there is a long way to go, the results of process measurements show the trend is improving. Not surprisingly, the views of those in industry who are doing software development day to day coincides with the more academic views found in the many articles and technical journals that document the problems and

---

[56]The recently released 1992 Critical Technologies Plan for DoD includes software engineering in the list of 21 technologies deemed essential for maintaining the superiority of US weapon systems (Aviation Week and Space Technology, May 20, 1991, p 57).

new technologies. The programs initiated by the Ada and STARS Joint Program Offices and the SEI have been a significant reason software engineering in particular has advanced as far as it has. Most of those interviewed agree that the most dramatic changes have occurred in the past two to five years.[57]

The study also revealed diverse opinions among software professionals but two themes were predominant.

Education in Software Engineering and Ada is Key

Formal education in software engineering and Ada, particularly at the undergraduate level, is key to their widespread acceptance. As previously stated, much of the resistance to Ada comes from its limited use by colleges which is in turn linked to a perceived lack of demand by industry. Similar comments were made about the development of software engineering discipline.

Today, based on the comments of many who were interviewed, much software engineering is ad hoc rather than a managed discipline. The SEI has taken the lead in countering this trend

[57]Mary Shaw relates that it takes about twenty years to introduce new technology into widespread use and advancements in software development seem to have followed that trend (Beyond Programming-in-the-Large: The Next Challenges for Software Engineering, SEI-86-TM-6, May 1986, p 8). Software engineering is recognized to be about twenty years old, Ada recently celebrated its tenth birthday, and although the concept of software process maturity is relatively new, its foundations lie in the TQM management philosophy which dates back to the 1950s with Demming's work.

by promoting software engineering education. The SEI has developed a curriculum in software engineering and a transition strategy that is aimed at both industry and academia.[58] Over the past two years, 27 universities, including the Air Force Institute of Technology, have adopted all or part of the SEI software engineering program. However, this work is currently focused at the graduate level. Industry comments reflect that software engineering practices will not become commonplace skills until it is a fundamental part of an undergraduate computer science degree. The Florida Institute of Technology (FIT) is one of several schools that is aggressively working to develop such a program. However, even with widespread acceptance outside of FIT by both industry and other universities within the state, it may take another seven years before FIT can graduate a person with an undergraduate engineering degree in software engineering.[59]

Continued support for efforts to push education for software engineering is essential to shift general focus away from the view that software development is merely coding and toward

---

[58]SEI's transition strategy is key to its professional development program in software engineering. It requires creating "transition agents" from academia and executives and middle managers in industry and government. Their is to disseminate the products SEI creates (courses, videotapes, textbooks, workshops, etc.) to advance software engineering education tailored to meet organizational needs.

[59]Dr. Charles Engle, FIT, says that many problems remain. For one thing industry needs to specify what they really expect from a software engineer. One fundamental problem industry faces is that computer science graduates have little understanding of the software life cycle beyond coding. Another problem is that many of the underlying math and engineering principles for software engineers are not yet fully developed.

increasing the collective understanding of the entire software life cycle. This attitude in turn highlights the need for quality and rigor in the software development process--the very foundation of the argument for Ada.

Using Ada as a core language from which to teach software engineering is logical since Ada was built around software engineering concepts such as modularity, abstraction, and data typing. Both the SEI and FIT programs use Ada as a core language to teach software engineering principles. Most important, Ada is optimized for "programming in the large", i.e. it was designed to develop large systems. One focus of the software engineering discipline is that large software systems must be developed in fundamentally different ways than small systems. That same recognition must be created in education of future engineers and computer scientists.

Debate Over Ada Continues

In spite of the mandates, Congressional action, and successes, strong resistance to Ada remains. There are a wide variety of reasons both technical and cultural. Most of these issues have already been discussed, as have the efforts to overcome them. The costs of training, investment in programming environments, and the continued belief that DoD will grant exceptions are the most apparent. The argument frequently voiced was that DoD needs to be focused on the software engineering process and not on the programming language which is largely independent of the process.

Overwhelmingly, the forces promoting Ada are its use in federal agencies outside the DoD and its increasing commercial use, particularly overseas. One opinion expressed by a top manager in GE Aerospace is that to be truly effective, Ada should be mandated throughout the Federal government. Her view was that the only technically valid reason for not using Ada is if the required development or operational environment is not well supported by Ada.[60] This type of reaction was commonly held by those who had converted to Ada for one reason or another. Ada 9x, more than any single factor, was seen as determining Ada's viability over the next several years. The need to openly embrace object oriented programming and bindings to SQL and x-windows were the two areas most frequently mentioned as essential to Ada's future.

Ada's enforcement of software engineering principles was also seen as a major factor improving productivity. Over and over again the success of many software projects was attributed to a small group of bright, highly talented programmers as the core element of the project. Frequently, such a group was preferred to a full staff even for large projects of a hundred thousand lines of code or more. This is not surprising since software construction is the product of a creative mind. Fred

[60]Interview with Cynthia Verbinski, GE Aerospace, January 15, 1991.

Brooks says that the difference between a great and an average approach are an order of magnitude.[61] The problem is that building and sustaining this type of cadre is usually difficult or impossible. From the comments received, good software engineering discipline, and particularly the use of Ada, make average or inexperienced programming staffs better able to produce quality code. The structure and definition which forces more work to be done up front translates into a better understanding of the problem before the detail work is begun. In one example, Harry Doscher of Motorola said that he was able to have a summer student successfully translate and run test plans written in Ada with little difficulty, something he could not do with similar plans written in C.

Conclusion

As programs become bigger and more complex, quality, maintainability, and reliability of the software have become more pronounced in the overall life cycle cost of a system. The principles of software engineering are at the heart of every aspect, economic and managerial as well as technical, of how these factors are achieved. The Ada programming language is a powerful tool for bringing about a permanent change to modern software techniques, but it suffers from an imbedded culture that

---

[61]Frederick P. Brooks, Jr., "No Silver Bullet, Essence and Accidents of Software Engineering," IEEE Computer, Vol 20, April 1987, p 18.

resists much of the rigor that Ada imposes. Process maturity measurement provides a means to define how well software engineering discipline has evolved within a company or organization. The achievement of good software engineering is fundamentally a leadership challenge. Experience shows that to replace "cottage-industry" mentality and procedures at the working level with disciplined, repeatable engineering processes requires commitment, investment, and aggressive leadership.

Appendix 1
QUESTIONS

Over last three years what three languages has you company used most to develop software for systems for government and commercial applications?

Does your firm use Ada in its development?
- How long has it been used?
- Feelings about level of expertise?
- For all development?
- For government contract development only?
- IR&D development?
- For internal, commercial development?
- Does Ada used for internal development conform to DOD-STD 2167A or DOD STD 1815?
- What problems does this standard create?

How much are the availability and cost of adequate Ada tools a factor in your decisions to use Ada or other languages?
- What tools are/were not available to support your particular program...if they can be/could be available would that change your decision, why?

If so, what aspects led to the decision to incorporate?

If C, what was the original motivation to switch to C?

What sets development of telecommunications systems apart from other software development such as weapon systems, MIS, etc.?
- How do these factors play in the decision on which programming language to use?
- How is Ada viewed in this context?

Are you working on or aware of any communications project that entails integrating commercial hardware/software into a larger Air Force or DoD telecommunications system?
- If so, what HOLs are being used in each and to what degree is the integration of interface of the two a problem?
- Is Ada a player or not?

What difficulty is there interfacing systems developed in disparate higher order languages (HOL).......does Ada help or hinder this process?

Even though Ada is not mandated for commercial off the shelf software do you think the government will ultimately require Ada for any software purchase?

What are corporate policies on selection of programming languages, for which development, e.g. commercial, IR&D, government contract?

How much does the concept of software engineering factor into your software development?

If it does is Ada viewed as the best vehicle to promote software engineering or are other languages just as good...which ones?

If not why not?

What are you doing to further software engineering education?

Do you think the Europeans and Japanese have endorsed/incorporated Ada to a greater degree than in the US, if so why?

If so do you see them gaining any advantage in the long term (or short term) as a result......what disadvantages?

Do you see the DoD standing by its commitment to Ada or do you think that the mandate is so much posturing?

To what degree do you think the mandate will change in the next 3-5 years..more stringent or less?

To what degree, if any, does this affect your decisions on which HOL to use for other development?

What is your company's position on standardizing its programming language for all new development?

If there is a desire to standardize what language is preferred and why?

Do you expect this decision might change in the next 3-5 years?

What problems would you anticipate/occurred in transitioning to Ada?

How do you expect to see these problems change in the future?

What languages do you see new graduates trained in coming out of school and how much does this affect your programming language decisions?

If universities put greater emphasis on teaching Ada, particularly if it were to become a core language, what impact would this have?

How is object oriented development (OOD) changing programming language decisions?

Do you see OOD as becoming more significant in development decisions in the future?

What HOLs best support a change toward OOD and why?

How does this affect decisions toward Ada?

What changes do you see in the future that will influence decisions positively or negatively toward greater acceptance/dependence on Ada?

What recommendations would you like to see made to the Air Force that would have the greatest impact on improving the current development climate?

What advantages do you see from use of Ada?
- Portability?
- Software maintenance?
- Software engineering attributes?

What disadvantages would you expect from transitioning to Ada?

What advantages exist with your present language?

How long has your company been using Ada?
- As a design language?
- As a development language?

Does your company conduct Ada training programs?
- Does the program include software engineering concepts and methods?

How much Ada specific equipment does your company have?

Of the cost to transition to Ada what percentage has gone for:
- Training?
- Software?
- Hardware?
- Other?

Has the government subsidized the transition in any way?
- If not what should the government do to push Ada adoption?

Have you developed any Ada tools internally and if so what impact has this had on your development in terms of time and/or cost?
- What are they?

Has adoption of Ada offered any competitive advantage or disadvantage?

Compared to similar systems implemented in other languages, what do you expect to be the effect of the use of Ada on your software sustaining direct labor costs: more, same, less?

What do you expect the use of Ada for the development of a system will have on post-development software support costs: increase, no change, decrease?

Are you aware of the SEI process maturity assessment and software capability evaluation programs?

- If so, how are they used or a factor in you company's software development?
- Have you had an assessment done (either in house or SEI/vendor assisted)?
- What was the reaction to the results?
- Has your company set a maturity level as a goal and why?
- What mechanism is in being to improve your process?
- How do you view the capability evaluation as a factor in source selection?

Appendix 2
SOFTWARE ENGINEERING PRINCIPLES

Abstraction: Manages complexity by extracting the essential information while omitting non-essential details. An abstraction is represented by each level of decomposition. It deals with suppressing irrelevant problem-domain details.

Information Hiding: Makes inaccessible those details which do not affect other parts of the system. It permits access to certain data and operations while preventing access to others that violate our logical view. It deals with encapsulating solution-domain implementation details.

Modularity: Allows purposeful structuring by partitioning the whole into manageable parts. It is achieved by designing components which model physical reality.

Localization: Groups logically related entities and pulls together the closely related data and operations so that they can function more independently.

Uniformity: Assures consistency in notation and level and type of design decomposition. It is achieved by eliminating unnecessary differences and conforming to standards and guidelines.

Completeness: Includes all elements necessary and sufficient to meet requirements. It is supported by full and consistent documentation and is accomplished by mapping the design and code to the requirements.

Confirmability: Accomplished by building in the characteristics necessary to prove that the system meets requirement and is done in a way that aids testing and verification.

BIBLIOGRAPHY

Andrulis, M. D. "Object-Oriented Development Aids Prototyping and Delivery," Signal, Vol. 45, No. 4, December 1990, p 76-78.

Ardis, M. and Ford, G. 1989 SEI Report on Graduate Software Engineering Education. CMU/SEI-89-TR-21, June 1989.

Baker, C. "Army Encourages Software Integration." Defense News, April 29, 1991, p 10.

Boehm, B. "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988, p 61-72.

Boehm, B. Software Engineering Economics. Prentice-Hall, Inc., 1981.

Brooks, F. P., Jr. The Mythical Man Month. Addison Wesley Publishing Co., 1975.

Brooks, F. P., Jr. "No Silver Bullet, Essence and Accidents of Software Engineering," IEEE Computer, Volume 20, April 1987, p 10-19.

Carlson, M. and Smith, C. N. Understanding the Adoption of Ada: Results of an Industry Survey. CMU/SEI-90-TR-10, May 1990.

Carroll, P. B. "Painful Birth: Creating New Software Was Agonizing Task for Mitch Kapor Firm." The Wall Street Journal, May 11, 1990, p A1, A7.

Cobb, R. H. and Mills, H. D. "Engineering Software Under Statistical Quality Control." IEEE Software, November 1990, p 44-54.

Crafts, R. E. " A European Ada Case History--Interview with Bengt Jorgensen." Ada Strategies, Vol. 3, No. 11, November 1989, p 10-13.

Crafts, R. E. "FY 91 Appropriations Bill--Use Ada, Its the Law." Ada Strategies, Vol. 4, No. 11, November 1990, p 1-4.

Crafts, R. E. "Motorola Using Ada for Commercial Communications." Ada Strategies, Vol. 3, No. 7, July 1989, p 8-14.

Curtis, B.; Krasner, H. and Iscoe, N. "A Field Study of the Software Design Process for Large Systems." Communications of the ACM, Vol. 31, No. 11, November 1988, p 1268-1287.

Diaz-Herrera, J. L. _Artificial Intelligence and Ada_. Manuscript, Department of Computer Science, School of Information Technology and Engineering, George Mason University, March 1991.

Doscher, H. "An Ada Case Study in Cellular Telephony Testing Tools." _Proceedings, Ada Europe 90_, June 1990, p 24-35.

Garner, J. H., LCDR. _Evolutionary Acquisition Revisited, A C2 Industry Study_. Manuscript, AFCEA, January 15, 1991.

Goldberg, J. H. "The Pentagon's Software Crisis Jeopardizes Key Weapon Programs." _Armed Forces Journal International_, June 1990, p 60-61.

Hughes, D. "Next Generation Defense Programs Will Increase Use of Ada Language." _Aviation Week and Space Technology_, March 28, 1988, p 60-61.

Humphrey, W. S. _Characterizing the Software Process: A Maturity Framework_. CMU/SEI-87-TR-11, June 1987.

Humphrey, W. S. Kitson D. H, and Kasse T. C., _The State of Software Engineering Practice: A Preliminary Report_. CMU/SEI-89-TR-1, February 1989.

Lewis, T. G., and Oman, P. W. "The Challenges of Software Development." IEEE _Software_, November 1990, p 9-12.

Mills, H. D.; Newman, J. R. and Engle, J. B., Jr. "An Undergraduate Curriculum in Software Engineering." _Proceedings, Software Engineering Education_, SEI Conference 1990, p 24-37.

Musa, J. D. and Everett, W. W. "Software-Reliability Engineering: Technology for the 1990s." IEEE _Software_, November 1990, p 36-43.

Myers, E. D. "What the Countess Didn't Count On." _Datamation_, February 1, 1987, p 32-36.

Quann, E. "10 Mistakes to Avoid When Taking On Ada Projects." _Government Computing News_, June 10, 1988, p 73.

Reed, G. P. "Trends: Ada Use Increasing for MIS." _Ada Strategies_, Vol. 3, No. 10, October 1989, p 7-10.

Reifer, D. J. "The Economies of Ada--Guest Column." _Ada Strategies_, Vol. 4, No. 11, p 5-7.

Richards, E. "Pentagon Finds High-Tech Projects Hard to Manage." _The Washington Post_, December 11, 1990, p A1, A6.

Richards, E. and Reid, T. R. "Mass Production Comes to Software." The Washington Post, December 12, 1990, p A1, A16.

Richards, E. "Society's Demands Push Software to Upper Limits." The Washington Post, December 9, 1990, p A1, A24.

Richards, E. "Writing Software: A Quirky, Labor-Intensive Scramble." The Washington Post, December 10, 1990, p A1, A10.

Schlender, B. R. "How to Break the Software Logjam." Fortune, September 25, 1989, p 100-112.

Siegel, J. A. L., et al. National Software Capacity: Near-Term Study. CMU/SEI-90-TR-12, May 1990.

Shaw, M. Beyond Programming-in-the-Large. SEI-86-TM-6, May 1986.

Shaw, M. Education for the Future of Software Engineering. CMU/SEI-86-TM-5, May 1986.

Shaw, M. "Prospects for an Engineering Discipline of Software." IEEE Software, November 1990, p 15-24.

Shaw, M. Prospects for an Engineering Discipline of Software. CMU/SEI-90-TR-20, September 1990.

Siegel, J.A.L.; Stewman, S.; Konda, S.; Larkey, P.; and Wagner, W.G. National Software Capacity: Near Term Study. CMU/SEI-90-TR-12, May 1990.

Smith, G. E.; Cohen, W. M.; Hefley, W. E.; and Levinthal, D. A. Understanding the Adoption of Ada: A Field Study Report. CMU/SEI-89-TR-28, August 1989.

Smith, G. N. and Carlson, M. Understanding the Adoption of Ada: Results of an Industry Survey. CMU/SEI-90-SR-10, May 1990.

Strassmann, Paul A. The Business Value of Computers. New Canaan, Ct: The Information Economic Press, 1988.

Walton, M. The Deming Management Method. Putnam Publishing Group, 1986.

Uhl, M. "Ada and the Tower of Babel." Forbes, August 27, 1984, p 132.

Wood, W., et al. A Guide to the Assessment of Software Development Methods. CMU/SEI-88-TR-8, April 1988.

The Ada Joint Program Office. Tracking the Use of Ada in Commercial Applications; Case Studies and Summary Report. Washington D. C., January 6, 1988.

Air Force Software Management Group (Hq USAF/SCW). Air Force Software Management Plan. Washington D. C., 20 August 1990.

Air Force Studies Board. Adapting Software Development Policies to Modern Technology. Washington D. C.: National Academy Press, 1989.

The Armed Forces Communications and Electronics Association. 1991 Military/Government Computing Conference and Exposition. Tutorial Proceedings.

The Association for Computing Machinery, Inc. Proceedings, Tri-Ada '90. December 1990.

The Commerce Business Daily, January 3, 1991.

Department of the Air Force. Deputy Assistant Secretary (Communications, Computers, and Logistics). Action Memorandum: Air Force Policy on Programming Languages (with attachment). 7 August 1990.

Department of Defense Directive 3405.1. Computer Programming Language Policy. April 2, 1987.

Department of Defense Directive 3405.2. Use of Ada(tm) in Weapon Systems. March 30, 1987.

Department of Transportation. Federal Aviation Administration. Action Notice: National Airspace System (NAS) Software Procedures. September 11, 1989.

Government Accounting Office. Embedded Computer Technology. GAO/IMTEC-90-34, April 1990.

Government Accounting Office. Programming Language...Status, Costs, and Issues Associated with Defense's Implementation of Ada. GAO/IMTEC-89-9, March 1989.

Goodard Space Flight Center. National Aeronautics and Space Administration. Ada and Software Management in NASA: Assessment and Recommendations. Greenbelt, Md., March 1989.

Office of the Under Secretary of Defense for Acquisition. Ada 9x Project Report, Ada 9x Requirements (Draft). Washington D. C., December 1990.

Office of the Under Secretary of Defense for Acquisition. Defense Science Board. Report of the Defense Science Board Task Force on Military Software. Washington D. C.: Government Printing Office, September 1987.

U. S. Congress. House. Committee on Science, Space, and Technology. Subcommittee on Investigations and Oversight. Bugs in the Program, Problems in Federal Government Computer Software Development and Regulation. Washington D. C.: Government Printing Office, September 1989.

U. S. Congress. House. Committee of Conference. Making Appropriations for the Department of Defense, Conference Report. 101st Cong., 2nd Session, October 24, 1990.