NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE SHARP EVOLUTION:
DEVELOPMENT OF THE SIERRA HOTEL AVIATION REPORTING PROGRAM FROM THE DECK PLATES

by

Christopher L. Williamson

September 2000

Thesis Advisor: Luqi
Second Reader: Oleg Kiselyov

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2000	Master's Thesis

4. TITLE AND SUBTITLE
The Sharp Evolution: Development Of The Sierra Hotel Aviation Reporting Program From The Deck Plates

5. FUNDING NUMBERS
JBC

ARO MA

6. AUTHOR(S)
Williamson, Christopher L.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
JBC, NRC, ARO & NPS

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT

Due to constant changes in the military environment, operations tempo, resource limitations, and leadership directives, the fashion in which the military computes its training and readiness is constantly in flux. Previous readiness calculations were accomplished from simple two-dimensional models of qualifications by dates. With the increase of more sophisticated requirements, a new six-dimensional model of training and readiness was invented to compute and even predict future readiness levels, for aviation as outlined in the Training and Readiness Manuel CNAP INST/CNAL INST 3500 Series.

Due to the complex requirements of the new T & R Manual, a software tool was required to track post-flight data and compute aviation combat readiness. The T & R Manual is revised at irregular intervals by independent type wings, resulting in a constant requirement to re-develop existing readiness models and tracking programs. To fulfill this requirement, a team of Naval Aviators with a combination of software engineering expertise, military operations, and project management experience was created to develop a modular based rapid prototype application.

This thesis will review the unique software development models utilized in rapid military application development, contrasting with existing application development models, and the utilization of non-traditional techniques to meet defense readiness requirements. This thesis will also review other readiness tracking systems to compare and contrast the ability to meet the diverse needs of fleet readiness models through efficient software development.

14. SUBJECT TERMS
Software Engineering, Combat Readiness, Software Management, COTS, Software Evolution Model

15. NUMBER OF PAGES

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

THE SHARP EVOLUTION.
DEVELOPMENT OF THE SIERRA HOTEL AVIATION REPORTING PROGRAM FROM THE DECK PLATES

Christopher L. Williamson
Lieutenant, U. S. Navy
B.S., United States Naval Academy, 1991

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

Author: _____
Christopher L. Williamson

Approved by: _____
Advisor: Luqi

_____
Reader: Oleg Kiselyov

_____
Luqi, Academic Associate
Software Engineering Curriculum

_____
Dan Boger, Chairman
Dean ISO

ABSTRACT

Due to constant changes in the military environment, operations tempo, resource limitations, and leadership directives, the fashion in which the military computes its training and readiness is constantly in flux. Previous readiness calculations were accomplished from simple two-dimensional models of qualifications by dates. With the increase of more sophisticated requirements, a new six-dimensional model of training and readiness was invented to compute and even predict future readiness levels for aviation, as outlined in the Training and Readiness Manuel CNAP INST/CNAL INST (Commander, Naval Air Force, United States Pacific Fleet, Instruction / Commander, Naval Air Force, United States Atlantic Fleet, Instruction) 3500 Series.

Due to the complex requirements of the new T & R Manual, a software tool was required to track post-flight data and compute aviation combat readiness. The T & R Manual is revised at irregular intervals by independent type wings, resulting in a constant requirement to re-develop existing readiness models and tracking programs. To fulfill this requirement, a team of Naval Aviators with a combination of software engineering expertise, military operations, and project management experience was created to develop a modular based rapid prototype application - SHARP.

This thesis will review the unique software development models utilized in SHARP rapid military application development, contrasting with existing development models, and the utilization of non-traditional techniques to meet defense readiness requirements. This thesis will also review other readiness tracking systems to compare and contrast the ability to meet the diverse needs of fleet readiness models through efficient software development.

TABLE OF CONTENTS

TABLE OF FIGURES

ACKNOWLEDGEMENT

I would like to thank my wife Donna for her unselfish support during the pursuit of this education, as well as her support throughout my military career. She has sacrificed her time and held our home together during the many long deployments and travel required in the service of this country. I would also like to thank Dr. Luqi and Dr. Oleg for their patients and encouragement to press on with this study.

Never give up, and never forget where you came from.

I. INTRODUCTION

A. BACKGROUND

The Old-Irish quote, "Necessity is the mother of invention,"[1] has epitomized the military's efforts to develop and build hundreds of tools and solutions to meet the needs of the armed services. Naval Aviation has constantly sought after a unified readiness tool to track all aspects of flight operations. For the last twenty years, this effort has been exemplified by series of legacy systems.[def]

1. Legacy and Past Systems

Over the past two decades, a number of attempts were made in creating software systems capable of archiving post-flight data and then determining aviation readiness based on cumulative operations data. Previous development attempts included systems formally engineered by naval system centers, and those informally developed by squadron operations users. One of the early systems, TRAX, was a PC-based computer tool designed to store, retain, and calculate training and readiness data for aviation units. It facilitated readiness calculations by permitting users to enter qualification data directly into the system as well as download data from the NALCOMIS System. TRAX was designed to work in conjunction with the Squadron T & R Matrices as it was established at the time of system design.[2] TRAX was only capable of managing a limited amount of operations data, based on the interface and inputs from the NALCOMIS System. Data was difficult to extract from the system due to its format, incompatibility to communicate

---

1 Attributed to Jonathan Swift, circa 1667-1745.

2 TRAX Interface, NALCOMIS OMA Legacy System Description Presentation, SPAWAR.

with standard presentation tools, data conversion errors, and most significantly its inability to expand to support the new T & R Matrix.

NALCOMIS, the Naval Aviation Logistics Command Management Information System, was designed to provide an automated Management Information System that provided maintenance, material, and operation managers with data for assignment and management of aircraft and equipment.[3] The NALCOMIS System used the NAVFLIRS (Naval Aircraft Flight Record) format for data entry. The NAVFLIRS format was based on an 80-column card data format for recording post-flight data. NATOPS required that a NAVFLIRS report be recorded for each attempt at a flight of Naval Aircraft.[4]

A second system, CANDE – Computer Aided NAVFLIRS Data Entry, was a software tool designed to allow flight crew to enter and record aviation post-flight data in the NAVFLIRS format, utilizing existing squadron computers. Once entered into a squadron computer, NAVFLIRS data could then be saved to a floppy disk, then transported to an assigned data service facility (DSF) for transmission up-line to an archive for analysis.[5]

A third system, GEEK, was designed by a fellow aviator from the F/A-18 Wing to manage Hornet Operations. GEEK was a Microsoft Access 2.0 based application developed to support the current version of the T & R Matrix. GEEK was designed around the single seat concept of operations, and was inflexible to support wide platform

---

[3] Introduction, Naval Aviation Logistics Command Management Information System Manual, SPAWAR.

[4] OPNAVINST 3710.7R, *"Naval Air Training And Operating Procedures Standardization (NATOPS) Manual"*, Interim Change 27, 09 Sep 1999, Para 10.3.

[5] OPNAVINST 4790.2, Aviation Maintenance Manager's Guide, Section 3.d.5.

needs of Naval Aviation in general. Due to its complexity and development format, when the service member was transferred out of his parent command, GEEK became a stagnate product.

These systems all failed to provide any long-term benefit to Naval Flight Operations due to their inflexible format, lack of complete requirements to support all potential customers, and their basis on antiquated methodologies. None of these systems were easy to use, provided all of the operations data required by aviation operations, or were customizable to meet the needs of various type wings. As the T & R Matrices were edited, the existing programs became obsolete and squadrons were required to revert to manual computation of command readiness. As each system would become unusable, squadrons would initiate private efforts to build their own aviation automation systems. Naval Aviation recognized that a single effort had to be made to generate a unified system to support flight operations or else all flight data would be lost.

2. Resulting System

Taking lessons learned from legacy systems, blending current requirements and implementing more efficient development practices, a team of Naval Aviators combined to build a new aviation automation support tool. This automation tool was to be developed to support the flight operations of all seventeen type model series aircraft. The resulting system would include over 750,000 lines of code in a relational database, with over 25 independent modules, in over 800 code and support files, with tens of thousands of unique procedures, functions, and classes. The system was to be deployed to over 330 operationally deployed aviation squadrons and detachments in the support of thousands of naval aircraft through electronic dissemination methods.

3. History of Readiness Measures

In the days of ancient Rome, Julius Caesar measured the readiness of his empire by the wealth of his soldiers, the number and variety of his weapons, and the mass and speed of his armies that could be mustered to the front lines of combat.[6] Combat readiness of the Roman Empire was rated by the number of legions that Caesar could parse throughout the various states, while still maintaining sufficient reserves to protect the central state of Rome. This simple measure of readiness had been used for centuries - simplistically rating combat readiness by inventory and troop strength.

Since its inception during the American Revolution, and then through its relatively brief history, the United States Navy and, primarily, those of Naval Aviation have followed this same methodology, with the exception of the last thirty years. Shortly after the end of the Vietnam War, lessons learned from that conflict taught us that a more efficient and accurate method for computing readiness was required to maintain Cold War Combat Readiness.[7] During the next twenty years, Naval Aviators computed their readiness by totaling readiness points from a standardizing list of qualifications flown by crewmembers assigned to each command. This concept was modeled and referenced in the Aviation Squadron Training and Readiness Manual. In the Summer of 1995, with the introduction of the newest edition of the Training and Readiness Manual COMNAVAIRPACINST / COMNAVAIRLANTINST 3500 Series Instruction[8], Naval

---

6 Julius Caesar, translated by W. A. McDevitte and W. S. Bohn, *"The Civil Wars"*, Internet Classics Archive http://classics.mit.edu//Caesar/civil.html, 1994.

7 Robert S. McNamara, with Brian Vandemark, *"In Retrospect: The Tragedy And Lessons Of Vietnam"*, Random House, Inc, 1995.

8 COMNAVAIRLANTINST 3500.63C, COMNAVAIRPACINST 3500.67C – *"Squadron Training and Readiness"*, 24 Jul 1995.

Aviation took a turn towards modernizing and rating its readiness on the weighted values of received qualifications against expiration periods. For years, this two-dimensional readiness matrix was tracked by hand using simplistic graphics, charts, or "grease boards" [def]. While inefficient, these tracking methods sufficed the need of computing readiness based on qualification points. One year later, the Naval Audit Service recommended further changes to the matrices to justify operations expenditures (See V.A 1996 Naval Audit Service Report on Naval Aviation). The Training and Readiness model shifted from a simple two-dimensional matrix to a more complex six-dimensional matrix as depicted in Figure I.1.

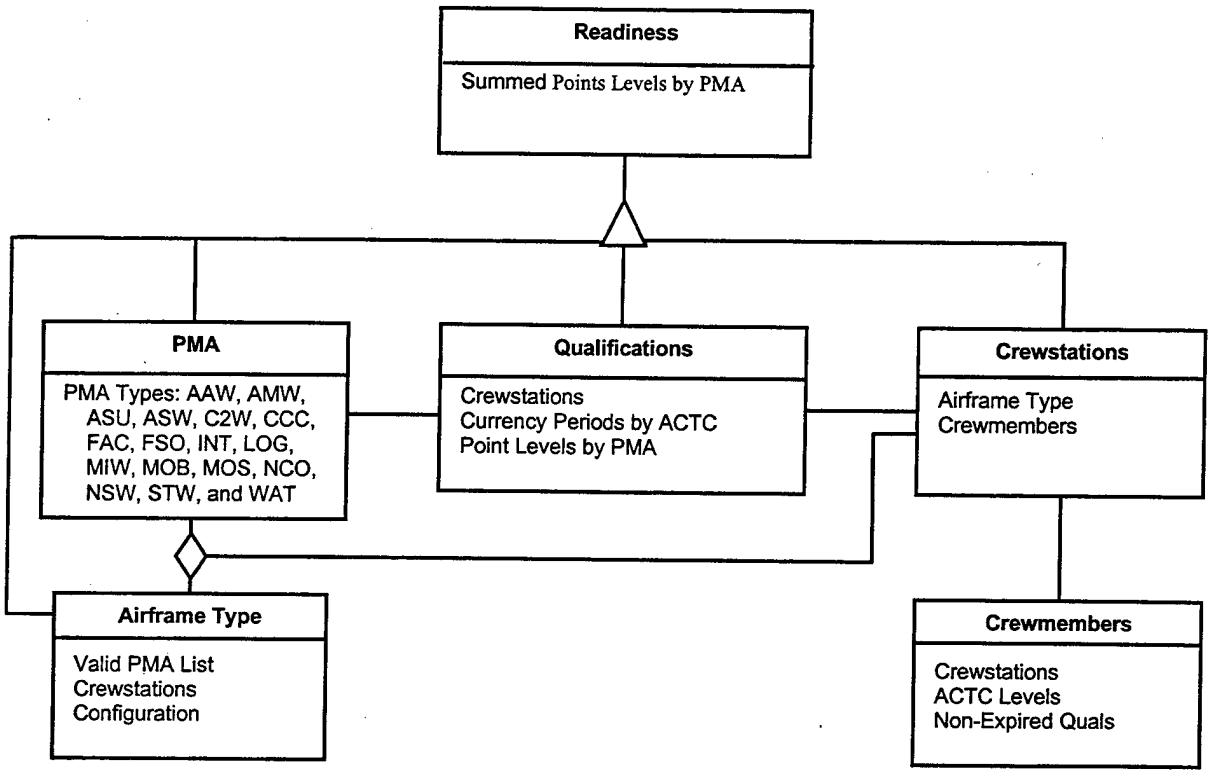


Figure I.1 Object Diagram of the Readiness Model

The newest Readiness Model depicted readiness as a function of the readiness points accumulated across valid Primary Mission Areas (PMA) (PMAs dependent by

Airframe Type). PMAs, as their name states, are the primary mission areas that a particular Type Model Series of aircraft would emphasize in its mission, tasks, and qualifications. There are seventeen unique PMAs accounted for and tracked by Naval Aviation (See V.A PMA List).

A particular Type Model Series might execute anywhere from four to eight different PMAs, based on its missions and aircraft configurations. Each mission is related to only one PMA designation. Each qualification is also related to only one PMA, but can receive qualification points spanned across all of its PMAs. Examples of this relationship can be viewed in the referencing 3500 Series Instructions. Crewmembers gain points by receiving qualifications, valid for their particular crewstation, assigned across PMA categories. The objective of each squadron crewmember is to gain as many points as possible in as many mission areas. The ratio of received points weighted against potential points denoted the crewmember's personal readiness. The combination of crewmember readiness and type wing configuration determined the overall readiness of the command or "T-Level". While this definition may be simplistically stated, the complete model for determining aviation combat readiness would eventually encompass an instruction of over 150 pages. This instruction would be required to reference over seventeen unique type wing [def] training manuals from the single seat F/A-18 Hornet, to the multi-seat P-3C Orion, even including the uniquely designed UAV Unmanned Aerial Vehicle. Each of these type wings reference dozens of squadron's Standard Operating Procedures (SOP), Carrier Air Group Commander (CAG) instructions, and other governing documents. Through this brief description, it is intended to illustrate that the aviation combat readiness model is in no way a bedrock standard, but rather a fluid model

that is affected by countless external forces and modified, and edited at irregular intervals. This thesis will attempt to model some of these external forces.

A collection of nine of these external forces were to be briefed in the summer of 1998 at a readiness briefing for then Commander in Chief, United States Pacific Fleet, Admiral Archie Clemins in Pearl Harbor, Hawaii. From these nine external forces, the concept of "P4 + WARTS" was introduced. Captain Phil Mills, AIRPAC N-8 Force Readiness and Operations, and Commander Peter Hunt, AIRPAC N-8 S-3 Readiness Officer, desired to create an acronym of the external forces that combined to drive aviation readiness. After reviewing several potential combinations, the two officers coined the phrase "P4 + WARTS", to mean "People, Planes, Parts, and Petrol, plus Weapons, Adversaries, Ranges, Temporary Assigned Duty (TAD), and Simulators".[def] Admiral Clemins found this model to so soundly and accurately depict the factors effecting readiness that he immediately incorporated it into his briefings, later to become a common term throughout aviation readiness circles.

These nine factors have become so intertwined in the readiness model that it would be nearly impossible to compute or predict a command's readiness without looking at the effect of one factor against the next. This concept directly coincides with the "causality principle",[9, def] in that cause must always precede effect. For example, without sufficient fuel, a flight could not be flown, regardless of the availability of ranges or weapons, thereby reducing readiness. Without maintenance parts, planes could not be flown, regardless of the availability of adversary forces or personnel to repair the craft.

---

[9] Eric Max Francis, "The Law List - *Laws, rules, principles, effects, paradoxes, limits, constants, experiments, & thought-experiments in physics*", Seven Sisters Production, http://www.alcyone.com/max/physics/laws/index.html, 01 Jun 2000.

The cause of external forces and "P4 + WARTS" will resultantly affect readiness. While these components have added to the accurate depiction and prediction of a command's readiness, they have also dramatically added to the complexity of the readiness prediction model.

4. T & R Transition

In the summer of 1995, the last edition of the two-dimensional Squadron Training and Readiness Manual was released as a joint COMNAVAIRLANTINST 3500.63C, COMNAVAIRPACINST 3500.67C instruction. This instruction remained in effect for only three years before it was cancelled by the release of the newly-refined, six-dimensional model depicted in the Squadron Training and Readiness Manual COMNAVAIRLANTINST 3500.63D, COMNAVAIRPACINST 3500.67D instruction on July 13, 1998. Less then fifteen months later, in the fall of 1999, due to changes in the operational tempo and mission requirements of the fleet, some type wing commands were required to once again execute a new version of their own Training and Readiness Instructions. This new model was later released to all aviation units in the spring of 2000 as COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E, on 24 Mar 2000[10]

Naval Aviation Units are required to report Aviation Combat Readiness by the fifth of each month, from all aviation squadrons and their detached units, worldwide, to their parent commands, type wing commanders, and type commanders[11] via a classified

---

10 COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E – *"Squadron Training and Readiness"*, 24 Mar 2000.

11 COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E – *"Squadron Training and Readiness"*, 24 Mar 2000, p. 7.

message. This message also includes information about a command's manning levels, flight-hour utilization, type mission accomplishments, and other important squadron actions and exercises. Using the latest in Message Text Format (MTF) technology, this document and its priority contents are potentially visible to the highest levels of Chief of Naval Operations' (CNO) office moments after its dispatch from the issuing squadron. This readiness message and its corresponding "T-Level", in conjunction with other readiness indicators, is used to determine which squadrons to send into combat, which squadrons to allocate assets to, and which squadrons to return to homeguard [def] for further training. It should be understood that this message holds the highest visibility and exposure of the fleet's aviation readiness for combat.

Within the last decade, naval aviation readiness models have gone through significant modifications from a simple two-dimensional model to the complex six-dimensional model. These models had to be accurate, complete, and workable, but flexible enough to meet the needs of all seventeen operational type wings. The concept of "P4 + WARTS" had to be considered, as well as the requirement to meet timely reporting deadlines. Due to the complexity of this new model, it soon became evident that it would be impossible for aviation personnel to meet the demanding report requirements using traditional tracking devices. It was determined that a software automation tool would be required to meet the needs of naval aviation readiness, or the newest readiness models could not be implemented. Failure to implement the newest readiness model would result in an immediate decline in the Navy's ability to accurately predict the combat readiness of the fleet and track its resources and personnel in performing the mission of national defense.

5. Tasking

In August, 1997 the Human Factors Quality Management Board (HFQMB) directed the Operational Risk Management (ORM) Process Action Team to evaluate a software product called SARA, short for Squadron Assistance Risk Assessment,[12] developed by the Boeing Aircraft Company of St. Louis, Ill..[13] SARA was initially designed to support the AV-8B Harrier community and later, U. S. Marine Corps Aviation. Commander Ken Ireland, CNAP N80A was tasked with leading the evaluation of the system. He initiated the evaluation by assigning ten squadrons as test and evaluation sites. In the early spring of 1998 while I was assigned as a pilot at one of the test and evaluation sites, HELANTISUBRON SIX (HS-6), I was called upon to review the proposed software tool. I was selected due to my knowledge of software development and my expertise in squadron operations. Due to SARA's inability to accurately depict the aviation operations model, my evaluation[14] was highly negative towards the product. These findings will be noted later in the thesis. Due to the large number of negative evaluations of the SARA product from all of the test sites, AIRPAC elected not to pursue acquisition of the product, but rather solicit the development of a new aviation operations tool designed for the U. S. Navy. After forwarding my review of the SARA product to AIRPAC, it came to the attention of another officer at AIRPAC, Lieutenant Commander (LCDR) Steve Ruth, an officer that I once served with at HS-6. LCDR Ruth recommended to AIRPAC that I be consulted on the development of any

---

12 *SARA Times*, Volume 1, Issue 1, The Boeing Company, St. Louis, Missouri, July 1999.

13 CDR Ken Ireland, *"Evaluation of 'SARA' ORM Software"*, April 1998, Letter Drafted to Chain-of-Command.

14 SARA Evaluation Results Folder, on file, AIRPAC N845 Office.

new readiness automation tool. A few days later, I was ordered out of my command at HS-6 and was tasked with the development of a new aviation automation tool – and the system now referred to as SHARP was born.

For the next two years, the SHARP development team would develop a new software automation tool, under the strictest of deadlines, against traditional development methods, despite interference and obstacles from various members of the military chain-of-command, and with competition from numerous commercial contractors and members of the Foreign Service. For the first time in recent history, a fleet-wide aviation readiness tool was to be developed by aviators, for aviators. This development team was tasked with designing a system that could be rapidly deployed to the fleet, be flexible enough to satisfy the changing needs of the aviation readiness model, and cost effective enough to meet the restrictive budget of operational forces.

6. Rapid Application Development

Rapid Application Development or RAD is the methodology of developing a software system through a highly efficient process of analysis, planning, design, prototyping, system development, change control, testing, and deployment in a cyclic environment.[15] Kent Beck, software engineer and developer, has been credited with keying the title "Extreme Programming" or "XP" to define a formal methodology of RAD. XP is based on the concept of breaking a system up into small projects or stories to be conquered as independent modules vice the development of a project as a complete

---

15 RADD Methodology, Analysts International, 2000.

unit.[16] This thesis will compare and contrast the development of the SHARP Project with the methods of Extreme Programming.

B. OBJECTIVES

For decades, Naval Aviators have attempted to compute operational readiness through a variety of primitive and advanced automated tracking tools. Despite all well-intended efforts to develop a successful automation tool, Naval Aviation was left with a myriad of fragmented systems that failed to meet the needs of flight operations. To resolve this dilemma, a team of software developers and naval aviators were brought together to design a single automated tool to meet the needs of all seventeen type wings in a system later titled SHARP. Throughout the development process, the SHARP team was required to re-subscribe to the methods of rapid prototype development to design a system for the fleet despite numerous obstacles. This thesis will outline the development and design of the Sierra Hotel Aviation Reporting Program or SHARP. Specifically, this thesis will compare and contrast the traditional models of software development, prototype development, rapid prototype development, and military system development using the SHARP system as a prototype example. This evaluation will emphasize strategic benefits within the military rapid prototype development environment, as well as catalog weaknesses in the design methodology. A review of Extreme Programming techniques, as defined by Mr. Kent Beck, will be compared and contrasted with the development of the SHARP Project. Aviation subject matter experts were intimately involved with every phase of design and engineering, ensuring the development of a quality project to the fleet. This thesis will emphasize the importance of true subject

---

[16] Kent Beck, *"Extreme Programming Explained"*, Addison-Wesley, 15 Oct 1999.

matter experts in the requirements search, design, development, testing, and implementation of DoD based products. These benefits will then be suggested for direct modeling into existing and future projects. It is the opinion of many software engineers that the military environment is a unique field of development due to its real time development needs, high level of mission critical systems, and reliance on competitive technology to satisfy governmental standards and requirements.

C. RESEARCH QUESTIONS

1. Primary Question

The primary question for research and discussion within this thesis is to determine if traditional software evolution models are acceptable in the rapidly growing environment of Military Rapid Application Development (RAD). Uniform members of the armed service demand that their weapons perform flawlessly. The consequences of a faulty weapon can result in nothing less than defeat or even death. While the job of national defense is fraught with danger, it is not a requirement that we sacrifice safety for the purpose of product distribution. With recent advances of software technology, logic-based systems have become more of a weapon for and against the military members. The SHARP software tool has been deployed on all of the Navy's aircraft carriers, helicopter detachment-assigned vessels, squadron operations offices, and various other support aviation units for determining combat readiness. SHARP is the ONLY United States based software tool that has determined combat readiness of aviation units actually engaged in combat operations, most recently in the Balkan and Yugoslavian conflicts.[17] Can traditional software engineering methods meet the needs of the military for small

development projects or should the military examine a return to Rapid Prototype Development?

2. Subsidiary Questions

a. COTS to the Fleet

In an attempt to meet the strict timelines for software deliverables in the military, some units have turned towards "commercial off the shelf" (COTS) products. Many of these products have been designed to meet commercial or industrial requirements and have not been developed with the military in mind. Some COTS products have been based on existing military technology and then modified for commercial users, actually increasing usability. These COTS products are then reintroduced back to the military as remanufactured technology, as in the example of the issue of commercial hand-held Global Position System (GPS) units for aviation commands to compensate the for the inability to rapidly modify airframes to accept internal GPS systems.[18] This thesis will attempt to answer the question if COTS products can be modified and customized to meet the rapidly changing needs of the military, namely, aviation readiness reporting.

---

17 MSGID/GENADMIN/VAQ-134/SHARP (c) , Subj: *Monthly Training and Readiness Report (MTRR)*, NOV 1998 Data.

18 MSGID/GENADMIN/COMNAVAIRPAC (u), Subj: *Distribution of Trimble Ensign GPR Receivers,* DTG 162053Z Jul 1993.

b. Is modified COTS still COTS?

Changes in military acquisition policy have dictated that greater efforts be made to incorporate COTS products into the military inventory, either as modules to systems under development or as stand alone systems.[19] COTS systems are defined as:

- Commercial items customarily used for non-governmental purposes and offered for sale, lease, or license to the general public.
- An item evolved from such an item, as previously stated, that will be available within sufficient time.
- Items that are standard modifications available in the commercial marketplace or are minor modifications.
- Any non-developmental item developed exclusively at private expense and competitively sold in substantial quantities to non-federal government agencies.[20]

A number of contractors have attempted to market aviation tracking systems to the Navy under the guise of COTS, in an attempt to gain favorable acceptance and acquisition. These models were marketed despite the fact that their systems would require significant changes and modifications to meet the current aviation readiness

---

19 Department of Defense Regulation 5000.2-R, *"Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs"*, 15 March 1996.

20 John Foreman, *"On the Front Lines of COTS - Lessons Learned, Speculation for the Future"*, Briefing slides courtesy of the Software Engineering Institute, Carnegie Mellon University, May 8, 1998. http://www.sei.cmu.edu/cbs/cbs_slides/stc98/frontlines/index.htm, Slide 12.

model. This thesis will attempt to answer to what extent one can modify a COTS product to the point that it is no longer considered COTS.

c. RAD Models

Due to a fixed deployment deadline, the SHARP team had to divert from traditional development models and revert to the concept of RAD. The RAD model has potentially gained as many detractors as it has gained in supporters. John Munson, a software engineer and professor of computer science at the University of Idaho, refers to software engineering models as "Cave Art". He continues by stating that, "it's primitive. We supposedly teach computer science. There is no science here at all."[21] His assumption that software engineering is in its Genesis is supported by the gross number of new software design models and requirements that exists today. The Department of Defense (DoD) alone recognizes over one dozen new software engineering requirement models in its publications, including the DoD Information Technology Security Certification and Accreditation Process (DITSCAP)[22] model, Configuration Management (CM) Software and Documentation Delivery Requirements Document[23], and the Defense Information Infrastructure Common Operating Environment (DII COE).[24] Each of these specifications dictates unique reporting requirements, specification criteria, and development models. The most frustrating point is that these models all fall under

---

21 Charles Fishman, *"They Write the Right Stuff"*, Fast Company, Issue 6, Page 95. http://www.fastcompeny.com/online/06/writestuff.html

22 Department of Defense Directive 5200.40 – *"DoD Information Technology Security Certification and Accreditation Process (DITSCAP)"*, 07 Oct 1999.

23 CM-165-60-05, *"Configuration Management Software and Documentation Delivery Requirements"*, Version 4.1 01 March 1999.

24 Defense Information Infrastructure Common Operating Environment (DII COE) Document Series, http://diicoe.disa.mil/coe/, Defense Information System Agency.

different umbrellas within the military chain-of-command, require an extensive amount of documentation and reporting in unique formats, potentially add significant delays to the development of software products, and increase the cost of development. None of these DoD accredited models are conducive to the concept of RAD. Extreme Programming could very well set the path for a revitalization of RAD. Is there actually a viable model to RAD, and can RAD be certified as a valid development model in the military environment? The SHARP system was able to distribute its first prototype copy to the fleet just six months after the concept was introduced. How can lessons learned from this prototype be expanded to other development teams?

d. Reusable Code

In an attempt to speed up the production of the SHARP system, reusable code was used to the maximum extent possible. This thesis will endeavor to answer to what extent reusable code is beneficial in the development of RAD applications.

e. Subject Matter Experts

Numerous contractors and developers have issued software products with the sole intention of meeting system requirement specifications (SRS) without understanding the driving factors behind them. One of the most contentious aviation software products is the Naval Aviation Logistics Command Management Information System (NALCOMIS).[25, 26, 27] This system is continually under development by the

---

[25] Naval Aviation Logistics Command Management Information System - NALCOMIS Information Hold Page http://www.massolant.navy.mil/nalcomis/nalcomis.htm, SPAWAR.

[26] MSGIG/GENADMIN/CNAL, Subj: "*SHARP-NALCOMIS (FIST) INTERFACE APPROVAL FOR CNAL AND CNAP SQUADRONS*", Jul 2000.

[27] Mark Burgunder, "*Feedback on Naval Aviation Flight Data White Paper*", CNAP 15 Jun 2000 – LT. Christopher L. Williamson, "*NALCOMIS Requirements*", Jun 2000.

Space and Naval Warfare System Center (SPAWAR) and contracted to ManTech International Corporation.[28] The requirements for this system are administered by aviation maintenance personnel and then submitted to non-aviation personnel for development. One of the greatest shortcomings in NALCOMIS is that aviation crewmembers are the primary operational data entry members, but are not involved in the requirement development. Aviation crewmembers are required to log all post-flight data into NALCOMIS, but can not log all operations data due to limitations into the system. Due to the fact that operational data is not collected in an efficient manner from post-flight data, operational information is unusable if not inaccessible from NALCOMIS. The SHARP system is the first aviation automation tool developed by aviators, for aviators, to collect operations data. Without aviation subject matter experts (SME), the SHARP system would be just another software tool, issued to service members without a true understanding of the requirements. This thesis will answer to what extent subject matter experts play in the requirements, developmental, and testing phases of software engineering.

f. Streamlining DoD Software Development

As part of the question to RAD Models, Reusable Code, and Subject Matter Experts, this thesis will address the question as to how the Department of Defense (DoD) can streamline its software development protocol to more efficiently produce a product. The first hurdle to overcome is to know which development protocol to follow. Due to the disjointed and complex list of instructions, it is difficult to know which instruction to follow and which office has jurisdiction over the development of the

[28] ManTech NALCOMIS Information Page - http://www.mantech.com/defense/infodef.htm.

product. The current development protocol is very bulky and difficult to understand. One SPAWAR software engineer, Mr. Don Johnson of the office of Horizontal Integration,[29] informed me that each of the departments or units within SPAWAR have their own autonomy to develop systems under whichever model they decide to follow, even if that model is exclusive to that unit. This thesis will recommend possible directions as to how the Department of Defense can streamline its software development protocols into a unified format, as well as what efforts are being made to solve this problem.

g. Uniform Service Development

Since World War II, the Navy Construction Battalion or Seabees have taken a can do attitude to build almost anything, under almost any environment and under the most hazardous of conditions. Since their inception in 1941, the motto of the Seabees has always been "we build, we fight"[30]. Their history details sixty years of uniform service members making countless efforts to build runways, buildings, and support services for the Navy, at a great savings to lives and resources. Nearly sixty years later the field of construction has moved from the dirt fields to the motherboard. With the changes in technology, the greater educational background of service members, and the increased reliance on military subject matter experts, it only makes common sense that military members receive a greater tasking with the development of technology system. What the Seabees did for over sixty years on the construction site can now be mirrored today in the development laboratories by uniform service members. It is prime time for

---

[29] Phone Con, 30 Jun 2000, Don Johnson, Office of Horizontal Integration, 619-524-7243.

uniform service members to be recognized for their abilities to develop software to the benefit of the Navy. This thesis will address to what extent uniform service personnel can serve on development design teams and engineer a successful software tool.

h. Uniform Service Education

Through the courtesy of the Naval Postgraduate School (NPS), I was provided an education beyond that of normal uniform service personnel. I was permitted to pursue my Masters Degree in Software Engineering through distance learning channels, while still serving operational commitments. Most members of the uniform service are required to take time off from operational commitments to attend postgraduate school. Due to manning shortfalls, many service members have found it nearly impossible to be assigned to such a tour. After pursuing my advance degree in the evenings, I was able to utilize my knowledge the next morning on the SHARP Project. This thesis will address questions regarding the assignment of uniform service personnel to advance degree education concurrent to their operational commitment, as well as the direct benefits to uniform service member assignment to development teams.

D. SCOPE, LIMITATIONS, AND ASSUMPTIONS

The primary scope of this thesis is concentrated around the evaluation, comparison, and contrasting of the unique development of the SHARP Aviation Tracking System with other like products, the uniform service members who developed it, with lessons learned and recommendations to the development process.

---

[30] Chief Of Naval Information, Seabees Home Page, http://www.chinfo.navy.mil/navpalib/factfile/personnel/seabees/seabee1.html, Department of the Navy.

This thesis is limited to a brief evaluation of the benefits of COTS products over the development of new systems, for the purpose of an energy-benefits analysis. This thesis is limited to a brief analysis of other military development teams for generating a project baseline and for comparison and contrasting development methods. This thesis is limited to a brief analysis of the complex readiness predictive model. Due to the complexity of the aviation readiness predictive model, further discussion of this topic will be reserved for future work.

There are no assumptions made for this thesis.

E. LITERATURE REVIEW, METHODOLOGY, AND ORGANIZATION OF STUDY

1. Literature Review

Due to the fluid and emerging nature of Software Engineering, its corresponding documentation, methodologies, and literature, this thesis will review information from a multitude of sources. Sources shall include traditional hard cover published literature, submitted thesis and doctoral works, lecture notes and handouts, and electronic media.

This thesis shall rely on Department of Defense Instructions, Requirements, and Standards for establishing authorized and suggested standards of development in the military environment. This thesis shall rely on Department of Defense and Department of the Navy Instructions for establishing a historical background and requirement for SHARP Development.

2. Methodology and Organization of Study

In order to objectively describe the events and significance of the development of the SHARP System, this thesis shall take a methodical approach to describe the events

before, during, and after the engineering of the system, as well as the crucial influences on the development of the system.

This thesis shall include a brief historical background of Naval Aviation Readiness before the existence of the system, the requirement to build the SHARP System, Research Questions, and Thesis Format. The thesis shall include a review of traditional software engineering techniques, a review of DOD software engineering requirements, a review of readiness requirements, and a review of the SHARP development model. This thesis shall include a review of search and discovery, a review of the development of the SHARP System, a contrast with contemporary models, and a review of interviews with DOD and Non-DOD contractors. This thesis shall include a review and analysis of Naval Aviation requirements, the SHARP System Requirements, a comparison and contrast with traditional development models, and a comparison and contrast with comparable systems. This thesis shall conclude with summary of the findings of the SHARP System Development Study, answers to the Thesis Questions, and recommendations to improve the development of software systems in the rapid application military environment.

For the purpose of this thesis, excessive abbreviations, quotations, and evaluations shall be held in appendixes.

F. DEFINITIONS AND ABBREVIATIONS

For the purpose of consolidation, definitions and abbreviations can be reviewed in the concluding chapters attached to this thesis.

G. CHAPTER SUMMARY

Historically, military combat readiness had been computed based on a factor of force concentration and weapon inventory. Within the last three decades, military readiness has made a progressive shift towards measuring its readiness through a more complex metric of various factors, including personnel qualifications. In the last the decade, from prompting of the HFQMB and the office of the CNO, a more demanding model of readiness, including the factor of "P4 + WARTS" was initiated. This model was far too complex to be managed using traditional tracking methods. To facilitate the evolution and execution of this new model, various software products were developed and distributed to aviation units. Despite repeated development efforts, the distributed software products failed to meet the needs of the customer squadrons and in some cases actually added to the burden of flight data tracking.

The guiding readiness document, or Training and Readiness Manual, was written with the concept of fluid operations in mind. This manual was constantly revised to reflect the current concept of operations as well as emulate the directives set forth by members of the chain-of-command. These revisions constantly changed the requirement set that any supporting software was required to follow. Due to the fashion and logic in which previous systems were designed, testing and evaluation made it evident that it would be more efficient to develop a new system rather than attempt to revise the existing systems.

During the efforts to determine a new system to support Aviation Readiness, I was tasked with creating a team to develop an alternative software tool based on positive points from existing systems, relevant requirements, and an outlook towards future

changes and requirements. The SHARP SRS listed over two thousand separate and distinct requirements and criteria for the product, including provisions for Flight Scheduling, Flight Logging, Electronic Log Books, Electronic Training Jackets, Operational Risk Management, and the creation of an electronic and classified Training and Readiness Message.[31]

The primary objective of this thesis shall be as a case study of the development of the SHARP System in the unique military rapid application development environment. The term RAD has received a great deal of negative connotations from traditional developers due to its radical and fluid design methodology. This thesis shall detail some of the benefits of the RAD environment, as well as its applicability to the military environment. For the purpose of this study, a comparison and contrast shall be made with existing systems, development techniques, and the unique needs of the uniform services. The secondary objective of this thesis shall be to answer research questions regarding software engineering in the military environment.

[31] SHARP SRS Document, on file with the SHARP Development Team, Commander, Naval Air Force, U. S. Pacific Fleet.

II. LITERATURE REVIEW, THEORETICAL FRAMEWORK, AND BACKGROUND

A. REVIEW OF TRADITIONAL SOFTWARE ENGINEERING TECHNIQUES.

James Burke, philosopher, commentator, and author of many great works on the human existence once wrote about the dynamic web of change and its balance of knowledge from the past to the future, stating:

> "If knowledge is an artifact, and innovation is the result of interaction on the web, then the way for us to better manage change is to become acquainted with the interactive process. So, in a future world changing too fast for the old-fashioned, specialist approach to education, it may benefit us to require young people to journey the web as a primitive learning experience, much in the same way as we taught their ancestors to read books after Gutenberg had invented the printing press. Schools might train students to weave their way idiosyncratically through the web, imagining their way to solutions, rather then learning by rote lists of data that will be obsolete before they can use them.
>
> We might even consider changing our definition of intelligence. Instead of judging people by their ability to memorize, to think sequentially and to write good prose, we might measure intelligence by the ability to pinball around through knowledge and make imaginative patterns on the web."[32]

As Mr. Burke so well stated, we can not simply continue to base our future on the concrete learning of the past, but need to be prepared to think with open minds toward to future, using the knowledge from the past. Before any real development of the SHARP Project could take place, the engineering team required a sufficient knowledge of previous development techniques to determine the best course of SHARP development.

---

[32] James Burke, *"The Pinball Effect: How renaissance water gardens made the carburetor possible – and other journeys through knowledge"*, London Writers Ltd., London, England, 1996, p. 6.

As part of the review of potential development techniques, a search and discovery of existing models was conducted from literature and lecture material available to the development team. Some of the more popular models for software engineering include the simple Build and Fix Approach, the Stagewise Development Model, the popular Waterfall Model, Test Development, Exploratory Programming, the Prototype Model, the Incremental Model, the Boehm Spiral Model, the Win-Win Spiral Model, the Legacy and Reuse Software Life Cycle, and the newly defined Extreme Programming Technique.[33, 34] A brief overview of popular models and findings can be reviewed APPENDIX C – SOFTWARE ENGINEERING TECHNIQUES.

1. Rapid Application Development

For the purpose of this thesis, it is necessary to give a brief explanation of Rapid Application Development (RAD). This explanation of RAD will be used as part of the comparison and contrast of the development of the SHARP System.

> "Rapid application development (RAD) has long promised to be a boon to the computing community. The idea is to develop a method of designing software so that the whole process is quick, painless, and nearly effortless. The tools should be easy to learn, powerful, and allow the design to interface his/her freshly minted application with other applications, databases, and file types."
>
> Ted Brockwood[35]

For many years, software development purists have looked at RAD with great disdain due to its radical practices and lack of formal methodology. These purists have

---

33 David F. Redmiles,. *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

34 James A. DeBardelaben, Lecture Notes of *"Cost Modeling for Embedded Digital Systems Design Module 57"*, Pennsylvania State University, Pittsburgh, Pennsylvania, 15 Sept 1998.

35 Ted Brockwood, *"Rapid Application Development"*, The Web Developer's Journal, 10 Jul 1997.

focused their energy on traditional models of structure and procedure, while RAD developers have utilized flexible methods of development based on the cyclic rapid production of small pieces of a major system. As Ted Brockwood stated in his article, development under the RAD methodology should be "quick, painless, and nearly effortless." The truth and fact of the matter is that RAD is quick, but is far from painless and is fraught with great efforts.

RAD requires great efforts from management to ensure that the rapid progression of the project is kept in check and controlled to prevent the cyclic development from getting out of hand. Rapid Development is accomplished through a stagewise cyclic development process that includes an/a:[36]

- **Requirements Analysis Stage** to describe the high level requirements of the project, the system's business use cases, and scenarios. The requirements are edited into a System Requirements Specification or other proprietary requirement document.
- **Project Plan/Estimate Stage** to develop and author a document covering estimated costs, schedules, tasks, dependencies, responsibilities, approaches, communications, and goals. This document is constantly to be reviewed and edited to fit the needs of the project.
- **Design Stage** to evaluate and assign requirements in a hierarchical level to design the development process. The design stage includes assignment of the data module, GUI, object, architecture, integration, data conversion, reports, and business process rules.

---

36 *"RADD Methodology"*, Analysts International, 2000.

- **Prototype Stage** to develop a working GUI model of the system's representation, interface navigation, and data incorporation. The prototype permits client review of the developer's interpretation of the system's requirements.
- **Development Stage** to expand the prototype into a working executable system. The development stage includes testing at the unit, system, and integration level; and the adherence to established system standards and GUI format.
- **Change Control Stage** to manage any changes in requirements or design standard.
- **User Testing Stage** to apply the test cases established and defined in the design stage and refined through the development process. User testing includes the classification, risk acceptance, and documentation of observed bugs/errors.
- **Deployment Stage** to deploy, install, and implement the new system. The deployment stage shall also include user training, technical training to ensure proper system utilization.

It can in no way be taken that RAD is a simplistic approach to software development, but rather a more refined approach to make software development simpler through fluid methods. Once each cycle of development has been completed, as with traditional spiral development models; lessons learned, modified requirements, and mitigated risk factors can be integrated into the next cycle to further increase the accuracy of requirement accomplishments and efforts to meet the customer's needs.

RAD permits developers to build a system more efficiently by the incorporating incremental design, multiple reviews of the requirements, and the engineering and presentation of a prototype early in the design. The cycle development method permits multiple returns to each stage of design to ensure completeness. RAD requires a development tool that is powerful, flexible, and easy to use, that can take a user from design, through prototype, to development using the same code and logic structure. Many prototype systems do not have code and logic structures that are easily transferable into executable/compiled code, resulting in a breakdown of efficiency. Any effort and energy to design a prototype is lost when it must be reengineered into a higher level logic at the development stage.

In an effort to increase RAD efficiency and to promote product development in a language that can create working prototypes convertible to executable/compiled code, many software language developers have shifted to "Visual" like languages.

2. Visual Languages

Visual Languages can be defined as any language, symbology, or architecture that takes advantage of a graphical representation of logic or predefined functions and procedures to be combined to create a working system. Many developers misinterpret the concept of visual language by commercial products misleading labeled as "Visual". These products are often referred to as Pseudo-Visual Languages.

Visual Languages permit the development of a product by the graphical placement of objects on a field. Each object has predefined propertied, functions, procedures, and attributes that determine the action and interaction of the object to other objects based on their physical representation on the field. The ultimate function of the

product is determined by the overall interaction of the combination of all objects and their placement on the development field. [def] Visual languages take advantage of program development through the two- (or more) dimensional representation of object placement and spatial orientation with other objects. Conventional textual languages are not considered two-dimensional since the compiler processes the object as a one-dimensional stream of code in text characters. Visual languages may be further classified, according to the type and extent of visual expression used into icon-based languages, form-based languages and diagram languages.

Pseudo-Visual Languages use a combination of object placement in parallel with textual code to represent the procedures and functions of the product. Users can graphically develop the flow and design of a system, compose data structures with associated references and links, and model logic and mathematical statements through the drag-and-drop and textual environment. Through this environment, users can quickly develop GUI level prototypes that directly relate to executable/compiling code. Object code can be further refined and modified to complete required logic statements.

The commercial sector markets a number of true Visual Languages, including Prograph, Logical Vision, Peri Producer, and Lab View to meet the needs of prototype development. The commercial market has also introduced a wide variety of Pseudo-Visual Languages to increase development efficiency. Languages and tools include project design utensils such as **Visio**; to model databases through **Eryn** and **SQL 7.0 DB Tools**; to model project timelines through **Microsoft Project;** and to assist with actual project development through **MS Visual Studio (Visual Basic, C++, J, and Fox Pro),** and **Symantec Visual Café,** as well as a host of other visual products.

3. Extreme Programming

For the purpose of this thesis, it is necessary to give a brief explanation of the concept of Extreme Programming (XP). This explanation of XP will be used as part of the comparison and contrast of the development of the SHARP System.

Extreme Programming is a concept of system development that incorporates a radical method of "speed-wise" design. Traditional development methods have incorporated the Waterfall Model to depict a top down stagewise flow of system engineering from the Requirement Stage to the Deployment Stage in a non-returning environment. Extreme Programming incorporates a form of the Spiral Model to permit repetitive reviews of system stages in incremental design. The backbone of this technique is based on the twelve practices of XP:[37, 38]

- **The Planning Game** where management and developers play out the rules, scenarios, and timelines for developing the system.
- **Frequent Releases** where customers are serviced with multiple releases of the project to solicit requirement verification and validation through each cycle of development.
- **System Metaphor** that establishes classes, patterns, and objects that forms the architecture of the system.
- **Simple Design** to establish a simplistic design to a complex problem. Simplicity is implied as the minimal solution. One principle is to "do the simplest thing that could possible work."

---

37 Kent Beck, *"Extreme Programming Explained"*, Addison-Wesley, 15 Oct 1999.

38 *"Extreme Programming"*, Wiki, Cunningham and Cunningham, Inc, 30 Jul 2000.

- **Unit Tests** to test the independent modules of each of the system units and **Functional Tests** to test the complete package.
- **Refactor Mercilessly** to consolidate and combine duplicate efforts into one unified code package or module.
- **Pair Programming** to pair engineers to the same workstation under the assumption that, not withstanding people skills, two programmers are more than twice as efficient as one programmer given the same task.
- **Collective Code Ownership** to promote the sharing of code across developing teams to ensure compliance with standards and reuse
- **Continuous Integration** to pool code together so that changes to one module simultaneously effect changes across all of the corresponding modules of the major system.
- **Forty Hour Week** rule to prevent overworking of programmers and developers.
- **Onsite Customer** to provide real time feedback of development by adding a subject matter expertise.
- **Coding Standards** to promote standardized conventions of semantics, variable declarations, and logic schema to ensure compliance and compatibility among all programmers.

After a review of the Extreme Programming Technique and its corresponding literature, it became evident that it was simply a repackaging of existing RAD methodologies into a disjointed schema of ideas. The concept of "playing" a "Planning Game" to produce a system management document borders on the infantile. "Frequent

Releases", "System Metaphors", "Code Standards", and "Continuous Integration" are common spiral techniques. "Simple Design," when defined as a minimal solution, is referred to as meeting the requirements through the most efficient method possible. Simplicity does not necessarily equate to reduced functionality or aesthetics but rather to resourceful development. Function and aesthetic requirements would be determined as part of the development "Planning Game." "Testing" is nothing new and spiral development models encourage unit and functional testing with each revolution of the development. Mr. Beck encourages automated testing methods, which are a direct factor of the development language chosen for the project. "Refactoring" and "Collective Code Ownership" are simply alternative semantic to refer to code reuse. "Pair Programming" is a novel concept, assuming that members have solid work ethics, like personalities, and the task is conducive to being worked in pairs. While some tasks may serve well to be developed in intimate pairs, it is very inefficient to have to justify a known principle to a work partner before one can code it into the system. Mr. Beck suggests that development teams should adhere to a "Forty Hour Week", contrary to traditional SE practices of late night "Pizza Sessions" where engineers prayed for midnight epiphanies. It should be recognized that the potential for errors increases as the work day labors on. Mr. Beck's encouragement to developers to know when to call it quit and go home to ensure that maximum performance is delivered during billable work hours, vice development through desperation and exasperation. "Onsite Customers" are commonplace in complex development scenarios and are almost a requirement to the success of a RAD or Spiral Development.

B. REVIEW OF DOD SOFTWARE ENGINEERING REQUIREMENTS

In parallel with a review of potential engineering techniques, the SHARP Development Team required a review of DoD Software Engineering Requirements and Instructions, and Commercial Development Standards to determine the best approach for development. Some of the more prominent development standards include the legacy DOD-STD-2167A and DOD-STD-7935A Standards, the transition MIL-STD-498, Commercial Standard ISO/IEC 12207, the Joint Instruction DII COE, DoD Regulation 5000.2-R, and the DITSCAP DoD Directive 5200.40. A brief overview of prominent development standards and findings can be reviewed in APPENDIX D - DOD SOFTWARE ENGINEERING REQUIREMENTS

Over the last two decades, the Department of Defense has attempted to direct the development of software projects through a series of instructions, directives, and standards. These standards were intended to support the development, design, and acquisition of various software systems from mission-critical to office automation tools. Legacy standards were progressively superseded by revised standards, some strengthening the requirements, formats, and procedures, while other standards reversed trends to loosen the reins of developers and granted more developer autonomy.

In an attempt to standardize DoD Software Development Efforts, a shift was made to integrate future military standards with established commercial standards or even go as far as to influence changes in commercial standards. MIL-STD-498 was established as a stopgap measure while DoD attempted to shift its efforts to toward IEEE accredited standards. Shortly after MIL-STD-498 was released to the fleet, efforts were made to institute the DII COE Standard to manage and direct development through a

barrage of documents and instructions. The DII COE Chief Engineer, Mr. Kenneth Wheeler, stated that the DII COE standard was not yet widely accepted in industry, but that he was attempting to impose his influence on defense contractors through required compliance with the standard. Each of the requirements and standards, as outlined in the appendices of this thesis, are fraught with loopholes and exceptions. Compliance is mandated but not entirely enforced. Few if any of the DoD Standards are accredited by any civil or professional organizations. Mr. Wheeler informed me in a phone interview that his office has no intention to seek accreditation from any agency because the "the return on investment is not there."[39]

After a sufficient investigation of available and required development standards, the SHARP Development Team realized that none of the prominent standards would fit the design requirements of the project. The project would either fall outside of the scope of existing standards or the ultimate burden of the standard would overshadow the actual project. It was elected to follow a logical process of development based on general software development practices.

C. REVIEW OF DOD READINESS REQUIREMENTS, SPECIFICALLY NAVAL AVIATION.

Naval Aviation, in itself, serves one of the most complex missions of the Department of the Navy. This complexity involves the requirement to operate a sea-going aviation unit. This marriage of sea and air combines instructions, directives, and regulations from across many aspects of the Navy to develop the requirements for Naval

---

39 Phone Con, 13 Jul 2000, Mr. Kenneth Wheeler, DII COE Chief Engineer, DII COE Agency Joint Office, Falls Church VA., 703-681-2304.

Aviation Readiness. For the purpose of the thesis, it is necessary to discuss the requirements that drive the development of the SHARP System.

1. OPNAVINST 3500.38

The OPNAVINST 3500.38 - Universal Naval Task List (UNTL), signed September 27th, 1999, provides joint force and naval commanders an interoperability tool for use in articulating their mission requirements under the sponsorship of the Navy Warfare Development Command.[40] The UNTL is designed to provides the Naval Services with a standardized tool for describing requirements for the planning, conducting, assessing, and evaluating of joint training. The UNTL is a single source document that combines the Universal Joint Task List (UJTL) strategic and operational level war tasks with the Naval Tactical Task List (NTTL) into a requirement-based "mission-to-task" matrix.

In the joint-training arena, the task list provides a common language for documenting war-fighting requirements and reporting procedures to the Joint Mission Essential Task List (JMETL). For standardization purposes, the UNTL uses the same common language and task hierarchy of the UJTL. The list describes what needs to be accomplished for a given task, the variables in the environment that can effect the performance of a given task or mission, standards for measuring and evaluating effectiveness, and measures of performance. The list does not specify how the task is to be accomplished, nor does it specify who is to perform the task.

Variables in the mission or task environment include, but are not limited to:

- Physical Environment - Sea state, terrain, or weather

40 OPNAVINST 3500.38, "Universal Naval Task List" Sep 27, 1999.

- Military environment - Threat, command relationships, allies
- Civil environment - Political, cultural, and economic factors
- Theater of operations - Host-nation support, bordering assets, neutrality
- Immediate operational area - Maritime superiority, air superiority, logistics lines
- Battlefield conditions - Littoral composition, open-ocean, overland

The UNTL is broken down into five sections and accompanying sub-sections, encompassing both the Naval and Joint Tasks as:

- Introduction
- Mission Essential Task List Development
  - UNTL Organization
  - Naval Tasks
  - Conditions, Measures, Criteria, and Standards
  - Developing Mission Essential Task Lists
- Universal Joint Naval Task List
  - Strategic Level - National Military Tasks
  - Strategic Level - Theater Tasks
  - Operational Level Tasks
  - Tactical Level Tasks
- Conditions for Joint and Naval Tasks
  - Conditions of the Physical Environment
  - Conditions of the Military Environment
  - Conditions of the Civil Environment
- Measures for Joint and Naval Tasks
  - National Strategic Level Tasks Measures
  - Theater Strategic Level Tasks Measures
  - Operational Level Task Measures
  - Tactical Level Task Measures

The UNTL applies to all Navy, Marine Corps, and Coast Guard (Department of Defense related missions) assets, and includes the Strategic-National, Strategic-Theater, and Operational levels of war tasks. For the purpose of this thesis, the UNTL provides a listing of missions and tasks to be accomplished by naval air assets in the execution of national defense objectives. Battle Group Commanders, Type Wing Commanders, and

the Joint Training Master Plan[41] outline which airframe assets are to be assigned to which task. Factors to be considered when assigning specific tasks include airframe capabilities, support assets, training objectives, personnel manning, and readiness levels.

2. COMNAVAIRPACINST 3500.63 / COMNAVAIRLANTINST 3500.67

While the OPNAVINST 3500.38 (Universal Naval Task List) includes all of the tasks to be performed by Navy, Marine Corps, and Coast Guard assets at large, the COMNAVAIRPACINST 3500.63 / COMNAVAIRLANTINST 3500.67 lists all of the tasks, missions, and qualifications to be performed by Naval Aviation Squadrons, based on requirements from the UNTL. The most recent version of the 3500.63/3500.67 series was authorized for release by joint authorization of the Commanders of both the U. S. Naval Air Force Atlantic and Pacific Fleets on March 24th, 2000, as the Squadron Training and Readiness (T & R) Manual.[42] A brief explanation and history of the 3500.63/3500.67 instruction was included in Chapter 1 of this thesis.

The 3500.63/3500.67 series is broken down into subject matter enclosures as:

- General Guidance
- Squadron Training Matrices Review and Validation Process
- Squadron Training Matrix Format
- Squadron Monthly Training and Readiness Message Format
- Status of Resources and Training System Report (SORTS)
- 17 Independent Type Model Series Training and Readiness Matrixes
- Authorized Aircrew Numbers
- Total Flight-Hour Requirements

---

41 Joint Training Master Plan (CJCSI 3500.02).

42 COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E – *"Squadron Training and Readiness"*, 24 Mar 2000.

The T & R outlines detailed guidance for naval aviation units to report monthly readiness levels to the chain-of-command via an MTF message format. This monthly message includes, but is not limited to:

- Squadron Data
  - Squadron identification
  - Months prior to next scheduled deployment
  - Manning levels by type of crewstation
- Readiness Levels
  - Target manning readiness levels by PMA
  - Actual readiness levels by PMA
  - Mathematical readiness values by manning and PMA
  - Qualification point by PMA
  - Mission Flight-Hours by PMA
- Flight-Hours and Operational Data
  - Training, Operational, and Overhead Flight-Hours Totals
  - Training, Operational, and Overhead Flight-Mission Totals
  - Special Interest Hours
- Operational Detachment Data
- Exercise Participation
- Contingency Participation
- Other Data as required by Type Wing
- Commanding Officer Comments[43]

While many of the values of the T & R Report are simple query totals from events and hours completed, the readiness level calculation requires the cross referencing of crewmember manning, crewmember qualifications and PMA point totals, crewstation composition, and crewmember ACTC levels. To reach a targeted readiness level a command must get a fixed number of crewmembers above a predetermined threshold of PMA percent points, as T1 $\geq$ 85; T2 $\geq$ 70; T3 $\geq$ 55; and T4 $<$ 55.

The actual T & R Matrixes for each TMS are the heart of the 3500.63/3500.67 instruction. Each matrix outlines the qualifications that a particular TMS can obtain to

---

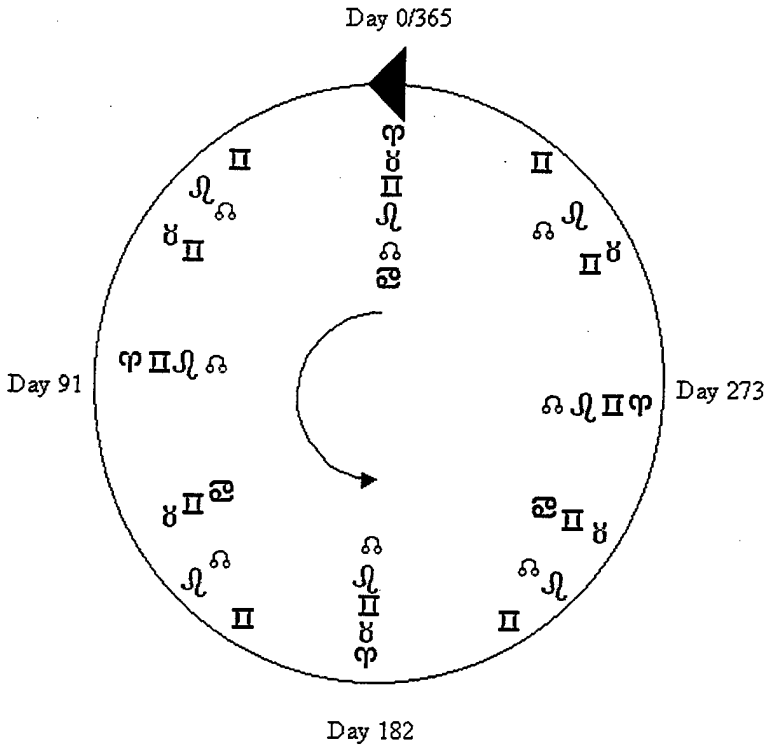
[43] COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E – *"Squadron Training and Readiness"*, Enclosure (4), 24 Mar 2000.

gain readiness points. The matrix qualification list is based on tasks and missions as assigned from the UNTL. Even though some of the qualification names listed on a particular T & R Matrix for one TMS may match the qualifications names of another T & R Matrix of another TMS, one can not generally assume that the qualification has the same level of importance, requirements, or resources. Each T & R Matrix is made up of a minimum of:

- T & R Qualification Name
- T & R Qualification Title
- The Media in which the qualification can be accomplished in – Aircraft, Simulator, or both
- Crewmembers who receive the qualification
- Valid currency periods of the qualification by ACTC Level, by crewstation
- The number of hours required to receive the qualification, and the annual flight-hours requirement for the qualification
- The point value of the qualification by PMA
- Notes
- Ordinance required for the qualification
- Resources required for the qualification
- Detailed qualification description and UNTL reference.

For the purpose of joint compatibility and mission accountability, each qualification in a particular TMS shall be referenced to one or more UNTL tasks. The goal of each squadron is to reach the highest level of readiness possible, taking into consideration the oversight and assets required for the qualification, requirements resources available, and the long-term mission objective of the command. The effects of "P4 + WARTS" pay considerable weight into the decision to execute some tasks or the option to wave specific qualification in favor of other tasks.

Due to the limited resources of most TMS, some commands will get into a dilemma where they will be "chasing their tail" to maintain readiness as depicted by Figure II.1.



Sym	Qual	Period
ϕ	ASW07	90 Days
ϑ	ASW12	60 Days
Π	STW05	30 Days
♌	STW07	45 Days
♋	STW07	45 Days
♎	NSW03	120 Days

Figure II.1 Training and Readiness Qualification Circle - Perfect

In the perfect model depicted by Figure II.1, each qualification is received on the first day of the cycle, and then renewed on the last day of the applicable readiness period. For example; ASW07 is received on DAY-0, renewed on DAY 90, DAY 180, DAY 270, and DAY 360, and so forth while NSW03 is received on DAY-0, renewed on DAY 120, DAY 240, and DAY 360, until the cycle repeats. While this model is optimal for squadron operations to maintain the highest level of readiness, it is highly impracticable. Due to a lack of resources, increased emphasis of operations tempo in a particular arena and a corresponding loss of emphasis in another, and the relationship driven factors of "P4 + WARTS", qualifications will slide. Some qualifications will be renewed before their assigned expiration date, operations will elect to not renew some qualifications until a later date, and others will be received on schedule, but to maintain some level of readiness, operations dictate that expiring qualifications be renewed – "chasing their tail."

No matter how hard a command flies, a qualifications will still expire and need to be renewed, forming a cyclic pattern of readiness.

The COMNAVAIRPACINST 3500.63 / COMNAVAIRLANTINST 3500.67 instruction is constantly under review from inputs by all TMS and TYCOMs.

3. OPNAVINST 3710.7

OPNAVINST 3710.7, Series R, Interim Change 27 – commonly referred to as NATOPS, or the Naval Air Training And Operating Procedures Standardization Manual, serves as the "Bible" for Naval Aviation. The latest interim change was released for issue by message on September 9th, 1999.[44] NATOPS is not only a manual or instruction for flight operations; NATOPS is a concept of operations. The background chapter of the NATOPS Manual states:

> "The Naval Air Training and Operating Procedures Standardization (NATOPS) Program is a positive approach toward improving combat readiness and achieving a substantial reduction in the aircraft mishap rate. Standardization, based on professional knowledge and experience, provides the basis for develop-merit of sound operating procedures. The standardization program is not intended to stifle individual initiative, but, rather to aid commanding officers in increasing their unit's combat potential without reducing command prestige or responsibility."[45]

The NATOPS Manual is broken up into eleven sections with additional appendixes as:

- Introduction
- Naval Air Training and Operating Procedures
- Standardization Program
- Policy Guidance

---

44 OPNAVINST 3710.7R, *"Naval Air Training And Operating Procedures Standardization (NATOPS) Manual"*, Interim Change 27, 09 Sep 1999.

45 OPNAVINST 3710.7R, *"Naval Air Training And Operating Procedures Standardization (NATOPS) Manual"*, Cover Page, 15 Jan 1997.

- Flight Authorization, Planning, and Approval
- Flight Rules
- Air Traffic Control
- Safety
- Aeromedical and Survival
- Miscellaneous
- Flight Records, Reports, and Forms
- General Instructions On Duty Involving Flying and Annual Flight Performance Requirements
- Classification and Qualification Of Flight Personnel
- Instrument Flight Requirements and Qualifications
- NATOPS Flight Personnel Training and Qualification Jacket
- Various abbreviations, codes, tables, and lists.

It outlines instructions, procedures, regulations, and limitations towards naval aviation operation. It provides a guideline for flight requirements and certification, as well as flight planning and safety. Naval operation and the T & R Manual are limited by the requirements of NATOPS. For example, NATOPS defines the minimum and maximum number of flight-hours a crewmember can log during a given time period, as well as minimum currency periods for some certifications.

Each TMS is required to publish an aircraft specific NATOPS manual to detail specific procedures, instructions, doctrines, limitations, and restrictions explicit to the particular type model. The type wing NATOPS is intended to be a more restrictive and distinct to the particular TMS.

4. OPNAVINST 3500.39

In an attempt to enhance naval operations safety, the Chief of Naval Operations and the Commandant of the Marine Corps authorized the release of OPNAVINST 3500.39 / MCO 3500.27, Operational Risk Management (ORM), on April 3rd, 1997,

addressed to all ships and stations. [46] It recognized that uncertainty and risk are inherent factors in military operations, but that naval forces must take new efforts to "balance risk with opportunity" to minimize the levels of risk while still meeting the obligation of assigned mission and tasks. In an effort to minimize risk, OPNAVINST 3500.39 establishes Operational Risk Management as an integral part of training, operations, and mission planning.

ORM is the management of risk by identifying potential risks and hazards, assessing the levels of the risks, identifying potential consequences of the risks, developing controls to mitigate the risks, and then implementing the controls during the earliest possible stage of operation or planning. The supervision of risk controls is the final and most important facet of ORM. Failure to manage risk and ensure compliance will render the ORM concept hollow. Information available through existing safety, training, and lessons learned data bases should be considered whenever practicable in making risk decisions.

To ensure full compliance with ORM, forces are to make every effort possible to train service members through leadership courses, General Military Training, safety schools, warfare qualification schools, and other applicable training courses. ORM shall be integrated into all aspects of tactical training, Personnel Qualification Standards, (PQS), Naval and Occupational Standards, Individual Training Standards, and the Marine Corps Combat Readiness Evaluation System.

By directive, Naval Aviation and the T & R need to fully comply with the concept of ORM. Flight operations cannot be expected to overextend crewmembers to achieve

---

[46] OPNAVINST 3500.39 / MCO 3500.27, *"Operational Risk Management"*, 03 April 1997.

qualifications beyond a safe level, nor can they expect crewmembers to achieve qualifications that they have not been properly trained for nor have assets to properly support the achievement of. ORM risk mitigation is directly effected by the factors set forth in the "P4 + WARTS" concept.

The OPNAVINST 3500.39 Manual requires a fleet review of the instruction no later then every two years. As of this thesis, no changes have been made to the instruction in the last forty months.

5. NWP 1-03.3

The NWP 1-03.3, the Status of Resources and Training System Joint Report-Navy (SORTS or SORTSREPNAV), is the principle report for Navy, Military Sealift Command, and U. S. Coast Guard units to provide general status data to the National Command Authority (NCA), the Joint Chiefs of Staff (JCS), the Chief of Naval Operations (CNO), members of the chain-of-command, and other operational commanders.[47]

The SORTS Report is designed for designated units to report their operational status and changes in their status directly do their chain-of-command. The SORTS report is composed in a machine-readable format, permitting its contents to be parsed directly into a centralized database, accessible by assets or echelons using the TRMS Afloat and TRMS Aviation systems. SORTS data includes:

- Transfer Data
- Organization Location Data
- Defense Condition (DEFCON) Data
- Nuclear Capability
- Deployment Status
- Organization Personnel Strength

---

47 NWP 1-03.3, *"Status of Resources and Training System Joint Report – Navy"*, Under review.

- Overall Status Ratings for Resources and Training
- Status Rating for Equipment and Supplies
- Status for Training
- Resource Level Limitations
- Major Equipment and Crew Locations
- PMA Status
- Mission Area Rating
- Primary Resource, Secondary Resource, Tertiary Resource, and Projected Status
- Special Capabilities
- Adequate Wartime Resources Available
- Shortages of Wartime Resources
- PMA Exceptions

In a classified format.[48]

6. SMART-R and the SMART Squadron

On February 9th, 1998, Acton Burnell released a High-Level Requirement document highly critical of the current Naval Air Squadron T & R System and the support infrastructure for managing the T & R.[49] The report outlined the growing problem of a lack of software support due to the increasing number of independent T & R software systems; the lack of solid requirements specifications and process models for T & R support; and for the lack of a data warehousing to archive flight operations data. This report was forwarded to OPNAV (N889) for evaluation and review. One of the outcomes of this review served as a spark to the concept of SMART-R.

On March 4th, 1998, the Naval Air System Command or NAVAIR released Synopsis No. 20062-98, seeking submissions for the development of the Aviation Squadron Management System Software.[50] In response to the request for submissions,

---

48 NWP 1-03.3, *"Status of Resources and Training System Joint Report – Navy"*, Chapters 5 and 6, Under review.

49 *"Naval Air Squadron T & R System: High –Level Requirements"*, Acton Burnell, 09 Feb 1998.

SAIC in team with ISYS, an Israeli Defense Contractor, submitted a proposed Operational Management System based on the Israeli Air Force Concept of Operations. In an attempt to evaluate the proposed Israeli system - SQOM, or the Squadron Operations Management System - NAVAIRSYSCOM stood up the Squadron Management, Automated Risk Tolerance and Reporting System (SMART-R) evaluation team.

NAVAIRSYSCOM was provided approximately $1.8 Million from the Secretary of the Navy's Quality of Life Fund to pay for the assessment and evaluation. In a recent telephone conversation with LCDR Darrell Lack, Program Manager for the SMART-R Project, he stated that "it would be a ridiculous use of money to investigate one project."[51] The assessment and evaluation of SQOM soon turned into an Analysis of Alternatives (AOA) of the four compatible systems: SQOM, the Boeing SARA Product, SHARP, and the Air Force prototype Patriot Excalibur System.

During its brief history, the SMART-R board had derived a set of aviation requirements for the design and development of an aviation operations management system. The most recent requirements list was released after an intensive meeting with members of both coast TYCOMs, 2nd and 4th MAW, members of the Naval Safety Center, OPNAV, N880, N881, and N889, using lessons learned from initial evaluations of all of the perspective management tools and the ever changing environment of Naval

---

50 NAVAIRSYSCOM, *"Request for Proposals: Synopsis: 20062-98: Aviation Squadron Management System"*, Commerce Business Daily, 04 Mar 1998.

51 Phone Con, 17 Jul 2000, LCDR Darrell D. Lack, SMARTR Project Manager, NAVAIRSYSCOM PMA-233 Naval Mission Planning, 301-757-8008.

Aviation. This new requirements list, titled SMART-R Criteria Definitions, dated February 23, 2000, included requirements for:

- Logistical / Life Cycle Support
  - User Customizable
  - Technical Support
  - User Training
  - Configuration management
  - Documentation
- Resource Management
  - Long Range Planning
    - Support Elements
    - Personnel
    - Weapons
    - Ranges
    - Simulators
    - Planes
    - Adversaries
  - Funding
    - Petrol
    - TAD
    - Parts
  - Integrated Scheduler
    - Provide Automated Flight Schedule Assistance
    - Provide Automated Maintenance Scheduling Assistance
    - Overall unit requirements
- Interoperability
  - Interfaces
  - Compatibility
  - Tailoring for Multiple Operation Modes
  - Accessibility
- Risk Management
  - Automated Risk Analysis
- Report Generating Capabilities
  - Provide Standardized Reports
  - Ad Hoc Queries
  - Electronic Training Jacket
  - Real Time Activity Tracking[52]

---

52 *"SMARTR Criteria Definitions"*, SMARTR, NAVSYSCOM PMA-233, 23 Feb 2000.

The intent of the AOA, as LCDR Lack put it, was to purchase something at the end of the evaluation, but political pressure, no true funding line, and a lack of authority would not allow for this kind of outcome. As of September 2000, the SMART-R Team will have expended its funds for the evaluation and will make its final recommendation to the AIRBOARD for action. LCDR Lack estimates the value of such a product to be worth little more then five million dollars per year, far below the cost proposed for existing products. With the exception of a Mission Needs Statement, the SMART-R board has no driving authority to purchase any system.

D. REVIEW OF THE SHARP SYSTEM SOFTWARE-ENGINEERING MODEL

1. SHARP Requirements

a. Training and Readiness Requirements

Initially, the primary requirement for the SHARP system was to engineer a product to support the newest release of COMNAVAIRPACINST 3500.63D / COMNAVAIRLANTINST 3500.67D, specifically;

- The entry and archiving of a unique T & R Matrix representing a particular TMS.
- The entry and archiving of a command's crewmembers and their supporting data.
- The tracking of valid qualifications by crewmember.
- The computation of a command's combat readiness.

While the initial scope of the requirements were limited to supporting and returning a command's readiness value, it quickly became evident that squadron users desired something that would better reflect the complete scope of squadron operations in

the electronic environment. For decades, Flight Operations Offices have relied on conventional practices to accomplish menial tracking and calculations. With the advent of technology and the distribution of IT-21 compatible computer units, it was time for flight operations to augment its efficiency by joining the information age and opening Naval Operations to the "Clicks and Mortar" environment.

The commercial sector has long touted the concept of updating traditional business practices from the "Bricks and Mortar" concept to the more technologically advanced and reliant concept of "Clicks and Mortar". "Bricks and Mortar" can be defined as the traditional business practice of non-electronic data transfer, and non-electronic commerce, housed in a conventional office space. "Clicks and Mortar" can be defined as the information age driven business practice of electronic data transfer, electronic mail, and electronic commerce, running in parallel to traditional business practices, housed in a conventional office space. Based on the commercial success of the "Clicks and Mortar" office concept, it would only seem a natural progression to adopt this practice into the military operations, specifically that of Naval Flight Operations.

In an attempt to better understand the requirements of Naval Aviation from the fleet perspective, a team of over thirty Pilots, Naval Flight Officers (NFO), and Aircrewmen were brought together in North Island, CA to form the Aviation Training Readiness and Requirements Information System (ATRRIS) Integrated Process Team (IPT). This team of aviators pooled their combined knowledge of naval aviation, operations, training, and readiness to author a complete list of requirements to model squadron operations and revolutionize the operations office. The ATRRIS IPT attempted

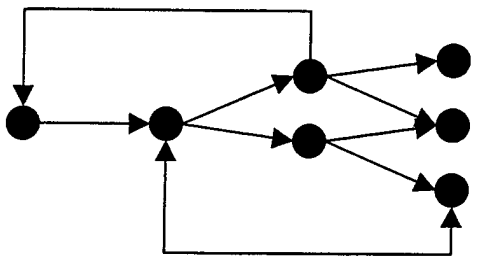


Figure II.2 Cause and Effect Reliance

The second type of requirement change, the "Trickledown Reliance" results when the actions, values, or theme of one requirement or its resulting values directly effects the resulting values of another requirement through a set of logic statements or calculations. While the "Cause and Effect Relationships" can relate one requirement to the next on the same level or on variable levels, the "Trickledown Reliance" would relate a senior requirement down to one or more subservient requirements. While the "Cause and Effect Relationships" directly relates values from one requirement to the next, the "Trickledown Reliance" mandates subservient requirements rely on the values and actions of senior requirements through logic, as depicted in Figure II.3. An example of "Trickledown Reliance" would be the fact that:

- Flight Night Vision Device Time must be less than or equal to a flight's Total Nighttime.
- Flight Total Nighttime must be less than or equal to a flight's Total Flight Time.
- Astronomical conditions must be such that a flight occurred during Nighttime.
- A crewmember must be capable of logging Night Vision Device Time.
- A crewmember's individual Night Vision Device Time must be less than or equal to his own Nighttime.
- A crewmember's individual Nighttime must be less than or equal to his own Total Flight Time.

- If the flight's Total Flight Time decreases it may affect an individual's Total Flight Time, depending on the initial Flight Time of the individual, which would trickle down to affect the flight's and individual's Nighttime and Night Vision Devise Time.

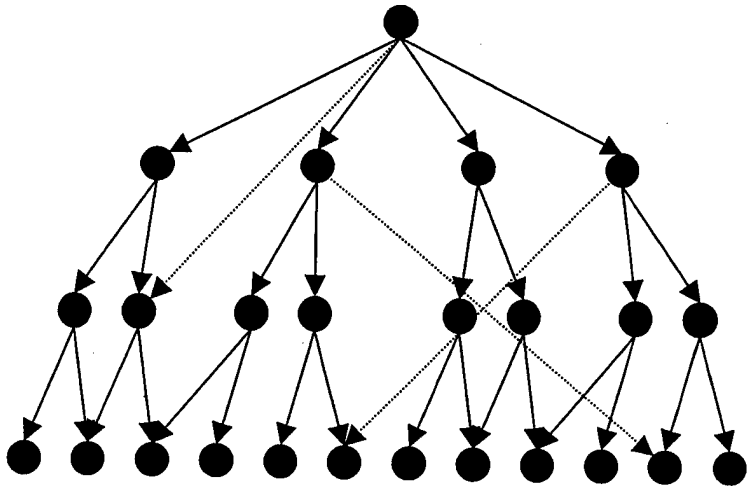


Figure II.3 Trickledown Reliance

Due to the great reliance or relationship of one requirement on the next, changes to one requirement can quick develop into a ripple effect through the SRS. Proper tracking and SRS documentation is required to monitor the changes and ripple progression.

Mr. Chris J. Date's recent book on Software Development refers to the logic of Trickledown Reliance as a Business Rule.[53] Mr. Date simplifies that the requirement and design of software systems should be based on what needs to be accomplished, vice how it should be accomplished. These "What" actions combine to build a set of business rules that outline the development of a system. The Trickledown Method relies on established business rules to define the methodology for the operation of minor systems, based on the "What" requirement of major systems. The concept of

---

53 C. J. Date, *"What Not How: The Business Rules Approach to Application Development"*, Addison-Wesley Publishing Co, 07 Apr 2000.

business rules is not new to the software engineering process, in that it has been a tool regularly used by managers to ensure the business process of development was accomplished in compliance with established function rules. The new application of business rules to the software engineering process down to the actual development level refines the methods that used to bind the creative process. While Mr. Date's book is based around the theme of databases and corresponding business rules, the concept and methodology can be expanded and applied across all aspects of software engineering.

c. The Requirement Shift

One of the more recent changes in the UNTL and its support documents was the reduction in ASW missions to the S-3B airframe. Due to the reduction of missions and the composition of the S-3B airframe crew, a number of changes were required to the T & R Instruction, resulting in changes to the SHARP Requirements.

The COMNAVAIRPACINST 3500.63 / COMNAVAIRLANTINST 3500.67, Squadron Training and Readiness Report, is revised approximately every eighteen months. As the SHARP program is designed to directly support the T & R Instruction, any changes to this instruction results in immediate changes to the SHARP SRS. The T & R Instruction is based on the inputs of the seventeen different type model series airframes and their corresponding Wing Training Manuals (WTM). WTMs are changed at periodic intervals, independent of each other. Knowing that the T & R Instruction and related WTMs would influence multiple programmatic changes, attempts were made to generalize SHARP requirements through logic statements, in an attempt to build a basic format of the concept of operations with customization. Once the basic concept of operations was mapped, changes to T & R and WTMs values could then be

inserted into the logic map of the requirements and, respectively, the code. Primary values within the T & R Manual were described previously in this chapter. As long as the basic concept of operations did not change, no real change had to be made to the SRS, but a requirement did fall on the user to make the customization to the SHARP Program in operation.

The initial paragraphs of the T & R directly outline the concept of operations for Naval Aviation. These document changes result in changes to the concept of operations resulting in changes to the SHARP SRS.

d. External Requirement Factors

As stated previously, the OPNAVINST 3710.7 – NATOPS serves as the aviation "Bible" of operations. NATOPS receives major revisions approximately every thirty-six months. The NATOPS Manual also receives interim updates on periodic basis. The current NATOPS Manual has received twenty-seven interim changes since its release three years ago.[54] NATOPS directly maps the concept of Naval Flight Operations and, correspondingly, its requirements. Any changes to the NATOPS concept of operation results in immediate changes to the SHARP SRS.

The OPNAVINST 3500.39 - Operational Risk Management Manual is scheduled for review every two years. The ORM Manual does not specify any rules for ORM, does not contain any models for making new rules, nor does it specify a process for determining specific rules. The ORM Manual does mandate that units should institute a form of operational risk tools into their business process. Due to a lack of official

---

54 OPNAVINST 3710.7R, *"Naval Air Training And Operating Procedures Standardization (NATOPS) Manual"*, Interim Change 27, 09 Sep 1999.

guidance, the ATRRIS IPT combined their knowledge of operational risk management from the squadron level and coupled it with standard operating procedures and personal flight schedule writing techniques to create a requirement for ORM. This consolidated requirement resulted in a move by the SHARP Group to build more than an Operational Risk Management tool, but instead to compose a requirement to build an Operational "Requirements" Management tool. Operational Risk Management has the implied outcome of reducing risk by removing risk. Risk is reduced by removing an individual from a particular task that he might not be qualified for, or by removing a particular task from a schedule. Operational "Requirements" Management implies managing the requirements for tasks by ensuring proper scheduling and management of personnel. If an individual was not qualified for a particular flight, task, mission, or qualification then he might be added to the flight scheduled to get qualified. If an individual does not fly, he would never get qualified. Many aviators regard Operational Risk Management as a punitive tool, while they view Operational "Requirements" Management as the standard concept of operations.

Until the Navy or Naval Aviation releases specific instructions on how to build ORM models, the SHARP SRS will continue to be based on a fleet perspective of ORM.

The NWP 1-03.3 – SORTS Manual is updated approximately every thirty-six months. The only data-point that the SHARP system provides to the SORTS Report is the aviation T & R Value by PMA. The requirement to generate the T & R Value is outlined in the T & R Manual. Pending on any additional requirements from the SORTS Manual, no further changes are predicted to the SHARP SRS.

The SMART-R Requirement Document outlined the SMART-R's interpretation of squadron operations and ultimately the concept of operations. The points and findings of the SMART-R were outlined previously in this thesis. The SMART-R added a new perspective to the SHARP Requirements List by incorporating the inputs of the higher echelons of the DoD as well as cross service operations. Previously, the SHARP SRS was based on the inputs of squadron users and their respective TYCOMs. The predominant voices at the SHARP Requirement Meetings were potential SHARP users or system clients. The SMART-R combined members from across the Navy and Marine Corps management level (OPNAV and HQ) with members of senior members of the aviation community (CNAP, CNAL, and MAWs) to develop a top-level view of squadron operations. This requirement list[55] was then compared with the existing SHARP SRS to ensure completeness and potential compliance in the event that the SMART-R Concept became a program of record. At this time, due to a lack of future funding lines, we do not assume the SMART-R will publish any new requirements.

After a review of all of the SHARP requirements, the team attempted to create a mapping of related subjects or areas of interest. This map was broken down by each of the primary functional Naval Operations subject areas, as:

- Personal Management
- Operations Management
- Aircraft Management
- Fuel Management
- Weapons Management
- Adversary Management
- Range, Route, and Airspace Management

55 *"SMARTR Criteria Definitions"*, SMARTR, NAVSYSCOM PMA-233, 23 Feb 2000.

- TAD Management
- Reports Generating Capabilities
- Astronomical Data
- Customization
- Program System Support

As the map was developed, each of the requirements were assigned to a primary function of the map, then broken into secondary level, tertiary level, and so on, until requirements were driven to their lowest level possible, resulting in a detailed requirements tree. After a review of the requirements map, it became evident that we would not be building one system, but a number of small, interrelated systems that would incorporate themselves together to meet the total requirements of Naval Aviation.

After gathering all of the requirements from the governing documents and ATRRIS IPT meeting members, a small group of aviators and developers worked to develop a brief framework of the SRS and initial prototype models of the perspective system. A specific discussion of the requirement gathering techniques, risk analysis, relationship and reliance, will be presented in the methodology section of Chapter 3 of this thesis.

2. SHARP Engineering and Development Model

As illustrated in the previous section, Section II.C – *"Review of DoD Readiness Requirements, Specifically Naval Aviation"*, it was demonstrated that the requirements for Naval Aviation Training and Readiness were, and remain, extremely fluid. After a review of many the standard development models, exemplified in Section II.A – *"Review of Traditional Software Engineering Techniques"*, it also became evident that no conventional model in its entirety would suffice the development of the SHARP system.

The Build and Fix Method of software development requires that the system be built, reviewed for completeness, and then rebuilt, until the resulting product suffices the needs of the client. Most cyclic development processes utilize some sort of Build and Fix Approach, or its basic framework, to manage the project development. In its basic form, without any real defined requirement or testing stage, the Build and Fix Method was not even considered. I wish to reiterate that the Build and Fix Method serves as the basis for most cyclic models, and its concept will be demonstrated to be useful later in this section.

Like the Build and Fix Method, the Stagewise Development Model also serves as a basis for many of the formal process models, in that it defines the development process by breaking it down into stages or incremental steps. The SHARP product needed a model that would formalize its developmental process by defining the steps or stages in which it should be built. As the Stagewise Model was only an incremental step to designing a more formal development model, it did not meet the defined needs of the SHARP Team.

Due to the fluid requirements defined by the SHARP SRS, the Waterfall Development Model was immediately removed from contention as a developmental process. The Waterfall Model mandates that requirements be solidified at the beginning of the developmental process and that new requirements would not be viewed until the end of the entire process.

The Waterfall Model is mirrored by the Test Development Model, with the additional aspect of Test Plan Authoring running in parallel to each of the development stages. Due to its top down development flow and inflexibility to changing requirements, the Test Development Model would not meet the needs of the SHARP Development.

The concept of Test Plan Authoring running in parallel with system development was found to be very beneficial, and was extracted to application to the eventual SHARP Development Model.

The Prototype Model introduced the notion of prototype development prior to complete system development, as a part of the Waterfall Model. As previously mentioned, the Waterfall Model was ill equipped to facilitate the SHARP Development Project, but the idea of a working prototype was found to be very beneficial to demonstrating the validity and completeness of the SRS. The concept of Prototype Development would become a key facet in the engineering of the SHARP Project, as discussed later in this chapter.

The Exploratory Programming Method would be one of the first models that introduced the client into the development process. With Naval Aviation Squadrons as the primary clients to the SHARP Product, and Naval Aviators as members of the development team, the primary element of a close client-developer relationship with Exploratory Programming had been met. As the Exploratory Approach still required formal reference to other fundamental stages such as Prototype, Requirements Gathering, Risk Analysis, Testing to be a complete model, it did not meet the needs of he SHARP System, but the concept of a close client-developer relationship would remain for the ultimate development model.

As mentioned in the conclusion of requirement discussion, the SHARP SRS had broken the product down into many modules vice that of one large system. The Incremental Model of development discussed the concept of developing modules of a system incrementally until a system was complete. As many of the requirements of the

SHARP System were not finalized until later in the program, it would seem more efficient to develop some of the modules before others, in an incremental approach. This method would also be rolled together into the final SHARP Development.

With the intent of taking advantage of previously designed systems, either through the improvement of legacy systems or the introduction of second party systems, logic dictates the use of the Legacy and Reuse Software Life Cycle Model. Due to the strict and short deadline placed on the SHARP System, developers were intent on taking advantage of as many predefined systems as possible to reduce the overall effort on the project. While the SHARP System had never been developed before, a number of other like systems had come close to meeting some of the requirements. As these systems were not designed to be up-line compatible, due to system compliance issues and logic concerns, these systems did serve as partial prototypes, and the SHARP Team was able to model and extract some requirements. While some systems were working operating programs, other systems existed solely as logic statements. For example, the SLAC Astronomical Prediction System developed by the Office of the Naval Observatory in Washington, D. C., modeled the mathematical logic for predicting the all solar and lunar phenomena. Logic such as the SLAC Program could be quickly absorbed into the development of the project. While the Legacy and Reuse Model has some benefits to the reuse of modules, it does not completely map all of the other required aspects of software development.

The most efficient method for developing a project with fluid requirements, multiple modules, and incorporating incremental design would be the Spiral Model. The Spiral Model permits a more flexible development process by introducing a stagewise

development that permits the redefining of requirements, reassessing of risk, and the redevelopment of the product in a cyclic fashion. The Spiral Model of development most closely matched the needs of the SHARP Project, with the exception of some key items found in other models: Test Plan Authoring, Close Client-Developer Relationships, Reuse, and Parallel Development with Modular Design.

Ultimately, the development model that was chosen for the development of the SHARP System was to be a hybrid of many of the concepts seen in previous models. The model consisted of the stagewise development steps of:

- Initial Requirements Gathering Stage
- Module Breakdown and Task Assignment Stage
- Solution Search Stage
- Risk Analysis Stage
- Prototype Development and Engineering Stage
- Evaluation Stage
- Requirement Certification and Requirement Search Stage
- Module Breakdown and Task Assignment Stage to repeat the cycle

From the basic view the only two additions to this model over the traditional Spiral Model would be the incorporation of the Module Breakdown and Task Assignment Stages, and the Solution Search Stages. From a deeper inspection, it becomes relevant that this model is more then a two-dimensional spiral, but rather a three-dimensional spiral that builds up like a wedding cake with each successive cycle, hence for the purpose of this thesis, it shall be referred to as the Wedding Method.

a. Initial Requirements Gathering Stage

The first step in the Wedding Process is to make an initial search of the product requirements. The requirement search does not have to be a complete search and definition of all requirements, but should include the basic theme of the system. The initial requirement search should outline the primary objective of the proposed system. It is not necessary to establish all of the constraints of the system on the first iteration. The interaction of product clients, subject matter experts, and developers is essential to ensure an accurate authoring and understanding of the requirements. Some level of test documentation should be accomplished in parallel with the requirement gathering stage, as subject matter experts will be readily accessible to define the potential goals of the test. Once the initial search is complete, the sets of requirements are defined, and a partial Test Plan is started, requirements need to be grouped in to system modules.

b. Module Breakdown and Task Assignment Stage

Each of the systems modules should be defined as major actions or functions of the system, like a System Management Module, Configuration Management Module, Reports Generation Module, Primary Operations Module, and applicable Secondary Operations Modules. Each of the system modules should then be subdivided into as many systems sub-modules as possible to develop a hierarchical tree of modules and their associated sub-modules. After the requirements and system have been broken down into their associated modules, the next task would be to determine which modules and sub-modules should be started on the first iteration. This is where the Wedding method diverges into multiple layers for development on an independent basis, building tier after tier, as depicted in Figure II.4.

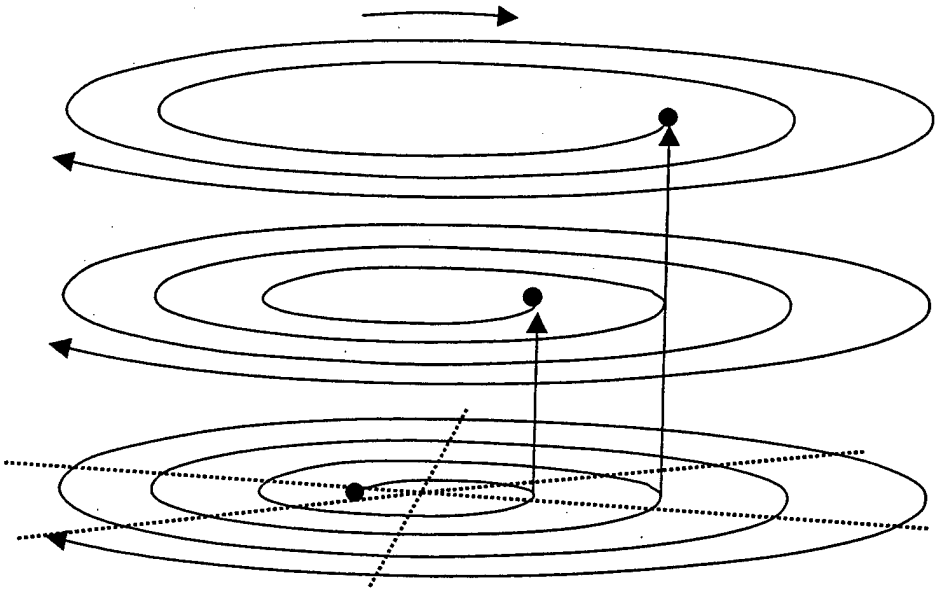


Figure II.4 Wedding Model with diverging Cycles

To best model the level of effort for module and project development, functions, tasks, and requirements should be broken down to their lowest possible level. If a function requires a level of effort of "2", then the function should be broken up into two equal sub-functions, each with a level of effort of "1". If a particular module has fifty-two functions and a total level of effort of "175", then the functions should be broken into 175 total functions. Such an equitable function breakdown permits easier evaluation and task assignment.

The tasks should be selected in the first iteration such that they form the backbone of the system and can assist in demonstrating a functional prototype for demonstration and project evaluation. Where possible, when requirements are in need of greater definition through prototype evaluation, these tasks should be selected such that their prototype can be evaluated to complete requirement specifications. The number of tasks should be determined by the amount of man-effort that can be given to the project and the man-effort that would be required to complete each of the sub-modules or tasks.

c. Solution Search Stage

Where possible, a search and evaluation should be made of legacy and second party systems that could meet the requirements of proposed first iteration tasks. This search should include systems that could be incorporated directly into the proposed system, working systems that can be used for requirement definition serving as a set of prototypes, or system logic that could be used during the development stage. If through observation and evaluation, new requirements are found in the Solution Search Stage to be beneficial to the system, then they should be fully documented for inclusion with the next cycle of development. Any changes, refinements, or edits to requirements due to second party systems should also be documented to ensure their inclusion into cyclic development. As part of the solution search, Test Plans and documentation should be authored from finding of potential solutions.

d. Risk Analysis Stage

The risk analysis stage of the Wedding Model closely resembles traditional risk analysis methods with a few complex caveats. Wedding Risk Analysis includes an assessment of development risk to the entire system, as well as a specific assessment of the sub-modules or tasks under development during the current cycle. If second party systems are to be utilized, an assessment of those alternatives should be evaluated for development risk over the engineering of new systems. Upon the completion of the Risk Assessment Module, a detailed list of risk controls should be authored. The first authoring should include the risk controls for the entire system development. The second authoring should include the risk controls for the sub-modules or tasks currently under development in the present cycle. These risk controls should be

incorporated into the Test Plan document to assist in final test and evaluation of the cycle. These two control documents would be forwarded to the Prototype Development and Engineering Stage to control the build, to the Evaluation Stage to assist with system testing, and to the next Requirement Assessment Stage to ensure compliance with the next cycle of development.

e. Prototype Development and Engineering Stage

As the first cycle is to be actually developed, sub-modules and tasks are to be assigned to development teams for programming. The development teams need to map the interrelations and reliance between the different tasks to ensure cross compatibility and communications exist once the different sub-modules are combined. As each of the tasks is developed, consideration should be made to findings in the Risk Analysis Stage. As each sub-module is developed independently, it is essential that an accurate Test Plan be mapped to ensure a complete representation of the development and evaluation of the system. The completion of the Development Stage would be the combination and linkage of the sub-modules for the purpose of composing a working prototype or operating system. Proposed prototypes, second party systems, and existing sub-modules would be combined to complete the Engineering Stage.

f. Evaluation Stage

The Wedding Method Evaluation Stage incorporates all of the new sub-modules developed, second party systems, and existing systems modules together into one complete working system for evaluation and testing. Evaluation and testing is completed against the Test Plan developed in the subsequent stages of the existing cycle, in series with the Test Plan developed in previous cycles for modules incorporated to the

current complete system. It is essential that every test element be re-tested with each iteration of the cycle to ensure the linkage and cross compatibility of all existing sub-modules or developed tasks has not corrupted the system as a whole. Intimate cooperation and interaction with subject matter experts, developers, and clients is essential to ensuring an accurate evaluation of the system against requirements or intended development outcomes. Test and evaluation should also evaluate the Development Risk Management Plan to rate its accuracy and applicability to the development process. Findings from the evaluation of the risk management plan are then passed back to the risk assessment stages in subsequent cycles as "Lessons Learned." An evaluation of applicable requirements against the existing product is the final element of the evaluation stage. Requirements are validated against the system to determine if the system is being developed to complete satisfaction. If new requirements are found through the evaluation stage, they are then posted to the next cycle of the Requirement Search.

g. Requirement Certification and Requirement Search Stage

As the cycle completes its revolution, it expands outward one ring but returns to the Requirement Stage of development. Requirement modifications from the previous cyclic development of the Solution Search Stage, Risk Analysis Stage, Engineering Stage, and Evaluation Stage are consolidated. Completed sub-modules and tasks are certified for completeness or evaluated for redevelopment. New requirements are sought out or discovered and added to the SRS. As with the first cyclic iteration of the Requirement Search Stage, Requirements are then distributed through the module tree for task assignment.

h. Second Module Breakdown and Task Assignment Stage

In the second and subsequent round of Module Breakdowns, each level of the Wedding Cake should continue to be evaluated as independent projects. Where necessary, levels could be further divided to better engineer the system. As depicted in Figure II.5, the Wedding Model is more then a two-dimensional incremental spiral, but a spiral with cross connecting spokes that permit information and design to move across stages as well as progressively forward.

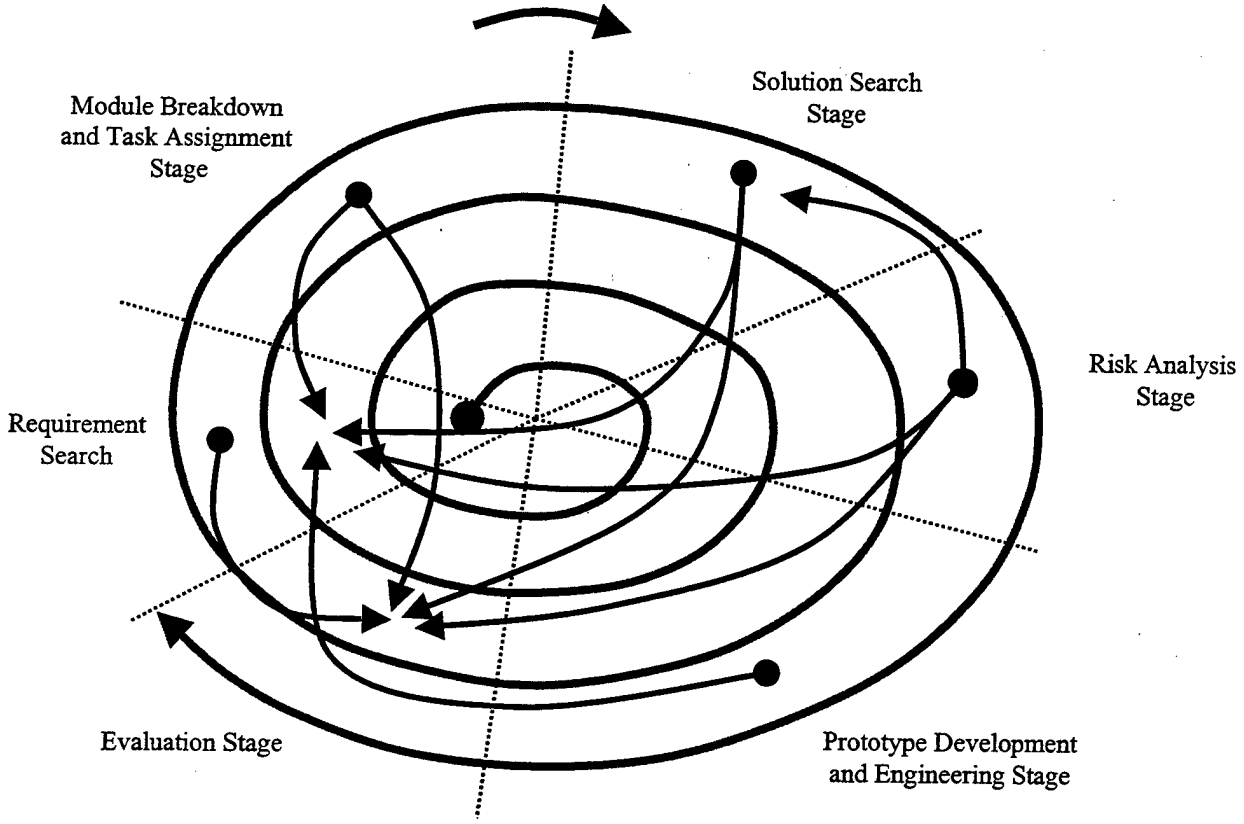


Figure II.5 Wedding Model with Spokes

i. The Solar System

The traditional Spiral Model dictates that each of the stages of development occur sequentially, in a two-dimensional plane. The Wedding Model breaks from the traditional concept by incorporating a radical design of multiple

independent modules, revolving around the same axis, at irregular rates, similar to the planets of the solar system that advance at independent rates, as depicted in Figure II.6. The Wedding Model has a number of inherent risks that have to be managed and controlled to ensure the success of the development. With each cycle of the system, independent modules of the development have the potential of falling out of sequence or rate with other modules. This freedom to accelerate or decelerate through the development process at independent rates adds to the beauty of the system, but also requires a strict management of interoperability links, cross development communications of Test Plans and Requirement Changes, and the monitoring of system progress as a whole.

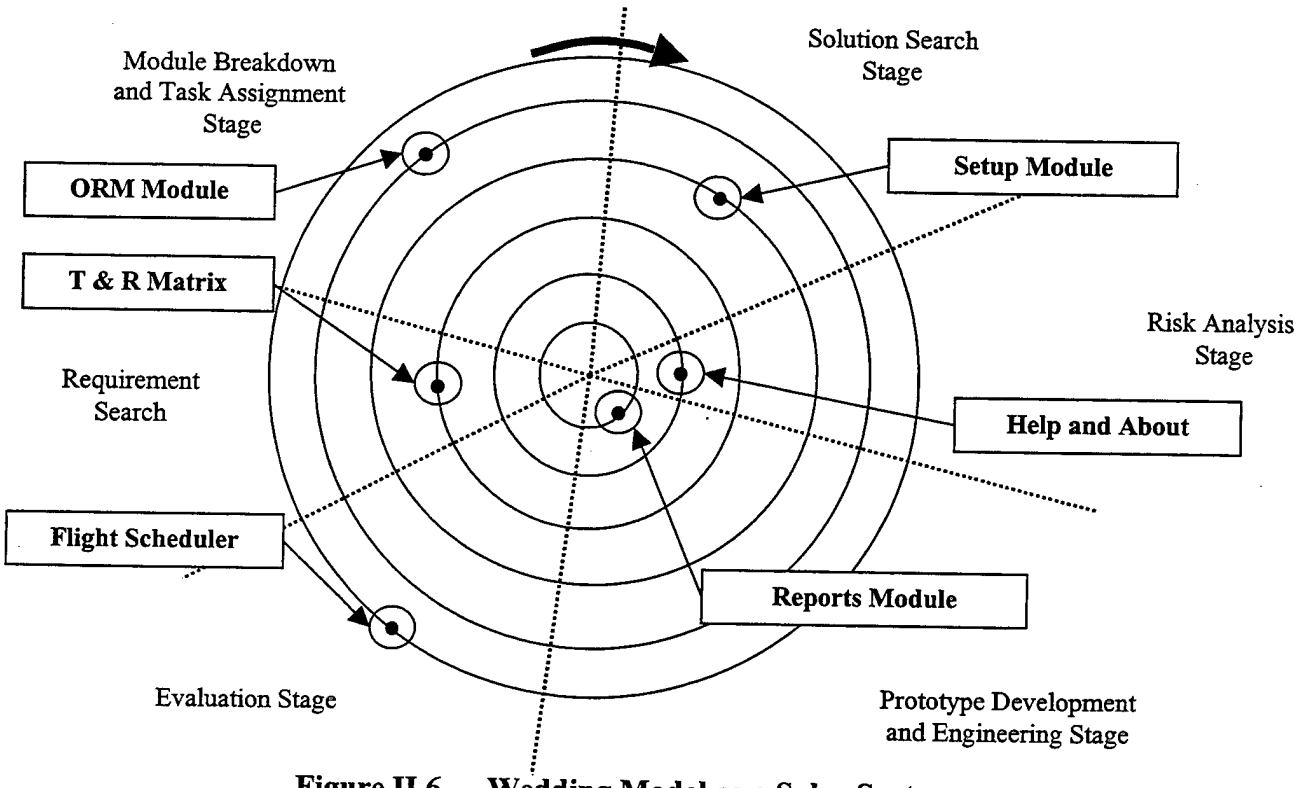


Figure II.6 Wedding Model as a Solar System

The independence of process progression actually increases the overall efficiency of the system. Some modules and their related sub-modules may require little

effort for development while other more complex modules would require a greater amount of effort. This module permits many smaller modules to be completed in a number of rapid development cycles, concurrent to the development of one larger, more complex module.

E. CHAPTER SUMMARY

The prominent author and philosopher, James Burke, stated that "the way for us to better manage change is to become acquainted with the interactive process *of change*."[56] This acquaintance to the interactive process requires an understanding of the foundations of development and not simply just a surface knowledge of the methodologies. For true growth and understanding, one must find the beauty in the method and not just the outcome - the outcome will most always be reached, it is the method that becomes a signature of the development. The development of the SHARP Project was not simply a cookie-cutter process, but the molding and refining of various methods and models to produce the successful outcome. To best arrive at this outcome, a number of processes had to be reviewed and understood. This thesis reviewed many of the formal methods of software development, DoD software engineering requirements, the Training and Readiness Requirements for building the project, and ultimately the actual method used to build the SHARP System.

Many engineers and managers relate software engineering methods to the Goldilocks Syndrome. When Goldilocks invited herself to sit town at the breakfast table of the Three Bears and partake of their porridge, she found one to be hot, one to be too

---

56 James Burke, *"The Pinball Effect: How renaissance water gardens made the carburetor possible – and other journeys through knowledge"*, London Writers Ltd., London, England, 1996, p. 6.

cold, and one to be just right. Many managers turn to software development and say that one method is too hard, one is too relaxed, and one is just right, without ever evaluating whether they should use their knowledge to take a little from each bowl of porridge and make it just right. For the purpose of the SHARP Project, the development looked at many of the software engineering method and found them all to contain some beneficial traits but also lacking some of the more intricate methods required for the particular product at hand.

The Build and Fix Approach offered a basis for the client-developer relationship through repetitive system review, but lacked any other formal method for development such as testing, requirements analysis, or integration.

The Stagewise Development Approach served as the basis for most other development methods, introducing the incremental approach to design.

The Waterfall Model was the first layer up from the Stagewise Development Approach, introducing a non-return flow of development from requirements to integration. Its logical process of development serves well for fixed-structure systems, but does not buoy well in projects with loose requirements, flexible design structures, and its absence of a system prototype.

The Test Development Approach was the second layer up from the Stagewise Development Approach. Its concept of authoring a formal Test Plan in parallel with the development process provides the test and integration teams with an accurate depiction of what the system should do, based on how it was designed. Unfortunately, due to its inflexibility as based on the Waterfall Model, it would not meet the fluid demands of the SHARP Development.

The Exploratory Programming Approach was similar to the Build and Fix Approach with the addition of a Use Stage in the development process that permitted the client and developer to utilize the system as part of the engineering progression. Such a method would work well in an environment where the developer and client serve side by side and have the opportunity to observe the product in action. As the Exploratory Programming Approach lacked any other formal process methods such as requirements, testing, and integration, it would not fully aid in the development of the SHARP System.

The Prototyping Model contained one of the important keys to the development of the SHARP System and could potentially assist in the definition of requirements. The use and development of a Prototype early in the engineering stage would permit clients and developers to better evaluate the requirements and logic of the system under design. A prototype has been found to reduce project risk and improve overall efficiency. Unfortunately the Prototype Model is an adaptation to the Waterfall Model and brings along some of the inflexibility traits that would hinder the development of SHARP.

The Incremental Model includes another important trait – The development of the project in increments or small modules. The development of incremental modules vice the development of an entire system significantly improves the engineering process by permitting stagewise evaluation with each increment of development. The Incremental Model would serve as the precursor the Spiral Model and eventually to the final process of choice of the SHARP System.

The Spiral Model, with its incremental method of Planning, Risk Analysis, Engineering, and Evaluation provided the keys to developing a system with fluid requirements. The Spiral Model still required some additional facets or stages of

development to meet all of the engineering requirements, but the Spiral Model's base structure would serve well as a platform for the ultimate design method.

The Legacy and Reuse Software Life Cycle Approach noted the use of second party systems into the overall design of a system. This approach serves well when second party systems are designed from the onset to be integrated into other systems. In attempt to improve the efficiency of the SHARP Development Process, the use of legacy systems had to be investigated. The ultimate design of the SHARP System would incorporate many of the Reuse Software Life Cycle Approach methods, in series with other beneficial Approaches.

After a review of the prominent development methods and the strict timeline that the SHARP Team constrained by, it was evident that a further investigation of alternative methods was appropriate. The theory and concept of Rapid Application Development provided the tools necessary to meet the SHARP Development Requirements. A complete RAD design, as discussed in this chapter, incorporated all of the required stages in an efficient cyclic design. The combination of these techniques, with a powerful prototyping and development language, and competent management would provide an impeccable model for building the SHARP Project.

A determination was made to integrate Pseudo-Visual Languages in the development process due to their ease of use, wide acceptance and employment in the commercial sector, and ready supply of talented developers. A brief description of Visual Languages was given with examples of commercial products.

One of the newest development techniques to gain attention is Extreme Programming or XP, popularized in numerous technical journals and papers. XP will be

used as a comparison and contrasting point against the final method of development for the SHARP Project. XP is based on an alteration from other RAD models to develop a new formal model of spiral development. After a review of the literature and critiques, it was evident that XP was a re-wrapping of existing techniques, with the incorporation of a number of juvenile-like methods and analogies.

When looking at the Department of Defense as a company, it mandates many rules, regulations, and stipulations directed to the design and development of software applications. This chapter reviewed many of the current and expired requirement specifications to generate an understanding of the motivation of DoD Software Engineering. This chapter also reviewed the lack of authority and oversight that many of the DoD directives hold to systems like the SHARP Project.

DOD-STD-2167A was one of the first standards to establish a common methodology for software development within the Department of Defense. It outlined each of the phases of development and documentation, but failed to give an outlet for non-standard development. Its inability to "pinball", as Mr. Burke noted, was its demise.

DOD-STD-7935A served as a document template library for design and development of Automated Information Systems. Like its mate, DOD-STD-2167, DOD-STD-7935A was found to be too inflexible fit into the fluid development process of today's engineering requirements. Many companies desired to incorporate custom design techniques and documentation, which would not be compatible with DOD-STD-7935A, resulting in its cancellation.

MIL-STD-498 was designed as a consolidation tool to join DOD-STD-7835A with DOD-STD-2167A and to provide a baseline DoD Standard for incorporation with

commercial standards. 498 informalized the development process by incorporating more autonomy at the team level, introducing a pseudo Spiral Model, and the removal of many formal review processes. It was later cancelled by the commercial method it was intended to introduce.

ISO/IEC 12207 introduced commercial method to the DoD establishing a life cycle development process, flexible approaches to CASE tools, the incorporation of ISO 9000 development techniques, and the procedures for developing outside of the existing standard. While 12207 had a large number of positive methods and tools, it had no jurisdiction requirements that mandated its implementation. Without it, the standard had "no teeth" in the DoD arena.

DII COE mandated a wide broad sweeping array of standardization changes to the development of DISA products. DII COE required a detailed certification criteria of all submissions, as well documentation and security protocols for development. Due to the limited authority of the development process, DII COE has the potential of becoming just another "stovepipe" requirement. These points were verified in a phone interview with the Chief Engineer of the Project.

DOD-REG-5000.2-R mandated requirements for the acquisition of major systems for the Department of Defense. This regulation would control the acquisition, definition, structure, design, review, and reporting of projects from the contractor to the fleet. While this regulation was very detailed and apparently beneficial, it was apparently over-burdensome to many contractors who would make attempts to downgrade their projects to ensure that they were not classified as "major systems."

The final directive that was reviewed in this thesis was the DITSCAP or DOD-DIR-5200.40, which established an accreditation process DoD IT systems. DITSCAP development covered project Definition, Verification, Validation, and Post Accreditation. While the DITSCAP Directive outlines a very beneficial process for system engineering and accreditation, it is not widely used by contractors or even DoD Developers.

While this chapter attempted to review many of the more prominent directives and instructions for DoD IT System Development, it by no way includes all of the potential regulations that could control such products. This chapter attempted to demonstrate that there are a number of countering and competing directives that could control the development of IT Systems. Many of these directives had great intentions, processes, and methodologies that could benefit system development, but they also lacked the compelling authority to induce developers to use them. This lack of compelling authority resulted in nothing more than, as the Chinese refer to as, "Paper Tigers."

This chapter also served as an introduction the requirements of Naval Aviation and the number of directives that combined to drive Combat Readiness. A brief description was made of the UNTL and of its reference to the missions and tasks of the Naval Forces. While the UNTL is not of direct relevance to combat readiness, its indirect effects upon many of the other requirement documents is sweeping.

One of the documents affected greatly by the UNTL was the T & R Manual or 3500 Series Instructions. The T & R served as the basis for the requirements of the SHARP System. The T & R also outlines the procedures for determining readiness at the command level. The 3500 Instruction was also affected by the requirements of all 17

different TMS Airframes, Wing Training Manuals, and Wing SOP. While the 3500 Series Instruction established a bedrock of Training and Operations, it is based on a fluid set of referencing documents that requires any resulting software support system to be flexible and adaptable to the changing needs of operations.

NATOPS or the OPNAVINST 3710 also serves as a guiding standard Flight Operations. While the T & R serves to establish a directive for the execution of specific missions and tasks, NATOPS outlines guidance for general flight operations with an emphasis on standardization and safety. As with the changes of the T & R, NATOPS has gone through 27 interim changes during its first 33 months.

OPNAVINST 3500.39, the Navy's ORM Manual, required all naval forces to utilize Operational Risk Management techniques in the planning and execution of operations. The ORM Manual lacks any specific guidance for the development of ORM Rules, but burdens commands with developing rules based on their own concept of operations.

NWP 1-03.3, the SORTS Manual, provides guidance for the reporting of unit readiness. The only direct requirement that the SORTS Manual mandates from the SHARP System is the reporting of the single readiness values for consolidation with other readiness indicators. The SORTS Manual dictates that the readiness values are classified, which in turn classifies the output of the SHARP System and its readiness value.

For the last two years, OPNAV had been leading an effort to define squadron operations IT requirements through the evaluation of independently designed competing software system tools. N889 was eventually commissioned with the SMART-R project,

with funding from the Navy Quality of Life Initiative, to evaluate potential software systems. One of the competing systems was the SHARP System. The intended outcome of the SMART-R Evaluation was to recommend a proposed system for acquisition to the AIRBOARD for purchase. Due to a lack of funding and real authority to mandate any acquisition, the SMART-R Group will complete its efforts in September 2000 with its closing recommendations. While the SMART-R effort was established to evaluate multiple systems, it also served to refine and strengthen the requirements of the SHARP System through competitive cross comparison.

The last portion of this chapter was dedicated to the specific development of the SHARP System. Recently, Naval Aviation has made the gradual transition to the "Clicks and Mortar" environment by the introduction of IT Solutions to complex requirements. A review was made of the fluid requirements and different models to develop the solution set. A discussion was made of the theory and logic behind the SHARP Project, including the "Cause and Effect Relationships" of modules and data, the "Trickledown Reliance," and logic of other modules and data to develop the resulting system. The final outcome of this evaluation and discussion was the introduction of the final development system for the SHARP Project and the Wedding Cake Model. The Wedding Cake Development Model is based on the Spiral Development Model with multiple layers or diverging cycles, and cross stages links or spokes, and independent development speeds of progressions.

As demonstrated by the requirements and by an evaluation of previously developed systems, it is evident that the required SHARP System had to be fluid, modular, and easily customized. The SHARP System was not to be developed as a

stagnant system but rather a constantly changing prototype that would reflect the current and future requirements with each new cycle of development. Such a development could only be accomplished using the prototype development technique discussed earlier in the chapter – the Wedding Cake Method.

III. METHODOLOGY

A. REVIEW OF PROCEDURES FOR SEARCH AND DISCOVERY OF EXISTING MODELS

Optimally, a development team is tasked with the design and engineering of a specific software IT solution system by members of their parent company's management or military Chain-of-Command. The development team's parent would task them with general requirements and direction/procedures for development and engineering. The SHARP Development Team was tasked with the design and development of the operations management tool, but was given no direction or requirements to build the system, with the exception of supporting the T & R. Many development teams might refer to this scenario as a worst-case, but the SHARP Team actually determined it to be the most Optimal scenario possible.

Given the requirement to build a system for immediate deployment to the fleet, the SHARP Team found it essential to institute a development plan that would prove flexible and efficient. Chapter Two briefly outlined predominant development models and DoD Software Engineering Requirements. This Chapter will detail the final process and methodology used by the SHARP Team to determine the ultimate models and standards for development.

1. Search for Existing Models

Software Engineering Development Models are also referred to as Software Life-Cycle Development Models, defined as:

> "The phases a software product goes through between when it is conceived and when it is no longer available for use. The software life cycle typically includes the following: requirement analysis, design,

construction, testing (validation), installation, operation, maintenance, and retirement."[57]

For the purpose of this thesis, predominant development models were discussed in Chapter Two and related appendices.

a. NPS Education

In the fall of 1997, I was enrolled as the first uniform service member to take part in a Distance Learning - Software Engineering Masters Course, sponsored by the Naval Post Graduate School. The course curriculum included studies in Software Methodology, Economics, Management, Testing, Design, Reuse, and Thesis Research. Studies and presentations in the Software Engineering and Management course provided a framework for determining a viable software development model. The course objective was to:

> "Educate the student in areas of great concern to the Department of Defense in the fields of software engineering and management. The course examines both the technical tools of software production as well as the software engineering techniques for software project management. Software testing, metrics and reliability are also covered. DoD software standards and metrics programs are included."[58]

The course included an in-depth review of stagewise development with the Waterfall Model, Prototype Model, Boehm Spiral Model, and Legacy and Reuse Models. This foundation granted me the ability to apply course practical knowledge and logic to determine the best practice for system development. The unique environment of distance learning allowed me to work during the day and attend course work in the late afternoons.

---

57 Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

58 Naval Postgraduate School Catalog, Naval Postgraduate, Monterey.

Techniques taught in the class could then be applied to the development process the next morning.

For the purpose of the SHARP Development, I was able to physically lay out all of the class presentation slides, handouts, and literature and do a systematic review of potential solutions, as noted in APPENDIX C – SOFTWARE ENGINEERING TECHNIQUES. Through the review and contrasted with the project development requirements, it was evident that none of the course prescribed solutions would satisfy the needs of the system, with the exception of the discussion about rapid application development. While RAD was only a topic of discussion, it was not reviewed as a formal model. In an attempt to create a complete development model for the project, it was necessary to combine knowledge from all of the different models. The resulting product was alluded to in Chapter Two as the Wedding Model.

Without the formal education from the NPS Program, it would have been more difficult to create a working development model for the SHARP System. Not all tasks and IT solutions nicely fit into the rigid structure of existing models. A background of solid software management education assists developers to tailor their life-cycle models to optimally fit the future needs of project development.

b. Subject Matter Expertise

Prior to assignment to the SHARP Development Team, members were previously tasked with a number of IT and non-IT system development projects as part of their military duties. One member, CDR Mark Burgunder, had an intimate knowledge of system development through his past experience with SPAWAR and other solution provider units. Another member, LT Joe Dundas, gained a business knowledge of system

development through his postgraduate education and leadership experience. I had also gained experience through my postgraduate education and previous system development tasking. As a part of military training, Naval Officers are taught the basic processes of leadership, management, and project development so that they may quickly adapt to their duties in military leadership. Many Naval Officers never get the chance to develop IT systems, but the concept of developments is nearly the same, from requirements, to risk, to design, and employment.

The pool of uniform service members with system development subject matter expertise is great. That pool is also populated with members who have specific subject matter expertise to particular tasks and fields of operation. All of the members of the SHARP Development Team had experience with system management, all of the members had experience with Naval Aviation, some of the members had experience with Naval Aviation Operations, and a few members had experience with actual software engineering. The combination of these experts solidified the base necessary to build a functioning software life-cycle model to support the rapid application development required for the SHARP Project.

c. Literature

In an attempt to gather the greatest collection of Software Engineering Models for comparison and contrast, the SHARP Team turned to three sources of information:

- Distributed Class Notes, Lecture Material, and Handouts
- Published Literature and Text
- Web Based Information

Specific lists of literature, notes, and web sites can be found in the reference section of this thesis. Before any analysis could be done from the given literature, it was important to understand the basis, bias, and theme of the document. Class material was directed towards educating students on the broad overview of the topic, with some comparison and contrasting arguments. Published Literature (Professional Text and Journals, Student Thesis and Dissertation Material, and Non-Profit Whitepapers) was designed disseminate information from the bias of the author and theme of the media. Web Bases Information ranged from biases commercial advertisements, to professional postings, to archive collections. Any information from the Web required independent verification and validation to ensure fact and content.

2. Search for DoD Software Engineering Requirements

For the purpose of the SHARP Development Task, Team Members contacted members of other development teams to query about existing Software Engineering Requirement, as set forth by the Department of Defense and Department of the Navy. Members of the team were directed to the new web based instruction sites for information on DII COE Manuals, DITSCAP Manuals, and other pertinent instructions. Copies of these instructions and manuals can be reviewed in APPENDIX D - DOD SOFTWARE ENGINEERING REQUIREMENTS. As discussed in Chapter Two, it was found that none of the DoD Software Engineering Requirement Documents directly controlled or governed the development of the SHARP Project. It was also evident that the employment of any of these requirement standards would overwhelm the development process and dramatically affect the timeline of distribution of the SHARP Project.

The SHARP Development Team faced a great amount of scrutiny when it was revealed that the Team was not following any of the established DoD Requirement Standards. Many of the organizations that posted scrutiny were the same organizations that established and policed the requirements. After a discussion with other development teams, solution providers, and commercial contractors, it was revealed that there was very sporadic compliance with DoD Requirements in the field of small IT systems.

After a review of all of the facts, requirements, initial development risks, and scope of the project, the SHARP Development Team elected not to follow any DoD Development Requirement Standard.

B. REVIEW OF PROCEDURES FOR DEVELOPMENT OF THE SHARP MODEL

1. Requirements Search

For the purpose of this thesis, the discussion of the requirement search shall be limited primarily to SHARP Version 3.1, with reference to previous versions. An optimal requirement search would be inclusive of all factors that could impact the operation of the system, regardless of how inconsequential it may appear on the surface. Additionally, a requirement search would include interviews and investigations with all applicable clients, subject matter experts, requirements, and regulations governing the system.

a. T & R Matrix Requirements

Initially, the requirement search for SHARP was based primarily on supporting the T & R Manual. The primary requirement authors were then CDR Mark Burgunder, CNAP N845; LCDR Peter Hunt, CNAP N836; and myself, LT Chris

Williamson, CNAP N845SH. A review and requirement tree was made of the T & R to find common points in the various type wing matrixes as well as to record divergences and differences. In an attempt to maintain some amount of consistency across the various wings, each TMS was required to develop a matrix that, on paper, fit a set format established by CNAP and CNAL. While the matrixes looked similar on paper, they all required distinctly unique implementation plans. Matrix points included, but were not limited to:

- T & R Qualification Name
- T & R Qualification Title
- The Media in which the qualification can be accomplished in – Aircraft, Simulator, or both
- Crewmembers who receive the qualification
- Valid currency periods of the qualification by ACTC Level, by crewstation
- The number of hours required to receive the qualification, and the annual flight-hours requirement for the qualification
- The point value of the qualification by PMA
- Notes
- Ordinance required for the qualification
- Resources required for the qualification
- Detailed qualification description and UNTL reference.

While all of the type wings were required to follow the basic format for T & R Matrixes, a number of type wings diverged from the norm and established unique formats and flows to model their own exclusive concept of operations. The F/A-18 Hornet Wing required an additional field to track repetitive qualifications, in that a specific qualification could be received multiple times on different flights for currency. The basic model implied that each qualification could be received an infinite number of times, but the sum of readiness points would only be based on the most recent instance of that qualification. The F/A-18 Wing Model permitted a qualification to be received a set number of times, and the sum of readiness points would be based on the addition of all

un-expired instances of that qualification, up to the set number of qualifications. This model permitted the F/A-18 Wing to track repetitive qualifications and to some extent imply proficiency by the fact that a crewmember could track the number of current instances of the qualification. In an attempt to best fit the requirement to all type wings, TMS were given the default of all qualifications being logged once, with the customization function of changing the qualification into a repetitive qualification with a number of instances. Another change was made to best model the fact that some instances of the qualification could have different point values based on the factors that existed at the time the qualification was logged, referred to as Degraders.

Due to degraders and repetitive qualifications, the sum of readiness points was no longer the total of the most recent instance of current qualifications. To best model the readiness of the unit, SHARP Requirements mandated that the readiness value be based on the total points of unexpired qualifications. In the case where there were multiple unexpired instances of a specific qualification, the instance of the qualification that had the greatest PMA point values would be selected for the calculation. In the case where repetitive instances of a specific qualification were permitted, the unexpired instances of that qualifications with the greatest PMA point values, up to the number of valid repetitive qualification as established by the matrix, would be selected for the calculation. It quickly became evident through the requirement search that simple idiosyncrasy in one matrix could quickly escalate into a wave of requirement changes affecting all of the wings as a whole.

Another critical change to the basic structure was the P-3 Orion Wing's requirement to track Tactical Crews. Type wings would determine crew compositions

based on a best-fit model of crewmembers by Crewstations, ACTC Levels, and seat numbers. Once the crewmembers were optimally determined and the crews assigned, the crewmember of each crew with the lowest PMA point value would contribute the overall readiness value of the unit. The P-3 Model required that crewmembers be permanently assigned by the operator to a specific Tactical Crew. Qualifications could only be received if all of the required crewmembers for a specific qualification were on the flight and met the required actions for the qualification, as a Tactical Crew. Qualifications are not received by an individual, but as a Tactical Crew entity. Some qualifications permit for exceptions that permit one or more members of the required crewmembers to be absent and still receive the qualification. The unit readiness is then determined by the sum of valid qualification's PMA points amongst Tactical Crews. This model required such a shift in the SHARP Requirements to warrant a parallel effort to tract crewmembers and compute readiness, based on the Best-Fit model and Tactical Crew model.

A number of other requirements came out of the initial investigation of the T & R Matrixes and the decomposition of determining unit readiness. It should be noted that the SHARP Product did not drive the format of the T & R Matrix, but that the T & R Matrix drove the SHARP Product and its supporting requirements. A discussion of the NALCOMIS System in the next chapter will note the driving of requirements by the technology of the system.

b. T & R Message Requirements

Once the team had completed a decomposition of the T & R Matrix, an additional evaluation of the actual T & R Message was completed. The first part of the T & R Message was the displaying of the command's readiness value. This determination

was accomplished from the calculations of the T & R Matrix, based on the calculation model (Best-Fit or Tactical Crew). The second part of the T & R Message included all of the flight-hour data for the unit. This display would be a summing and breaking down of fight hours and actions for the particular unit over a set period of time. Data included:

- The number and hours of Total Training Events
- The number and hours of Actual Training Events minus Transit Time, as a subset of Total Training Events
- The number of instances and hours of Transit Time, as a subset of Total Training Events
- The number and hours of Total Operational Events
- The number and hours of Operational Events where no Concurrent Training took place, as a subset of Total Operational Events
- The number and hours of Operational Events where Concurrent Training took place, as a subset of Total Operational Events
- The number and hours of Total Overhead Events
- The number and hours of Overhead - PMCF Events
- The number and hours of Overhead - Deployment/Detachment Transit Events
- The number and hours of Overhead - Aircraft System Abort Events
- The number and hours of Overhead - Weapons System Aborts Events
- The number and hours of Overhead – Range Aborts Events
- The number and hours of Overhead – Weather Aborts Events
- The number and hours of Overhead – Performance Incomplete Events
- The number and hours of Total Sorties
- The number and hours of Day Sorties
- The number and hours of Night Sorties
- The number and hours of Contingency Special Interest Hours accomplished on a sortie
- The number and hours of Deployed Special Interest Hours, subdivided into day and night, accomplished on a sortie
- The number and hours of Embarked Special Interest Hours, subdivided into day and night, accomplished on a sortie
- The number and hours of Exercise (International) Special Interest Hours accomplished on a sortie
- The number and hours of Exercise (Joint) Special Interest Hours accomplished on a sortie
- The number and hours of NVG Special Interest Hours accomplished on a sortie

- The number and hours of OPFOR Special Interest Hours accomplished on a sortie
- The number CV Arrested Landings subdivided by day and night, for the particular period.
- The number HELO Shipboard Landings, subdivided by Clear Deck, Free Deck, and RAST Landings, further divided by day and night, for the particular period.
- The Average Hours per Pilot, NFO, and Aircrewman, and the percent of T & R Required Pilot Hours accomplished by the Unit for the particular period.

All squadrons were found to fit into the T & R Message Flight-Hours calculations without any modifications to the requirements by TMS. While one TMS might not utilize some of the types of hours, another TMS might not utilize a completely different set of hours, but the message was kept constant to ensure standardization across all TMS. The requirement was that the system was written to encompass the widest possible view of the all type model series. Some requirements may be included at the request of only one particular TMS to support their unique needs, but when later incorporated in to the system and noticed by the fleet, many other TMS would find a need to incorporate it into their concept of operations. One example of this was the concept of Degraders.

Degraders were formally instituted by the F/A-18 Wing to track instances of qualifications received where some, but not all, of the requirements for the particular qualification were accomplished on the sortie. Under the F/A-18 Type Wing Manual, a set of independent Degraders and degrading values were listed for most qualifications. Degraders included a twenty-five percent reduction of qualification PMA points if the aircraft was limited in altitude due to fuel, a fifty percent reduction of qualification points if weather limited the ability to accomplish the qualification, or a ten percent reduction of qualification points if a specific tactic was not able to be implemented. This

methodology allowed for units to still receive qualifications, be they at reduced values, when factors limited aircrewmen from completing all of the required actions. While this concept appeared foreign to most other TMS, it was slowly adopted to better model points received by some qualifications received under degraded situations. The P-3 Wing looked at adopting this model to resolve a crewmember configuration factor in their aircraft. The H-60 F/H Wing adopted this model to demonstrate the reduction of readiness due to a lack of support ordinance for qualifications. New ideas from one wing gradually found application with other wings.

The review of the T & R Message and other documents to determine requirements is referred to as Backwards Analysis. Backward Analysis is formally defined as "an analysis to determine properties of the inputs of a program (*or requirements*) from properties or contest of the outputs."[59] Backward Analysis is one of the most popular techniques used by developers to determine how to build a system based on the required outputs of the system. If one knows what is supposed to come out of a product, it is significantly easier to determine what has to go into the development. Once the team had a blueprint of the T & R Message, all that was required was a analysis of the output, reverse engineering of the logic and methodologies, decomposing the declarations to their lowest common state, and then documenting the results. The reverse of Backwards Analysis is referred to as Forward Analysis, or "an analysis that determines properties (*or requirements)* of the output of a program from properties of the inputs."[60]

---

[59] Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

[60] Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

The development of product using Forward Analysis is usually reserved for products of indeterminate outcome such as dynamic report systems, user customized systems, and logic driven systems. While it is difficult to derive requirements from Forward Analysis, Forward Analysis is beneficial as a design technique to develop dynamic systems. The SHARP System would take advantage of the Forward Analysis Technique in the design and implementation of the Setup and Configuration Module of the system that would permit users to customize the SHARP to reflect the needs of their particular TMS.

c. Support Requirements

Once the initial requirements were mapped for supporting the T & R Message, a review of the requirements revealed various supporting requirements that were needed to establish and maintain the function of the system at large. While the T & R Matrix noted that qualifications should be received across various crewstations, there was no direct requirement to have a support table listing all potential crewstations and their attributes. Many of the initial requirements derived directly from the T & R Manual were incomplete in that they only listed the surface requirements of the system and did not list the supporting prerequisite of the system. The development team was located at the AIRPAC Headquarters, providing them untethered access to representatives of all of the various type model representative's desks. As subject matter experts in Naval Aviation and with access to various local TMS representatives, we were able to go through the requirements matrix and fill in most of the values needed to support the higher level systems.

While the requirements search was underway and being finalized, another effort was made to develop the initial prototype to model the requirements and

demonstrate functionality. This method was inline with the Wedding Cake Development Method discussed earlier in this thesis in that it permitted the development of some modules and sub-modules of the system while requirements were still being gathered for other modules and sub-modules of the same systems. This prototype was then used to verify the requirements of the system, as well as to demonstrate the potential system to the clients or rather, to other Naval Aviation Units at the first ATRRIS Meeting.

d. ATTRIS Meetings

During the development of the SHARP System, four ATRRIS Meeting were held to demonstrate any new prototypes under development, to review system requirements, and to discuss the current and future direction of the project. Three of these meetings were held in San Diego, California, at NAS North Island, while one ATRRIS meeting was held in Virginia Beach, Virginia, at NAS Oceana. Attendees of the meeting included members of all of the various type model series from both coasts, members of the CNAP and CNAL Training and Operations Staff, and members of OPNAV and NAVAIR Headquarters Staff. A significant portion of the meeting was dedicated to chalk talk of the SHARP System with members brainstorming about requirements and about design criteria. Many members came prepared with reports, printouts, drawings, and animations of the requirements that they required to be developed into the system.

Many of the ATRRIS Members were highly critical of the SHARP System in its present format due to the fact that it did not meet all of their personal operations requirements. The SHARP Development Team informed members of the ATRRIS IPT that they were supposed to be highly critical of the system and dissect it to unearth any

potential weakness and shortcoming. Members would be quick to point out oversights, misdirection in development, and shortcomings to requirements. While members of the IPT might criticize the present system, they would be quick to recommend solutions and improvements to enhance the design of the future product. Their critical view was imperative to ensure compliance with the needs of the fleet.

e. SMART-R Requirements

These various inputs, coupled with the T & R Instruction Requirements, were combined to generate a consolidated SRS for the development of the SHARP System. Even when it appeared that a significant amount of the requirements had been solidified, a new team was invited to join the development process and requirements search for SHARP – the SMART-R Evaluation Team. The SMART-R Team added a new view of the requirements to include a look from the Navy / Marine Corps Perspective, vice the limited review from the limited Naval Aviation Perspective. The SMART-R also introduced requirements that extended beyond that of just Flight Operations, to include information and requirements from the support side of logistics operations. After a complete investigation and review of the SMART-R's requirements, it became evident that, to build a viable Operations Support Tool, all of the primary and secondary requirements needed to be considered.

The ATRRIS IPT did a superior job at compiling a set of primary requirements needed to support Naval Aviation Operations. The SMART-R complemented the requirement search by closing the loop on ATRRIS IPT findings and adding secondary or support requirements. The SMART-R Team was able to poll inputs from a more diverse body of clients and users such as members of other armed services,

foreign services, private contractors, and higher echelons of Naval Operations. Due to a lack of budget and logistical limitations, the SHARP Team had to reserve its search almost exclusively to Naval Aviators, Squadron Personnel, and the support instructions that guided our operations. The SMART-R Team, with the geographic advantage of being located near Washington D. C., the physical access to members of higher echelons of the military service, the task of evaluating competing systems, and the financial funding to afford additional support personnel gave the SMART-R a distinct advantage to find requirements above and beyond that of the SHARP and ATRRIS search. It should be noted that the SMART-R Team was not able to document all of the primary requirements due to their detachment from the actual operators.

Due to the rapid development timeline requirement of the SHARP System, initial development had already started before all of the requirements had been finalized. In a cyclic approach to design, a small group of the AIRPAC Team members meet to discuss the risks of the different modules of development, options to reduce risk factors, and potential courses of actions taken to mitigate those risks. As requirements were finalized and risk mitigation plans were developed, the SHARP Team set out to design prototype models to best demonstrate the requirements. The prototypes were disseminated to various users, squadrons, and detachments for their evaluation of the developer's interpretation of the requirements and for the incremental integration of the system into the command. The resulting prototypes generated and solidified the requirements of the SHARP System, ensuring that each of the independent needs of the various units were meet by one integrated product.

2. Product Environment

a. Visual Basic

The backbone of any successful project is the programming or development language that is used to build the system. Due to the fluid environment and timely requirements that SHARP was to be developed under, it was necessary to find a development language that supported Rapid Applications Development. A chosen RAD tool had to be easy to use; had to accept reusable code; be a recognized industry standard; have a valid compiler and efficient debugging tools; had to be capable of creating usable prototype designs; and had to be cross compatible with the windows operating system. While the commercial sector markets dozens of potential software development packages, the SHARP Development Team was limited to four particular development suites, due to DoD license limitations, approved software development products, and product accessibility. The SHARP Team was limited to selection from the Microsoft Visual Basic 5.0/6.0 Development Suites, the Sybase PowerBuilder 6.0 Package, Inprise (formerly Borland International) Delphi 4.0, and the standard C++ Language Kit. The strongest development language of the four was undoubtedly C++. Due to the complexities of the C++ Language Suite and the management effort required to maintain a development, it was quickly ruled out as a valid RAD Tool.

In an effort to determine the best development tool for designing the SHARP System, the development team turned to a number of Benchmark Summaries and Evaluations to obtain an independent judgment of potential development solutions. The most inclusive Benchmark found was the NSTL Benchmark of Visual Basic,

PowerBuilder, and Delphi.[61] NSTL based its evaluation on the compatibility, performance, usability, acceptance (bug) testing, and BIOS evaluation of the three different development tools. A copy of the Benchmark Study can be found in the appendices of this thesis.

The summary of the NSTL Benchmark stated that "Visual Basic provides the most powerful enterprise development package. Visual Basic provides excellent performance, is the easiest to learn and use, and provides an unmatched feature set. The new OLEDB drivers give it the fastest data access among the programs tested. The new database tools give it the strongest database programming tools of any development tool on the market. Visual Basic is the hands-down favorite for COM-related programming tasks. Creating ActiveX controls and Microsoft Transaction Server components is simplified by exceptional debugger support and the native COM object model. The new ADO support and Microsoft Transaction Server integration make it easy to use Visual Basic for three-tier applications. For web development Visual Basic's integration with ASP and IIS will appeal to the many developers already using that platform. For the 95% of Visual Basic developers that are programming database-centric applications, an upgrade to 6.0 is a no-brainer. The new version makes it easier to develop your program, and it makes the end product faster."

"Delphi offers similar performance to Visual Basic, but lacks Visual Basic's compelling user interface and drag-drop database programming tools. We found the Borland Desktop Engine to be slower, less reliable, and less manageable than the OLEDB drivers used by Visual Basic. ActiveX and COM components in Delphi offer

[61] Benchmark Study of Visual Basic 6.0, Delphi 4.0, and PowerBuilder 6.0, NSTL, 03 Mar 1999.

comparable performance to Visual Basic, but are more difficult to program. Delphi has some feature advantages for advanced programming, but most programmers would prefer to do this type of programming in C++. For Delphi 3 programmers, an upgrade would be mandated if you have urgent CORBA requirements. The other features like the code browser and minor Object Pascal enhancements are not that compelling."

"PowerBuilder is a bad choice for developing anything for a Windows platform. It is tolerable for client/server development, but it is intolerable for web or three-tier development. The performance problems, bugs, and poor debugger (still bad, though much improved over previous versions) make it ill suited for this work. If you must work in PowerBuilder, the debugger alone is worth the upgrade."

Visual Basic was sited for its ease of use, its drop and drag development environment, ability to access and manipulate data centric objects, compatibility with the Windows OS, and overall reliability. The Visual Basic debugger provided error detection from the application level down to the transaction server level. Delphi was plagued with numerous bugs and poor documentation, making development of a new system nearly impossible without a strong background knowledge of Delphi's fundamental logic principles. Without an intuitive development process, Delphi was found unacceptable for the SHARP Development Team's needs. PowerBuilder was found less compatible with Windows OS than we had hoped, creating a non-standardized GUI and excessive DLL conflicts. The PowerBuilder Debugging Tool could not be used in the development environment, but rather off line, restricting real time debugging and solutions correction. NSTL actually recommended that user now upgrade to the new version of PowerBuilder

6.0 for some development projects due to the number of faults and errors in the new system.

After a full review of the NSTL findings, personal preferences of development, the availability of skilled programmers and engineers in specific languages, and the ultimate risk of using the different languages, it was determined that the Visual Basic Development Suite would best meet the needs of the SHARP Project.

b. Hardware Requirement

Shortly after developing the prototype models, it quickly became evident that the existing fleet deployed computer systems would not support the SHARP Product. The effort to install IT-21 based system was planned and funded, but had not yet reached the fleet in 1998, when the SHARP Project was initiated. IT-21 PC Standards were defined as:[62]

- 200 MHz Pentium Pro CPU
- 64 MB EDO RAM
- 3.0 GB Hard Drive
- 3.5" Floppy Disk Drive
- 8X IDE CD-ROM
- Dual PCMCIA/PC Card Reader
- PCI Video with 2MB RAM
- 17 Inch Monitor (1280-1024)
- Pointing Device (Trackball or Mouse) and Keyboard
- Soundblaster (compatible) Audio Card with Speakers
- CPU compatible 100 MBS Fast Ethernet NIC

The minimum requirements for IT-21 computers would meet the needs for installation and execution of the deployed SHARP System. Due to the rapidly approaching deployment timeline, it was elected to take money from the Flying Hours

---

62 Archie Clemins, Admiral, *"IT-21: The Path to Information Superiority"*, Quadrennial Defense Review Secretary's Report to Congress, Jul 1997.

Program and purchase four computers for each squadron and detachment. Since the SHARP System was to be developed by members of the operational staffs of CNAP and CNAL, the development teams had ready access to members who could channel the financial burdens of hardware onto other operational budgets. These computers would serve as stopgap systems for the Operations Office until the deployment of actual IT-21 computers two years later.

c. MDB vs. SQL

VB has the ability to support a wide variety of database formats, from the Microsoft Database Format (MDB) to the Structured Query Language (SQL).[63] Due to site licensing cost and footprint distribution issues with the SQL Database, the SHARP Development Team was required to look for alternatives that could still provide the robust features of a dynamic database, be widely compatible with other related systems, and use conventional data queries to access data.

The MDB format, using MS Jet Engine Version 3.51.2723.0 was authorized for open application distribution as part of the SHARP Development Package. MDB, via the VB Development Environment, used standard SQL script to access data from the database. Relationships, Indexes, and Keys were customizable through database design and could be dynamically reformatted through executable code. Based on its small footprint, compatibility, and license issues, the Microsoft Database Format was selected to support the initial data requirements of the SHARP System.

---

63 Microsoft Data Engine for Visual Studio 6.0, Public Affairs Release, Microsoft Corporation, 23 Nov 1999.

Microsoft has elected to include the SQL 7.0 MSDE (Microsoft Data Engine) as part of the new Office 2000 release.[64] As well as the MSDE release, SQL 7.0 has also authorized the release of the run time database component with complied products. Efforts are now underway to incorporate SQL 7.0 into future releases of SHARP to improve efficiency and optimize the product to a more robust supporting database. SQL 7.0 will add additional relationship integrity to the system, as well as data triggers that permit logic statements to be executed in parallel with data manipulation from the database level.[65]

d. Dissemination to the Fleet

Today, there are over 4,108 Operational Naval Aircraft, deployed to over one aviation hundred squadrons and support units, on twelve Aircraft Carriers and over two hundred aviation capable vessels and shore stations deployed worldwide. The logistical dilemma of shipping a copy of the software to all aviation units using the postal system was found impracticable due to the postage cost and delay in transit. The Fleet Post Office (FPO) System notes that shipping time for packages to overseas and deployed units could be as short as seven days or as long as fifty days when shipped via standard mail.[66]

In an effort to get around the logistical problems of mailing the product, the SHARP Team devised a three-prong approach to the dissemination requirement. The

---

64 Microsoft Data Engine for Visual Studio 6.0, Public Affairs Release, Microsoft Corporation, 23 Nov 1999.

65 Evaluation Guide, Enterprise Application Development Support, Public Affairs Release, Microsoft Corporation, 10 Sep 1999.

66 Postal Bulleting 22011, United States Postal Service, 18 Nov 1999.

First solution was to post the entire system on a World Wide Web (WWW) Site hosted by AIRPAC for users to download the program and then install it on their system. Most ships and stations had access to the Web through either Base or Ship provided Local Area Networks (LAN) or via telephone line. Once on the web, users could download the system at their convenience and install it where needed. The Web also provided users with a ready location to download the product in the event that their existing system had crashed, had to be re-installed, or the original copy was lost and needed to be refreshed.

The Second method was to post the entire system on the SIPRNET. Some units did not have reliable connections to the Web due to logistical problems at their assigned duty station or station of assignment. The SIPRNET provides a more secure network for uses by defense personnel to access sensitive information through authorized locations. SIPRNET dissemination provides the same benefits as Web dissemination, with the additional feature of a secure link and alternative connection methods.

The third and most basic technique was to abandon "Snail Mail" methods for E-Mail. The SHARP Team has consolidated a list of over 330 persons, units, or locations that requested to be added to a centralized list of recipients for update bulletins, announcements, and releases to the program. When requested, the SHARP Office could quickly mail a packet of SHARP Products to the required recipient to immediately meet their specific needs. Some units were unable to reach the SHARP Pages due to network problems, firewalls, and connection rates. The E-Mail alternative allowed for an immediate response to time critical requests from the fleet. Most E-mail packets were limited to 3.0 Mb in size to ensure that they would fit through the e-mail firewalls of fleet

servers. Some servers required files packets to be less then 1.0 Mb in size, resulting in a quick repackaging of the system, and update.

One user referred to this technique of dissemination as "Navy Shareware," akin to the shareware software that users can download free from the web and then decide if they like it. This technique has a number of shortcomings including limited product tracking, reliance on web conductivity at the user level, the unintentional release of product to unknown individuals, unauthorized access to the product, and no hard media of the product. These shortcomings were easily outweighed by the rapid rate of dissemination, easy access to the product, and reduction in overhead cost to manage the distribution.

3. Leapfrogging Technique and the Three-Year Rule

One of the most popular and often quoted software / hardware development principle is Moore's Law from 1965 that states;

> "The pace of microchip technology change is such that the amount of data storage that a microchip can hold doubles every year"[67]

This law was later revised in 1975 to double every eighteen months. This principle is based on the concept that technology is constantly improving thereby adding to an exponential improvement of hardware to support the improvement. This cyclic reliance on technology to hardware to technology results in a spiraling improvement of the rate in which we can process data, build new systems, and update legacy products.

---

[67] Rishi Khanna, Jennifer Kwan, Jason Lamin, Ingred White, *"More on Moore's Law"*, University of Texas, 1997.

Through my research, I have realized that another rule applies to software system longevity, that - No IT systems should remain in its present state for longer then three years. This short life span is limited by the fact that:

- The System is replaced by an update to itself, manufactured by its parent developer.
- The System is replaces by another system, manufactured by an independent developer.
- The System is no longer required and is ruled obsolete.

Industry and Defense have exemplified the Three-Year Rule in the recent decade by the following examples:

Windows 1.0	November 20, 1985
Windows 2.0	April 2, 1987
Windows 3.0	May 22, 1990
Windows 3.1	April 6, 1992
Windows 95	August 24, 1995
Windows 98	June 25, 1998
Windows NT 3.1	August, 1993
Windows NT 3.5	September 6, 1994
Windows NT 3.51	June, 1995
Windows NT 4.0	August 24, 1996
Windows NT 5.0 Beta	August 1998[68]
SHARP 1.0	May 1995
SHARP 2.0	September 1996
SHARP 3.0	August 1998
SHARP 3.1	March 1, 2000
SHARP 4.0	December 29, 2000

Many systems developed by the commercial sector or by defense contracts have unintentionally followed the Three-year Rule, under the threat of being beaten out by competition or due the fact that requirements have changed so dramatically to warrant a

new system. The time to develop many systems takes longer then the three-year window applied to this assumption. In an attempt to remain within the three-year window and not be outdone by competition many developers use a leapfrogging development technique where one system is developed, but before its it released, another team is stood up to start the development of the next system, and so on. To ensure continuity and add subject matter expertise between the two systems, some members of the first team may be pulled off the development and assigned to the second team, as depicted by Figure III.1.

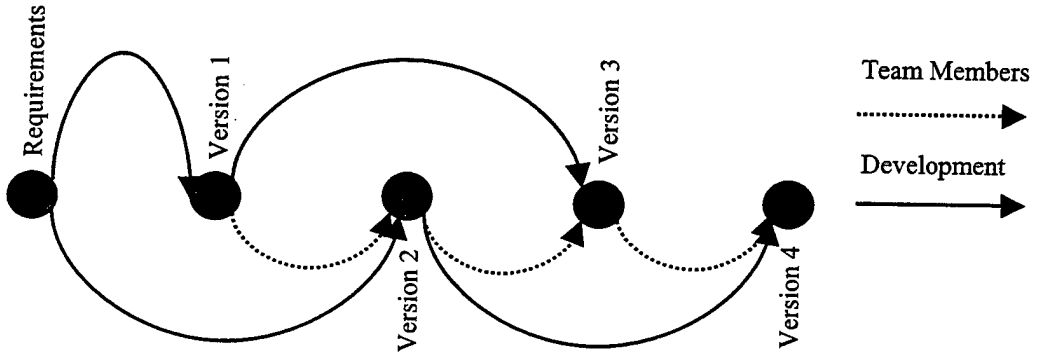


Figure III.1 Leapfrogging Development Technique

If properly executed, the Leapfrogging Development Technique would permit a developer to start the development of the first version of the system, then soon after, start the development of the second system. Through this technique, the first version serves as a prototype to the second system. Requirements and lessons learned by the first version can be applied to the second and subsequent iterations of development. To assist in the understanding of the development requirements and to assist in minimizing risks, members of the first team can also be applied to the development of the second system,

---

[68] Frank Condron, "*Frank Condron's World O'Windows - Microsoft Windows History*", http://www.worldowindows.com/wintime.html, February 27, 2000.

bringing along proven techniques, methods, and reusable systems. Through leapfrogging, this technique can be applied through the entire life of the system.

In the development of the SHARP System, the chain of events dictated that a product be produced that was flexible enough to meet the fluid documents requirements established by the chain-of-command. To best manage this requirement a number of independent efforts were tasked with the design of the system on various layers of the "Wedding Cake."

The first iteration of SHARP was started in 1995, using an Excel Spreadsheet with embedded script and support code. This Spreadsheet was designed at HS-6 to track qualifications on a "Ribbon Sheet". A Ribbon Sheet is a long continuous sheet of information containing crewmembers up the vertical axis, qualifications along the horizontal axis, and qualification expiration dates as the resulting value at the intersection of the horizontal and vertical axis. Various other support spreadsheets were written to archive calculations, code, and views to compile the Ribbon Sheet. Two aircrewmen and myself were tasked with maintaining and utilizing the initial version of SHARP 1.0. It was only used within my home squadron – HELANTISUBRON-SIX.

The second iteration of SHARP, also developed exclusively for my home squadron, was started in late 1996. This version incorporated the Visual Basic 4.0 coding language in a 16 Bit Environment. Visual Basic 4.0 was chosen for its ease of use, compatibility with the Windows Operating System Environment, and site license to the Navy. SHARP 2.0 was built upon the requirements of version 1.0, with the addition of modules to capture additional operational requirements such as flight-hours tracking and mission or task accomplishments. SHARP 2.0 was build in parallel with the to the

operation of the version 1.0, with 1.0 users assisting in the requirements search and development of 2.0.

When it came time to start the development of 3.0 in March of 1998, version 2.0 was not complete and a number of requirements remained open. A stop-gap effort was made to lockdown the development of 2.0 while requirement search was done for version 3.0. Version 3.0 included bring SHARP into the 32 Bit Environment, the expansion of the system to include all 17 different type model series aircraft, and the expansion of functionality as noted in the SRS Document. Version 3.0 was disseminated via the Web, both through the SIPRNET and INTERNET, which permitted a rapid integration to the fleet. A decision was made to lock down the progressive servicing of Version 3.0 in the anticipation of my transfer from the project and to establish a baseline for the contractor to work from. The new Version, SHARP 3.1, was published via the Web and on CD to the fleet twenty-four months after the release of Version 3.0. While efforts are being made to maintain and develop Version 3.1, SHARP 4.0 has already been tasked for development by AIRPAC.

The SHARP Model required rapid changes and successive releases through its development in an attempt to keep up with the fluid requirements of Naval Aviation. It would have been extremely difficult to efficiently manage the engineering of the product without using a Leapfrogging Technique to model the efforts of two parallel project developments.

4. Development Technique

a. Applied Wedding Method

In compliance with the Wedding Cake Model discussed in the Chapter Two, there were a series of stages to be completed to develop the SHARP system, as depicted by Figure III.2.

	Stage	Task
1.	Tasking	Develop an IT Software Solution to support the CNAP / CNAL T & R Matrix.
2.	Requirements Search Stage	Gather initial requirements set for the development of the T & R support tool. The initial set of requirements was to include the basic framework or logic backbone for the system.
3.	Module Breakdown and Task Assignment Stage	Initial investigation revealed no need to break down requirements set. Team would continue to work as a single unit to develop the prototype task.
4.	Solution Search Stage	Determination of the required hardware, development language, and supporting database to meet system requirements.
5.	Risk Analysis Stage	Determination of potential risks: • with supporting software against system requirements • with supporting hardware against system requirements • with supporting database against system requirements • through cross comparison of requirements
6.	Prototype Stage	Development of initial prototype.
7.	Evaluation Stage	Evaluate prototype to determine if: • the hardware will meet system requirements • the software will meet system requirements • the database will meet system requirements • system requirements can be meet through development • the requirements meet the client's needs Evaluation team consists of developers and clients

	Stage	Task
8.	Second Cycle – Requirements Stage	Gather detailed requirements set for the development of the T & R support tool, based on prototype evaluation results. Develop test methods based on requirements.
9.	Module Breakdown Stage	Assign team Members to Build: • Main Menu Module • Configuration Module • Crewmember Support Module • T & R Support Module • Flight Logging Module • Report Module Team members diverge to own task, assigned requirements, develop test methods.
10.	Solution Search Stage	Main Module Team Establish data configurations and links between modules. Determine tools required to accomplish each task.
11.	Risk Analysis	Independent Teams evaluate risk of independent module development. Main Module Team Evaluate risk of system development up to this stage.
12.	Prototype Development	Independent Teams Develop own Prototype Module. Main Module Team Develop Main Module Where able, completed modules shall be joined to the Main Module.
13.	Evaluation Stage	When each Independent Module Prototype is complete, evaluate if: • the independent module will meet module requirements • the hardware will meet module requirements • the software will meet module requirements • the database will meet module requirements • the requirements can be meet through module development • the module requirements meet the client's needs Evaluation team consists of each independent module developers, the Main Module Development Team, and the client. Where applicable, the client shall be delivered the working evaluation product for employment.

	Stage	Task
14.	Additional Tasking	Develop a Software Solution to support all aspects of Naval Aviation Operations
15.	Third Cycle – Requirements Stage	Gather detailed requirements set for the development of the Aviation Operations Support Tool, based on new tasking, prototype evaluation results. Develop test methods based on new requirements.
16.	Repeat Cycles as required to meet requirements	

Figure III.2 Lifecycle Development of the SHARP System

The first step of development was to receive tasking for the actual project. The SHARP Team was tasked in the spring of 1998 to develop a system to support the CNAP / CNAL T & R Matrix. Tasking included definition of the initial project requirement, a line of authority for the development, and funding lines for appropriating services, supplies, and travel. The initial tasking was assigned to a group of uniform service members, selected for their proven history of project development, software engineering knowledge, and aviation subject matter expertise.

The second step was to initiate a requirement search of initial baseline requirements. These requirements were based on the T & R Instruction, supporting instructions and documents, and personal knowledge of the development team. It was not necessary to define all system requirements at this early stage, but to generate a general requirement set or theme of requirements to build the first prototype on. Team members were also required to compose test scenarios for the evaluation phase during the requirement search.

The third step was to evaluate the system for Module Breakdown and Task Assignment. In the early stages of development, it was not advantageous to break the system down into modules before the initial prototype was evaluated. For the first cycle of development, the engineering team remains as a unified group to properly examine the direction of the system. Team members were also required to compose test scenarios for the evaluation phase during the Module Breakdown. If the team members noted that requirements were not properly formatted to permit module breakdown or were not complete, the team should make recommendations to be reviewed on the next visit to the requirement search.

The forth step was to make a solution search for the tools and instrument required to support the development. It was imperative to make a determination early in the development process as to the design language required to design the SHARP Project. A conclusion was made to use Visual Basic as the design language based on an evaluation of potential design languages, ease of use, its ability to compile an executable rapid prototype, and the preponderance of qualified engineers. Further evaluations were required to determine the database format capable of supporting the SHARP System development. Due to economic limitations and distribution paths, the SHARP Team elected to incorporate the MDB Database Format for the initial stage of design. In some cases, it is necessary to choose a lesser tool due to the limitations of the overall support structure. Finally, an evaluation was required to determine the hardware necessary to support the deployed system. Due to the rapid time line of the project, it was necessary to select the hardware early to ensure sufficient time to have it ordered and delivered. Team members were also required to compose test scenarios for the evaluation phase

during the Module Breakdown. If, during the Solution Search, the team noted that requirements should be modified to match potential solutions, then they should be registered for review in the next visit to the Requirements Stage.

The fifth step of development was to evaluate the requirement and solutions for risk. For the first cycle of development, the risk analysis should evaluate the software, hardware, and database selection to ensure their ability to support the project requirements. Risk should be assigned to determine the level of difficulty in implementing each of the three system tools. An initial evaluation of project requirements should be made to determine the risk of developing the system as a whole. The results of the risk analysis should be used to revise the overall system requirements and to manage the actual system design. Team members were required to compose test scenarios for the evaluation phase based on risks determined during the Risk Stage. If risk was determined to be too high for the development phase, then the development manager could return the cycle back up to the Solution Search Stage to determine alternatives.

The sixth step is to develop the initial prototype of the system. In the first cycle, it is only necessary to develop a GUI phase prototype to demonstrate the potential look and feel of the system. The start of this process may be dedicated to white-board designing of the look and then transposed into a working model. The sole purpose of the prototype at this stage is to demonstrate the assignment and perception of system requirements. Based on development techniques, team members should compose test scenarios to evaluate the completed project. If, during the development stage, the team realized that requirements could not be satisfied, were not complete, or were noted by the

client to not meet their needs, changes should be registered to be reviewed in the Requirements Stage.

The seventh stage of development was to evaluate the completed prototype against project requirements. An evaluation should be made against the hardware, software, and database selected and of their performance against system requirements. A second evaluation should be made to determine if the prototype would meet the project requirements. A final evaluation would then be made to determine if the requirements would meet the client's needs. It is imperative that both the clients and developers be involved in the evaluation phase.

At the completion of the first cycle, a reinvestigation of system requirements was necessary to determine system function at a greater depth. With each re-visit to the requirements stage, requirements analysis should be made to a greater and greater depth or to a greater width as tasks increase.

The ninth step was to make a Module Breakdown and Task Assignment of the system. Independent teams were tasked with the development of each module. A single team was assigned to develop the Main Menu Module, which then served as the key or backbone to the entire system. The Main Menu Module contained the menu logic, data transfer protocols, and system standardization design for the entire system.

The tenth step was to determine tools required to meet the independent module's development requirements. These tools could have been existing modules further developed to meet new requirements, reusable code incorporated into the new system, or the incorporation of parallel efforts that also meets the independent module requirements.

The eleventh step in the process was to determine the risk of the development of each of the modules. Risk would either be an evaluation of the difficulty to accomplish the independent tasks, or an evaluation of the over tasking of individual teams. With the intent of keeping program development cycle time to a minimum, independent teams should not be tasked with too large a module. Historically, larger projects have demonstrated a greater change failure. In an attempt to avoid this trend, the Wedding Model recommends the breaking up of the project into small manageable modules. The risk stage should serve as a check and balance to the Module Breakdown Stage.

From the point of Task Assignment on, independent teams were free to proceed at their own pace to meet the specific requirements of their independent module. This freedom permitted each team to determine solutions, analyze risk, develop prototypes, and evaluate their design for requirement completeness. The development team of the Main Menu Module was assigned with consolidating each completed module prototype when they themselves phased through the development stage.

The Main Menu Module Team served as the final check and balance of the system. As each independent team would complete their cycle of development and evaluation, the Main Team would take the completed module and combine it to produce a single product. This product would then be evaluated with the client for completeness. If sufficient to meet the client's needs, the prototype was employed to client stations.

With each successive revolution on the development cycle, independent teams would re-evaluate requirements for completeness. If requirements were complete and clients needs were satisfied, the team would then be available for new module tasks.

In some cases, a development team may be broken up to conquer smaller module of their original module.

As development continues and each independent team completed their own module, the development process diverged into multiple layers. The Main Module continued to serve as a repository for reusable code and served as the backbone for each completed system to link to. In some cases, modules were determined to be too difficult to complete and failed in their development. As each module was engineered as an autonomous unit, their failure did not effect the function of other working systems.

Clients were permitted intimate access to the life cycle development of the system through the requirement to the evaluation stage. This intimate access afforded the client the ability to modify the product before the development team spent excessive efforts to engineering the system. As the client determined the existing system to satisfy working requirements, then the client could deploy the completed product to the fleet.

b. Motivations and Theory

Einstein stated that, "Everything should be made as simple as possible, but not simpler."[69] The simplest solution to the SHARP Development Problem was to break up the complex project of aviation readiness automation into a number of smaller simple projects, without removing any of the greater functionality. These smaller projects could then be engineered as independent units, tested, and then consolidated for a final system. As long as each single unit functioned properly, satisfied the testing criteria, followed the main linking protocol of the system, and the main system functioned properly, then it could be closely assumed that the entire product would function.

---

[69] Albert Einstein, "Autobiographical Notes", Open Court Publishing Company, 1991

Basic principles of Software Management imply that the longer a project takes to develop, the larger the project is, or the more complex the project appears, the greater the chance is that the project will fail.[70] This principle can be graphed as noted in Figure III.3. Studies by the Standish Group found that a great percentage of large projects failed due to their size, their complexity, and by the fact that many developers were moved off the project before system design completion.[71] These project failures resulted in significant cost overruns and delays in system completion. In an attempt to prevent repeating the follies of the past, the SHARP Development Team attempted to reduce the scope of the project by developing small units of the project in independent modules. The small size permitted rapid module development, quick verification of requirement completeness, and safeguarded the project against the turnover of developers. As an individual would be moved off of the team, his impact would be isolated to a smaller portion of the overall system in comparison to traditional programming techniques. The military environment did not permit great tolerance to overruns and delays when it came to uniform service member development. The concept of "Development by Mandate" where members of the SHARP Project were ordered to produce a product encouraged the Team to take the most efficient steps possible to develop the best product in the shortest amount of time.

The SHARP Development Team used a Top-Down Design Methodology – defined as a design technique which aims to describe functionality at a very high level, then partition it rapidly into more detailed levels one level at a time until the detail is

---

[70] Man-Tak Shing, CS3460 Software Methodology Slides, Naval Postgraduate School, Monterey, CA, 1999.

sufficient to allow coding.[72] This theory implies taking the product at its basic level for an initial decomposition, and then continuing to slice the product up into greater and greater increasing levels of detail for further decomposition. The Wedding Model permits this kind of decomposition, with the project given a high level decomposition, and then each independent module is further decomposed to reveal the intimate workings. As each cyclic visit, the module is further decomposed until all of the requirements have been discovered and developed.

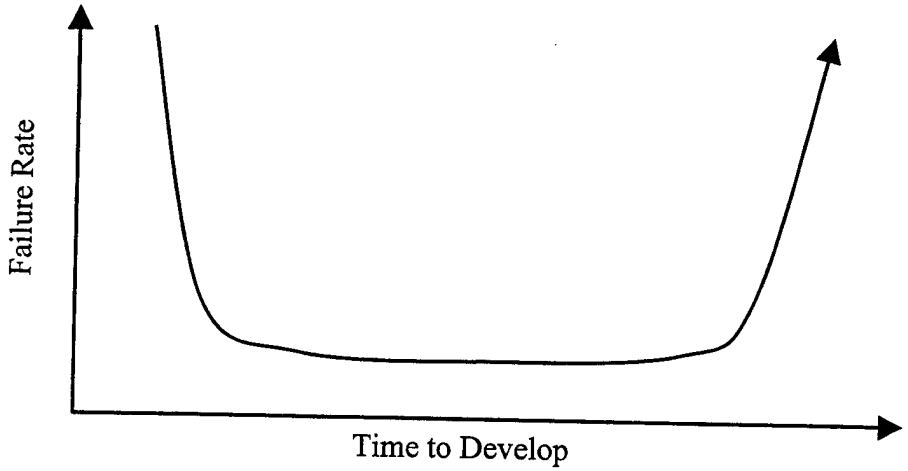


Figure III.3 Failure Rate vs. Time to Deploy

5. Commercial Sector Motivations

a. From the Trowel to the Electron

As discussed in Chapter Two of this thesis, the commercial sector is making a gradual shift in techniques and operations from the "Bricks and Mortar" Environment to the "Clicks and Mortar" Environment. This shift could not be accomplished without a significant realignment of resources and change in habits. The

---

71 *"CHAOS"*, The Standish Group, www.standishgroup.com, 1995.

72 Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

SHARP System was an initial step in the process of moving Naval Flight Operations from the days of grease boards towards to the world of electronic data processing and transfer.

This thesis uses the terms "Trowel" and "Electron" to define the mechanism that creates each of the unique environments. The trowel represents the conventional methods of building systems, products, and outputs by manual labor. The electron represents the use of electronic or smart systems to design systems, products, and outputs. While it is very difficult to develop a product that is exclusively "Electron" based, the use of a smart system can potentially increase the efficiency of the operations by removing the expense and potential for human error.

In the trowel-based world, businesses would manually gather data, raw supplies, and use traditional manufacturing techniques to produce a product. Naval Aviation would use manual techniques to prepare schedules, record post-flight events, and manage squadron operations to produce readiness calculations and other reports. In the electron-environment, businesses have the ability to reduce hard material to electronic storage, encourage automation devices, and increase efficiency by producing the same product using technological advances. Naval Aviation Operations benefit by electronically preparing their schedules, building automated business rules for operations, and preparing accurate reports based on a controlled user input environment.

It would be difficult to build a system that could remove all mortar-based environments from Naval Operations, but the SHARP System would dramatically reduce the overall workload of the aircrew by adding a streamlined and automated operations business process. Using this motivation, coupled with the subject matter expertise of the

development team, the SHARP SRS was written such that it outlined every possible way to reduce the overall workload of the command through instituting automation. This automation was not a change in the operating process, but rather an establishment of a virtual form of the existing environment. This virtual form reduced the overall training requirement of the SHARP Project because the project directly matched the existing concept of operations.

b. Software Shopping in the Bazaar

One of the very popular papers about open source development is Eric Raymond's essay entitled *"The Cathedral and the Bazaar."* While the development of Red Hat and other Linux open-source products may be controversial, the methodology is based on sound management, new generation development techniques, and logical business practices.

Eric Raymond thought that all large systems should be built like a cathedral, with no beta products to be released before the complete product was ready for distribution. After his review of Red Hat, he admitted that the bazaar style of many small product development efforts is more efficient and flexible. The SHARP System also used this bazaar type atmosphere to encourage multiple independent efforts in the development process. The SHARP Bazaar was set up with basic ground rules to ensure that each "stall" would "sell" compatible products with autonomous flair and timelines. A customer could purchase the products he desired and place them into his Main Module "Shopping Basket" to create a system customized to meet his type wing's unique needs.

The first principle of the *"Cathedral"* essay was that *"Every good work of software starts by scratching a developer's personal itch."* SHARP was developed from

my own personal itch to make my job easier when I served as a Pilot Training Officer. Necessity drove the development of the initial system, and proved the concept to build the current product.

"Good programmers know what to write. Great ones know how to rewrite (and reuse)." The SHARP System today is developed on the bones of yesterday's product. Through reuse and rewriting, using the concept of top-down development, the SHARP Team was able to efficiently tune the product to meet the fluid requirements of Naval Aviation.

"Plan to throw one away; you will, anyhow,"[73] and *"Often, the most striking and innovative solutions come from realizing that your concept if the problem was wrong."* The SHARP Development Team has thrown many iterations of the system away with each revolution of the development cycle. The beauty of any spiral development is the ability to refine the product through repetitive prototype design. It would be difficult to see any true similarities between the first iteration of the product to the current release due to the constant reevaluation of requirements. Cyclic development gave us the opportunity to take a step back after each development phase and redirect our efforts to better meet the client's needs. If it became evident that the present cycle was ill developed, then the SHARP Team could simply not implement the new design into the Main Menu Module and return to the requirement stage.

"If you have the right attitude, interesting problems will find you." The SHARP Team has constantly fought the interesting problems of software engineering,

---

[73] Frederick P. Brooks, Jr., *"The Mythical Man-Month: Essays on Software Engineering"*, Addison-Wesley Publishing Company, July 1995.

command management, competition, and the military development structure. It was only through engineering knowledge, strong will, and dedication of the team members that the project was able to strategically work past these interesting problems.

"When you lose interest in a program, your last duty is to hand it off to a competent successor." The military provides a great tool for refreshing uniform service member through active duty service rotations. These rotations permitted a fresh body of developers and evaluators at irregular intervals. As part of the rotation, members would be able to hand off their efforts to a new developer before they lost interest in the total endeavor.

"Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging." The SHARP Team treated all users as requirement analysis, developers, and evaluators to ensure that the subject matter experts were given the greatest access to the system. There comments, criticisms, and drive are what guaranteed the success of the project.

"Release early. Release often. And listen to your customers," and *"The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better."* SHARP users were given frequent access to the consolidated product so that they could take advantage of new tools, criticize misinterpretations of requirements, make recommendations for redirection, and test the system for bugs. Squadrons were informed that their ideas would be taken seriously, and many were able to see their recommendation implemented in very the next release. Squadron users had a unique perspective of the program as they were implementing it at the "Tip of the Spear."

"Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone," and *"If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource."* With over three hundred squadrons and detachments using the SHARP Product in one release version or another, it was inevitable that someone would experience a bug that was not noted by the development team. As bugs were found, an error window would prompt the user of the specific module that was in operation and line of code, which they could then forward to the SHARP Team for quick repair. There were "NO DUMB REQUIREMENTS." If a squadron user requested an improvement, it was treated as if was the most important improvement to the system.

"Smart data structures and dumb code works a lot better then the other way around." SHARP was based on supporting a database of information. The database was constantly refined and improved to ensure optimal performance. Many improvements to the database compensated for shortcomings in code.

"Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected." The initial version of SHARP was designed to support the Training and Readiness Matrix, but soon became a tool to support virtually all aspects of Naval Aviation Operations. Users realized that they could customize the product to meet their own unique needs and then could forward these changes back to the development team for full fleet implementation.

"To solve an interesting problem, start by finding a problem that is interesting to you." The converse of this method is find developers interested in the

problem. Every developer on the SHARP Project was intimately involved with Naval Aviation. The users were their friends. They had a relationship with members of the chain-of-command and understood the risk of failure to their own reputations and to Naval Aviation.

The same attributes that made Red Hat a development success are applicable throughout a myriad of software development projects. A close client-developer relationship, integrated management practices, and small independent design are imperative to the success of any project. The SHARP Team is just another example of a successful implementation of these principles

C. CHAPTER SUMMARY

Before the SHARP Team could develop any product, they were required to determine the optimal model of life-cycle development process, based on a collection of know and accepted models. Software Life-Cycle evaluations were accomplished using techniques learned from course studies at the Naval Postgraduate School, past development knowledge, and software engineering subject matter expertise. After a detailed study on the subject, it was determined that none of the existing models would satisfy the requirements of the development requirements due to the strict deadline and fluid requirement set of the system. The only potential solution to the development dilemma was to create a customized life-cycle model based on a combination of rapid application development techniques, spiral development models, and parallel development methods.

The SHARP Development Team was made up of a number of highly skilled uniform service personnel who were subject matter experts in Software Engineering,

Project Management, and Naval Aviation Operations. The talented members of the team put together to design and manage the SHARP System were not unique to the armed services. Officers and enlisted men serve in a variety of billets that afford them the opportunity to learn and hone their skills in project development and management. A number of officers have received special training in information technology and software engineering as a part of their duty assignments. This prior inner-service training afforded the SHARP Management Team a large pool of qualified individuals to select from. The Navy provided me with significant training through duties and assignments with operational squadrons and through curriculum with the Naval Postgraduate School.

Once the team had a foundation of members, a search was made of software engineering, project management, and system development literature to determine potential life cycle models that could support the demanding needs of the SHARP Development Project. After an exhaustive search and evaluation, it was determined that a highbred parallel spiral development model would be the most practical model that could support the design of the system. This conclusion was only reached through an evaluation based on the past development experience of team members, a review of formal models, and the application of logical analysis of development risks.

In parallel with the evaluation for potential applicable life-cycle development models, an evaluation was made for applicable DoD Development Requirement Standards. Due the scope of the existing requirements, the lack of requirement oversight and authority, non-accreditation of standards, and excessive overhead burden that the standard would force on the RAD of the system, it was determined to not follow any DoD Development Requirement Standard.

The development of the SHARP system required an in-depth search and discovery of requirement to cover the entire scope of the project. The greatest difficulty of the requirement search was to capture the unique requirements of each of the seventeen type-model aircraft for development, without limiting the functions of any other aircraft series. The ideal development would produce one single system flexible enough to encompass all potential aircraft types. A series of backward and forward analysis techniques were used to ensure a complete investigation of requirements. In parallel with the SHARP Requirement Search and Development, the SMART-R Assessment provided a corresponding requirement matrix for cross comparison. This cross comparison ensured a reflective requirement view from alternate members of the military aviation chain-of-command.

The SHARP team was restricted to a limited number of software development products due to merchandise available to DoD development groups. In an attempt to determine the best software development solution, the SHARP Team relied on a series of white papers, personal preferences, and development considerations to determine the optimal product. The SHARP team decided to develop using the Microsoft Visual Basic 6.0 Suite with the Microsoft Database. The development environment was left open ended to future development and expansion by new products in the event that the DoD software development atmosphere changed.

One of the prime goals of the SHARP Development Team was to find a way to distribute the software to the fleet in the most resourceful manner possible. The Internet provided the most cost effective and efficient vehicle for near instantaneous distribution of the software, as well as a convenient means of archiving database templates and

newsletters for user access. Electronic transmission media provided a reliable method for the transmission of data, program updates, trouble calls, evaluation and critiques, as well as beta-tester bug reports. The SHARP Team was able to leverage off the unclassified Internet, as well the classified SIPRNET.

The actual development of the SHARP project required the adoption of a series of unique development techniques. This chapter reviewed the concept of the three-year rule, in that no IT system should stagnate in its present state for more then three years. This rule reinforced the concept of fresh development, reusability, competition, and modernization of legacy software products. A series of commercial and DoD based examples were given. This chapter also reviewed the concept of leapfrogging development, in that one team might start a new iteration of development while another team is still working on the previous stage. A part of the old team might be transferred to assist the new team with integration and continuity, as well as to complement the development knowledge base of new team members. This method of leapfrogging personnel and development increased productivity and ensured a smooth transition through from one stage of development to the next.

A detailed discussion was given regarding the implementation of the Wedding Development Method, as introduced in Chapter 2. The Wedding Method centered around the development of a main system module referred to as the Main Menu Module. Upon the completion of the Main Menu Module, consecutive modules are then built onto the system likes spokes on a wheel. This method permitted and encouraged independent accomplishment of multiple simplified tasks, in contrast to the development of the entire system in a single effort. The Wedding Method permitted the review of requirements by

small independent teams. Each of these teams was able to review the requirements from a different angle and thereby ensuring completeness and continuity to the comprehensive project.

The Wedding Method relied on a simplification of the project down to its root level, in parallel with Einstein's theory of simplicity. This theory proposed that a problem should be broken down into its simplest form, without removing any of the essential body of the problem. The SHARP Team worked to break the project out into a series of smaller projects, without removing any functionality. This breakdown actually increased functionality of the project by permitting a greater access to each module as an independent entity.

The chapter concluded by comparing the SHARP development with *"The Cathedral and the Bazaar"* essay. While the *"Cathedral"* essay was not used as a basis for the development of the SHARP project, it was ironic how well the methods paralleled each other. The success of the SHARP development relied on an open development environment, a large pool of testers, a strong base of subject matter experts, and a drive to complete the project. A strong customer-client relationship was essential to ensuring the success and viability of the program. The *"Cathedral"* encouraged rapid releases, in harmony with the SHARP Team releasing new versions on an almost bi-monthly basis. Finally, the military encouraged personnel to step away from a project as they reached a "burn-out" phase. When I completed my efforts on the SHARP Development, it was best to walk away to permit a new series of qualified developers to take my place. The *"Cathedral"* encouraged developers to know when to leave, and when to find a new project to work on.

The development of the SHARP Project was not just the creation of an automated system. The development of the SHARP Project was the tasking of professional military personnel to build a tool that they would personally take into combat, that would be used to reduce their workloads, and that would reflect on their conduct as Naval flight personnel. The intimate relationship of the team to the client encouraged the development of a more quality program, the frank feedback from users, and the trust of both sides to make scarifies to ensure project completeness. Such a relationship would be difficult to establish outside of the military environment for such a project.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMPARISON AND CONTRAST OF DATA COLLECTED

A. REVIEW OF THE REQUIREMENTS OF NAVAL AVIATION READINESS

Naval Aviation Readiness is based on a series of classified and unclassified documents and instructions that govern short and long-term flight operations, to ensure the highest level of combat readiness. These instructions and documents reflect the needs and mandates of the authoritative chains-of-command. Most of these instructions and documents are issued through independent channels at irregular intervals, and reflects the unique requirements of seventeen aircraft models and over twelve air wings. The core of Naval Readiness is based on the concept of "P4 + WARTS"

B. REVIEW OF THE REQUIREMENTS OF THE SHARP SYSTEM.

The SHARP System was primarily developed to support the requirement to report Naval Aviation Readiness. The secondary requirement of the SHARP System was to automate, track, and report all aspects of Naval Aviation Operations.

C. PRESENTATION OF COMPARISON OF STANDARD SOFTWARE ENGINEERING MODELS AGAINST THE SHARP MODEL AND COMPARABLE AVIATION OPERATIONS SYSTEMS

1. SARA

a. Background

The SARA System was developed in 1996 by the McDonald Douglas company, at the request of the United States Marine Corps. The USMC was searching for a software automation tool to reduce the increasing mishap rate of the newly acquired AV-8 Series Harrier Aircraft. McDonald Douglas, and later the Boeing Company,

developed SARA to exclusively support the flight operations of the AV-8, and later with additional functionality to cover other aircraft type model series. In an attempt to reduce aviation mishap rates, SARA was marketed as an operational risk management tool, to highlight potentially unsafe flight scenarios through a set of user customized rules.[74] [75] Testing and evaluation found SARA to be lacking critical data validation rules that jeopardized data integrity.

b. Specifications

SARA is a PC-based application, currently released in series version 3.x. The SARA System was originally designed using Microsoft Access 2.0 as the Front End GUI, Logic Module, and Supporting Database. Despite current and emerging technology, the SARA development team has elected to maintain the development product in Access 2.0 with external supporting modules in C++. SARA boasts over 450 form objects, 800 query objects, and 100 report objects with embedded code and external modules. Due to the chosen development language and infrastructure, it is difficult to generate accurate system specifications.[76]

c. Requirements Search

The initial requirements for SARA came from the USMC, from a broad request to generate a system capable of reducing AV-8 mishap rates. Additional program requirements were provided by the Marine Corps Safety Office, in collaboration with

---

74 Phone con, 20 Sep 00, Robert S. Greenfield, Boeing SARA Software Engineer and System Consultant, St. Louis, MO, 314-232-2601

75 SARA Information Home Page, http://www.boeing.com/defense-space/military/aerosupport/sara/sara.htm

76 SARA Overview, the Boeing Company, On File with the SHARP Development Office.

squadron users, members of the Marine Chain-of-Command, and subject matter expertise by development members. In an attempt to win contractual agreements with other branches of the armed services, the SARA team gathered requirements from USN, USAF, and USCG Aviation Safety Offices for integration into future versions of the system.

d. Development Model

Due to the fragmented infrastructure of the Boeing Company, the development environment and requirements of the system, and the loss of essential team members, the SARA Product was designed using a series of different development models. The team consists of twelve members, of which only two members are actual developers and engineers. Initial development cycles were based on quick solution requests with "no formal methodology at the base."[77] Depending on the system requirement, module of development, and who was doing the actual design, a mixture of development methodologies were instituted. Engineers were given the flexibility to work with the methods they were comfortable with, based on the task assigned.

e. Dissemination

SARA is initially distributed to requesting commands via an installation CD. Incremental updates are distributed by the Internet, via a secure web site hosted by the Boeing Company. Incremental updates can range in size from 50 Kb to 25 Mb.

---

77 Phone con, 20 Sep 00, Robert S. Greenfield, Boeing SARA Software Engineer and System Consultant, St. Louis, MO, 314-232-2601

f. Future

The future of SARA is in doubt due to recent investigations and evaluations by the SMART-R assessment team. SARA continues to operate in a 32-bit environment using a 16-bit software development tool. The Program Manager admits that there are commercial applications of the SARA product, but there are no active efforts to market the product. There are considerations toward updating SARA from the Access 2.0 environment, but there are no active efforts in place to make this update.[78] In the light of the SMART-R assessment, the USMC is re-evaluating its contract with the Boeing Company for the SARA product. There are no plans by any other branches of the armed services to purchase SARA.

2. SQOM

a. Background

In 1987, a team of Israeli Air Force Officers developed a first generation aviation automation system to track flight operations for land based Israeli Flight Units, titled SQOM (referred to as PAMOT or POMOT in Hebrew). SQOM was deployed and evaluated for approximately seven years to two fighter squadrons. After that period, SQOM was evaluated for future expansion and redesign to meet the new needs of the IAF and for marketing to foreign military services. Since 1996, efforts have been made to market and integrate the SQOM-2 to the United States Navy.

---

78 Phone con, 20 Sep 00, Ken Bloms, Boeing SARA Program Manager, St. Louis, MO, 314-233-9929

b. Specifications

The original SQOM version was maintained using a DEC/VAC system, while the SQOM-2 system was written in a proprietary language referred to as "Freedom", designed by members of the SQOM and ISYS development team. Freedom is a development environment capable of constructing a powerful database driven, object oriented program, capable of interfacing in the IT-21 environment. The database format is fully compatible with all major relational database.[79] Due to the proprietary nature of the system development language, it would be difficult to determine a relative scope of the product.

c. Requirements Search

SQOM was developed around the IAF Concept of Operations. SQOM was market to the Navy as a COTS product that would be modified to meet the needs of the local concept of operations. Both the developers and users could accomplish this 0customization through front end or back end system manipulation, depending on the level of customization. In an attempt to gather U. S. Navy requirements, SQOM was deployed in one naval squadron for a period of 45 days. Before this deployment, the engineers resided with the squadron and made customized changes to the product.

d. Development Model

SQOM was developed using a standard spiral development model, in parallel with the Freedom Development Environment. System subject matter expertise comes from Israeli Air Force Officers as well as ex-Naval Flight Personnel.

---

[79] Phone Con, 21 Sep 00, Stanley F. Bloyer, SAIC Navy Marine Corps Project Manager, Arlington, VA,

e. Dissemination

Due to the size, complexity, and customization of the SQOM system, installation and dissemination can only be done on-site by SQOM Installation Trained Personnel.

f. Future

At the completion of the SMART-R assessment, SQOM received negative marks for its customizability, integration hurdles, and ease of use. Due to the complexity of the SQOM system, the proprietary nature of the development environment, Foreign Service security issues, and the inability to meet the United States Navy Concept of Operations, the DoD elected not to purchase the SQOM system and has terminated its relationship with the development team. The SQOM Team continues to market its product to other armed service branches throughout the world. At present, no foreign service has purchased the SQOM Tool for its armed services. The Israeli Air Force continues to use SQOM version 1.0.

3. NALCOMIS

a. Background

The history of NALCOMIS can be broken up into two parts: The development of the initial Legacy OMA product in 1991, and the follow on Optimized version of OMA in 1998. NALCOMIS was designed to meet the automated maintenance requirements of AV3M and the NAVFLIRS Flight Record. NALCOMIS was never intended for use as an operational management tool, nor was it intended for use in computing Training and Readiness values. Currently NALCOMIS and its related backbone are deployed thought out the fleet, in every aviation flight and support unit. Its

database is accessible via an exclusive worldwide network that links squadrons to supply and support branches of naval aviation.

b. Y2K

Due to the age of the Legacy version of OMA, there was a period that NALCOMIS was found non-Y2K compliant. Dates were stored in single and two digit year format, and four digit Julian date format. OMA required an estimated 2% change in the lines of code to ensure compliance. OOMA is fully Y2K compliant.[80]

c. Specifications

The Legacy version of NALCOMIS OMA was engineered with Informex 4GL, directly coupled to an Informex proprietary database, residing on a UNIX platform. The database format was written to comply with the 80-column card format required by the NAVFLIRS document and the up-line reporting AV3M database.

The Optimized version of OMA was written using the Powerbuilder development suite, linking to a Sybase formatted database, on an NT platform. Both systems were written to support relational databases.

OMA was conceived in 1991, and fully integrated to the fleet by 1997. OOMA was started one year later in 1998, and not scheduled for full fleet integration until 2004. The original version of OMA consisted of over 800,000 lines of code, while the Legacy version had only one million lines of code. Most of the increased lines of code were accounted for by added functionality to the Optimized version, in tandem with

---

[80] Phone con, 09 Aug 00, Tom Klooster, NALCOMIS OMA Division Head, Chesapeake, VA, 757-523-8146

efficient coding practices. OOMA has over 3206 system objects, 473 stored procedures, and 50 form views in 169 C-Files.

d. Requirements Search

The primary purpose of NALCOMIS was to support the maintenance wing of Naval Aviation. The requirement search was based on publications, work habits, and the concept of operations of the Maintenance Department. No attempt was made to mirror the requirements to the Operations Department. The subject matter experts to the project development team consisted of active duty and retired members of Maintenance Departments, complemented by members of Naval Aviation Supply and Acquisition, and Space and Naval Warfare Center. The Legacy Requirements Documentation consisted only of a general functional description of the project. The Optimized Requirements Document consisted of approximately 500 lines of general bullet requirements.

e. Development Model

NALCOMIS was developed using form of the traditional incremental spiral models. The engineering team would take the existing requirements, and develop a product for review by the fleet development team. The initial requirement base came from a functional description by NAVAIR, based on technical publications on file. No attempt was made to further define the requirements. The FDT would then test and evaluated the product for completeness, and make recommendations for changes. Due to the development infrastructure, the actual requirements document was never refined or modified based on recommendations by the FTD. In the end, the FDT submitted over 4000 Trouble Reports or Change Proposals to the system. The OOMA product consisted

only of improvements to the user interface, improved language, and data replication. There was no real increase in user functionality.

Due to the government bureaucracy involved in software development, members of the NALCOMIS team admit that the project suffers from a significant overhead burden. Attempts are being made to develop OOMA in parallel with the maintenance of Legacy OMA. Such a practice requires a large support staff to ensure that both development teams and customers receive proper support for their particular product.

f. Dissemination

Due to the size and complexity of the system, NALCOMIS can only be distributed to the fleet via CD. The database network permits real-time updating of the NALCOMIS database. The program footprint requires approximately 20 Mb.

g. Future

The short and mid range outlook for NALCOMIS includes the development and distribution of the OOMA product to replace the Legacy distribution. NALCOMIS has been accused of suffering from the Second System Effect defined as, "When one is designing the successor to a relatively small, elegant, and successful system, there is a tendency to become grandiose in one's success and design an elephantine feature-laden monstrosity."[81] NALCOMIS has been accused of breading the preverbal software dinosaur that has become so big and integrated into all facets of operations that you can not kill it, but you hope that one day it will become extinct as it

---

[81] Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999. Attrib. to Fred Brooks, *"The Mythical Man-Month"*

can not adapt to future changes. Due the future emphasis on Enterprise Resource Planning (ERP), Mr Klooster fears that a COTS product will replace NALCOMIS, developed by SAP, by 2010.

4. Patriot-Excalibur

a. Background

The Patriot Excalibur System has a very similar history to the SHARP Program, but while the SHARP System was developed to serve Naval Aviation, Patriot Excalibur was designed to automate Air Force Flight Operations. A group of Air Force Officers, complemented by a small team of government employees and contractors, worked to develop a PC based, database driven, customized tool to track and report all relevant points of flight operations. Patriot Excalibur was first introduced to the Navy in the early winter of 2000 as the development team was preparing to distribute an initial release to members of operational flight wings. The Air Force is currently field-testing version 1.02.

b. Specifications

Patriot Excalibur is developed using the Microsoft Visual Studio and Visual Basic 6.0. The program has a relatively small footprint at only 400,000 lines of code for its initial iteration. The preliminary version of Patriot Excalibur is developed using the Microsoft Database, but is investigating the upgrade potential of the SQL Database.

c. Requirements Search

Patriot Excalibur was developed because there was no comparable solution to meet the requirements of Air Force. The development team was comprised of

pilots, aviation support officers, and a civilian staff assigned to support flight operations. These subject matter experts were able to take personal and professional knowledge, coupled with support documentation and instructions, to generate a baseline set of requirements for an initial prototype development. Pending the success or failure of the initial versions of Patriot Excalibur, requirements will be modified to ensure that a successful program is developed to meet the future needs of the exclusive operations of the United States Air Force.

d. Development Model

Patriot Excalibur continues to be developed using a spiral RAD development methodology. Initial versions of the program serve as prototypes for future iterations of the system.

e. Dissemination

The Patriot Excalibur system plans to distribute the program through a limited release on CD, complemented by open distribution via an Air Force hosted Internet Web Site. The small size of the system permits dissemination through a large variety of electronic media, including the E-Mail and the SIPRNET.

f. Future

The Patriot Excalibur Project is in its infancy, and it is difficult to determine the long-range direction of the project. The program was given a preliminarily evaluation by the SMART-R Assessment Team, but the results were found to be inconclusive due to the scope of the project and limited operating history.

5. SHARP

a. Background

SHARP was originally developed in 1996 to serve as an exclusive operations automation tool for HELANTISUBRON – SIX. SHARP was brought to the attention of COMNAVAIRPAC during the failed evaluation of the SARA system. Once adopted by AIRPAC and AIRLANT as their joint TYCOM aviation automation system, SHARP was quickly modified to support the flight operations of all seventeen type model series aircraft. This version was ultimately disseminated to over 330 operational naval squadrons and detachments worldwide. A more detailed description of the SHARP background was provided previously in this thesis.

b. Specifications

The SHARP System was developed using the Microsoft Visual Studio and Microsoft Database. The decision to use the Microsoft Suite was a factor of authorized software products, available developers, and the best match to the system requirements. The interim product consists of over 750,000 lines of code in a relational database, with over 25 independent modules, in over 800 code and support files, with tens of thousands of unique procedures, functions, and classes.

c. Requirements Search

Initially, the primary requirement for the SHARP system was to engineer a product to support the newest release of the Training and Readiness Manual and report the combat readiness of all operational naval aviation units. While the initial scope of the requirements were limited to supporting and returning a command's readiness value, it

quickly became evident that squadron users desired something that would better reflect the complete scope of squadron operations in the electronic environment.

In an attempt to better understand the requirements of Naval Aviation from the fleet perspective, a team of over thirty naval aviators were brought together in North Island, CA to form the ATRRIS IPT. This team of aviators pooled their combined knowledge of naval aviation, operations, training, and readiness to author a complete list of requirements to model squadron operations and revolutionize the operations office. The ATRRIS IPT attempted to incorporate the unique requirements of seventeen different type wings, tempered by the requirements of six different motivating documents.

Aviation subject matter experts determined all system requirements. Those experts were also the ultimate system users.

d. Development Model

The SHARP Product was developed using the Wedding Method, as described earlier in this thesis.

e. Dissemination

In an attempt to distribute the SHARP Product to the fleet in the most efficient and cost effective manner possible, the Development Team relied on the Internet and SIPRNET as a dissemination vehicle. The net provided a reliable method for the transmission of data, program updates, trouble calls, evaluation and critiques, as well as beta-tester bug reports. A baseline CD was also made available for users make initial installations of the product, but was not required as installation sets were also made available on the net.

f. Future

The SHARP Project is currently under transition from version 3.1 to version 4.0. Version 4.0 will incorporate additional modularity, refined object modeling, and increased functionality. A transition from the Microsoft Database to the SQL Database will increase system operating speed, reduce top end limitations to the database structure, and improve system response in the network environment. SHARP was judged overall favorably by the SMART-R assessment team, and has guaranteed the future survivability of the program.

6. COTS Debate

In 1996, with the release of DoD Reg 5000.2-R, military acquisition professionals and defense contractors emphasized the use of COTS products to supplement development efforts.[82] As noted in Chapter 1 of this thesis, COTS products are defined as:

1. Commercial items customarily used for non-governmental purposes and offered for sale, lease, or license to the general public.

2. An item evolved from such an item, as previously stated, that will be available within sufficient time.

3. Items that are standard modifications available in the commercial marketplace or are minor modifications.

4. Any non-developmental item developed exclusively at private expense and competitively sold in substantial quantities to non-federal government agencies.[83]

---

82 Department of Defense Regulation 5000.2-R, *"Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs"*, 15 March 1996.

83 John Foreman, *"On the Front Lines of COTS - Lessons Learned, Speculation for the Future"*, Briefing slides courtesy of the Software Engineering Institute, Carnegie Mellon University, May 8, 1998. http://www.sei.cmu.edu/cbs/cbs_slides/stc98/frontlines/index.htm, Slide 12.

Shortly after the release of DoD 5000.2, a prominent defense contractor spoke out with a controversial statement against the mandate of COTS solutions in the unique military environment:

> "The requirements for computer systems within those DoD programs in which Mercury Computer Systems is typically involved cannot be satisfied by Commercial-Off-The-Shelf (COTS) computer technology alone. There are various requirements that are simply beyond the capabilities of the system designed, built and sold in commercial markets. DoD, in an attempt to meet the military cost reduction requirements of the Clinton administration, have begun stipulating COTS technology be used in all applicable applications. It is our contention that requiring COTS components be itself will not produce the desired magnitude of cost reduction nor improvements in 'time-to-development.'"[84]

Mercury executives stated that military environment required specialized tools meet the unique demands of defense operations. While commercial products could satisfy administration and office automation requirements, there are no commercially available products that could predict aviation readiness or automate flight operations without changing the Naval Concept-of-Operations.

A number of contractors attempted to repackage and distribute software that was not a true COTS product, but that could loosely be referred to as such, in an effort to increase marketability. SQOM was marketed to the Navy as a COTS compliant solution to aviation automation, but was oddly based on the Israeli Concept-of-Operations. For SQOM to work for the U. S. Navy, it would require significant redesign in structure, flow, and logic. The ISYS team referred to that redesign as "customization". That customization could only be accomplished by the development team and not by the user.

SQOM satisfied COTS Definition #1 by making itself available for sale to the general public. Unfortunately, with the exception of the Israeli Air Force, no other branch of service or country has implemented the SQOM product to meet their aviation automation requirements. SQOM failed Definition #2 as it was unable to create an optimal product within the time set down my SMART-R Assessment Team. SQOM failed Definition #3 because modifications mandated by the U. S. Navy would completely change the logic pattern of the product and render it useless to other commercial clients. Definition #4 required that any product be developed at private expense, while the ISYS Development Team requested and received $1.8 Million from the Department of the Navy to make the required changes to field the SQOM Product.

The Boeing Company has attempted to market SARA to the Navy as a COTS product, based on its early success in the Marine Corps. SARA satisfies Definition #1, but is not viable to any other customer due to its exclusive logic to support the Marine Flight Plan. SARA satisfies Definition #2, as its format could theoretically be changed in sufficient time to meet the Navy's requirements, but were not attempted during the SMART-R Assessment. Modifications to the SARA product would completely change the logic flow of the program, and would not be considered minor. The base logic to determine Training and Readiness is unique for each branch of service, resulting in a failure of Definition #3. Boeing has informed the U. S. Navy that it would make no changes to the SARA product without contract or payment for services. The U. S. Navy

---

84 *"White Paper: Applying COTS Products and Services to Major Defense Programs"*, http://www.mc.com/COTS_folder/cots_mtb/cots_mtb.html, Mercury Computer Systems, Inc, 01 Jun 1996.

has informed Boeing that it would not purchase SARA unless it was modified to meet the unique demands of Naval Aviation, failing Definition #4.

Future efforts to refine defense system automation will compete with the new challenge of Enterprise Resource Planning. Future products will no longer be completed systems, but rather a set of development tools that when combined will create a powerful automation suite. These ERP products will be viable commercial development tools, satisfying Definition #1. The resulting suite will logically satisfy Definition #2. As the ERP products are intended to be modified, Definition #3 will be satisfied. ERP products will be developmental items except from Definition #4. There is nothing to guarantee that these ERP products will improve software development, automation efficiency, or that they will even be able to replicate the requirements of Naval Aviation Operations.

COTS implies using a commercially proven product that can be seamlessly integrated into the existing defense environment more efficiently then the development of a virgin product. The military has a number of divisions that are run like a business, where commercial software products can be integrated quickly into the operating process, but there are far too many other divisions that require uniquely tailored products that meet the specialized needs of combat operations.

D. CHAPTER SUMMARY

The development of the SHARP Project was fraught with complications and controversy due to a number of competitive products that attempted to eliminate the SHARP System. During the course of development, SHARP and its competitive products faced a barrage of evaluations and assessments to determine the most viable solution. Each of the products were evaluated for their ease of use, completeness to

project requirements, cost effectiveness, defense technology compliance, and long range survivability. This thesis reviewed some of the findings of the assessments in parallel with the comments by the developers and their published literature.

SARA, the Boeing Company product, was based on an old 16-bit software environment, critically limiting its functionality and long range survivability. The program lacked essential data validation required by Naval Aviation Documentation. Team members admitted that the SARA product was developed with no formal methodology, and that engineers were given the luxury of developing in whatever environment they felt comfortable. SARA was determined to not meet the significant requirements of Naval Aviation and even lacked the critically functionality for Marine Corps Aviation.

SQOM was developed by the Israeli Air Force, and then marketed to the U. S. Navy as a solution for aviation automation. SQOM attempted to market itself as a COTS product despite failing to meet the basic definition of a COTS product. SQOM was developed using a proprietary language and database format, untested outside of the SQOM / ISYS environment. Security concerns, coupled with an unproven development language and extensive customization requirements resulted in a negative recommendation by the SMART-R Assessment.

The NALCOMIS product has continued to improve over time, though be it a few years behind current technology. The military bureaucracy would result in a product that would take over seven years to develop and deploy to the fleet. Comparable systems were capable of being deployed in less then nine months by leveraging off of existing infrastructures and taking advantage of optimized and refined development techniques.

NALCOMIS was marketed as an aviation automation tool, but was customized towards supporting the Maintenance Department over the Operations Department.

In an attempt to meet the unique needs of the United States Air Force, members of the service recently came together to build a new aviation automation tool called Patriot Excalibur. The Air Force offices and staff members relied on rapid prototyping, spiral development, and subject matter expertise to develop a potential replacement for legacy systems. It is too early to forecast the future of the Patriot Excalibur system, but initial results indicate that the project would make a successful working prototype for future development. Much of the Patriot Excalibur development is in parallel with the SHARP development process.

The chapter concludes with a discussion of COTS products and the attempt for many contractors to repackage their products to qualify under the DoD Reg 5000-2R requirement to integrate COTS into military acquisition. A review was given of the four basic definitions of COTS and a cursory evaluation was made of commercial products that refer to themselves as COTS. It was evident that the SQOM product was not to be considered as a COTS product based on the basic definitions. SARA also failed to classify as a COTS product.

While administrative and executive COTS products may meet the business requirements of an operational command, it would difficult to find an acceptable operational automation product that would not require extensive customization to meet the unique needs of Naval Aviation. One of the failed methods of COTS was purchase a system that marketed itself as an operational solution, only to require users to modify their own concept-of-operations to make the solution work. Combat, conflict, and

aviation operation will not change to meet a software solution. Solutions should be developed that mimic and automate the existing structure without modification or compromise. Such a solution would be difficult to find in a true COTS product.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The development of the SHARP Product was not by happen chance. Its development was the result of a refined development method, professional subject matter experts, uniform service member trained and experienced in software development, and an intimate developer-client relationship.

The SHARP engineering team made an extensive search and discovery for potential development methods, only to realize that none of the existing formal models in themselves would meet the development requirement. The team was looking for a method that would permit rapid application prototype development, encourage efficient testing and requirement review, client interaction, and parallel modular development. Rather then settling for an inferior model, the SHARP team devised an ingenious method, referred to as a Wedding Method for its similarity to a multi-layered Wedding Cake. The Wedding Method encouraged the breakdown and simplification of the system to take advantage of the project in its smallest component possible. The method also permitted the parallel development of multiple modules, increasing productivity, testing, and customer critique.

Every member of the development team was a designated of naval aviation with service in deployed operational commands. Their professional knowledge was critical to requirement analysis, testing, and product evaluation. Each of the members of the team had an intimate relationship with a particular aviation type wings, as well as members within that wing. That accountability and name to face relationship encouraged a greater level of performance and customer dialog with the development team. Far too many

programs are submitted to the fleet with misinterpreted requirements due to a lack of subject matter experts at the base level of development. The SHARP Team was dedicated to provide a product that they would be proud of, that would represent the fleet, and would withstand the scorn and criticism of doubters and critics. SHARP was developed by military members ordered to build a product. They did so in the proud service of their military.

As each member of the SHARP Development Team maintained contact with the fleet, a special relationship was established that encouraged frank feedback on the development of the project. Each and every member of the SHARP user group had the direct phone number to the AIRPAC development office, e-mail addresses of team members, and the home phone numbers of critical members of the team so that they could be contacted after hours and on weekends. Users realized that their inputs directly effected the development of the project. They were ensured that every idea would be taken into account and implemented into the next iteration of the project because, "If the fleet though that it was important enough to be in their program, then it should be in their program." Developers need to remember that a program is not their property, it is the property of the customer.

Uniform service members have been trained and educated in myriad of professional subjects including operational proficiency, leadership, management, and teamwork. A few select members are specially trained in project development and software engineering, courtesy of military and civil post-graduate education. The SHARP Development Team was comprised of officers and enlisted uniform service members who were demonstrated professionals in the field of project development and

information technology. The Naval Postgraduate School – Software Engineering Curriculum was directly responsible for the education of some members of the development team. The application of military provided academic knowledge to defense service projects functions only to justify the importance of an educational institution like the Naval Postgraduate School.

The success of any project is a true factor of the strength of the development team, their ability to adapt and overcome difficult situations, to persevere despite a lack of resources, and to seek out alternatives when the existing structure no longer applies.

B. ANSWERS TO RESEARCH QUESTIONS

1. Primary Question

Question: Can traditional software engineering methods meet the needs of the military for small development projects or should the military examine a return to Rapid Prototype Development?

Critique: The "Mongolian Hordes Technique" or the "Chinese Army Technique", defined as - assigning a large number of inexperienced programmers to a job that would be better preformed by a few skilled ones. The term was first used by Dr. Fred Brooks in his book "The Mythical Man-Month", Chapter 3.[85]

Answer: No. Traditional software engineering methods continue to fail to meet needs of the military for small development project requirements. The NALCOMIS project is a prime example of a development that has been hampered by the military bureaucracy. If the project was broken down into smaller components it might very well

---

[85] Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

have been completed in a more cost-effective manner, in less time, and more reliably meet the requirements of Naval Aviation.

Many commercially viable software systems are based on the redevelopment of legacy systems. Previous iterations eventually serve as working prototypes for future development. To remain competitive and to take advantage of leading technology, commercial developers re-issue systems in phase with technological advancements. A strong rule of thumb should be that - "No IT systems should remain in its present state for longer then three years." If the military is to remain technologically viable, it needs to review and change its development habits to build and deploy software solutions more efficiently to the fleet.

Existing instructions, requirements, and regulations mandate a considerable amount of overhead to each project. They are based on methodologies, techniques, and technologies that are a few years old. In the world of software engineering, where technology is known to double every eighteen months, an instruction that is based on a concept already three years old is actually separated by two generations from present technology. None of the current DoD development requirements are certified by civilian standards, nor do any intend to become certified in the future.

The DoD tradition of gathering a large body of personnel together to develop software, in keeping with the Mongolian Horde Technique, must be abandoned. The SARA Development Team, with a roster of 17% engineers and developers and 83% overhead personnel is absolutely unacceptable. Streamlined, efficient, rapid prototype development is a commercially proven methodology that would save the Department of Defense precious time and resources. These working prototypes could be developed,

deployed, reviewed, and modified at more efficiently then with traditional practices. Failure to adopt commercial practices will only result in the Department of Defense playing "catch-up" to industrial technology. It is an embarrassment that the Department of Defense should play second fiddle to industry when it comes to defending the nation. It is by no surprise that a team of dedicated uniform service members could do what others had not. History reminds us that the Mongolian Hordes were eventually concurred by small bands of rebels and bandits.

2. Subsidiary Questions

a. COTS to the Fleet

Question: Can COTS products be modified and customized to meet the rapidly changing needs of the military, specifically aviation readiness reporting?

Answer: No. COTS implies that it product viable in the commercial market. There is no commercial market for aviation readiness reporting. That concept is exclusive to the military market, and the logic unique to each branch of service. No one tool can be built that would meet the needs of each branch of service, much less across international borders, unless services would be willing to dramatically change their concept-of-operations to match the new tool. Any tool that would be modified to track the logic of all potential customers would not be considered COTS.

b. Is modified COTS still COTS?

Question: How far can you modify a COTS product and still be considered COTS?

Answer: The four-part definition of COTS states in Rule #3 that modified COTS are "Items that are standard modifications available in the commercial

marketplace or are minor modifications." Many COTS products are modified or customized at the user level to ensure compliance with the client's concept-of-operations, as long as such customization is part of the standardized product. Customization at the code level is not a minor modification. Developer induced alterations to repair baseline shortcomings does not certify a product as COTS. If modifications are made that are not part of the standard commercial release, then a product is not COTS.

This thesis makes a great effort to disprove the concept of COTS to replace the unique system requirement of Naval Aviation Operations. What is done in Naval Aviation is unmatched to anything in the commercial sector. The only true solution to the demanding requirement of aviation operation automation is the development of customized rapid prototype systems.

c. RAD Models

Question: Does a viable model exist to engineer RAD, and can RAD be certified as a valid development model in the military environment?

Answer: Yes. Of the traditional development models, the Exploratory Programming Model, Prototyping Model, and Spiral Model all adapt well to rapid application development. They are built around the concept of cyclic stagewise development, permitting multiple revisits to the requirement stage. Each of the models encourages customer review and evaluation, and are adaptable to small system development. None of these development models are endorsed by current DoD Development Instructions.

The SHARP System was developed using a modified version of the Spiral Model, referred to as the Wedding Method. The Waterfall Method is the preferred

development model of DoD Software Engineers, known for its rigidity and complete requirements at the start of the development process. Due to the ever-changing atmosphere in the military environment, software developers need to look towards more flexible development practices that permit interim changes to a project. A variety of RAD methods are reviewed by professional publications as potential software development solutions. The DoD must review existing RAD methods for applicability in the military environment, as either pure development methods or hybrid methods like Extreme Programming.

d. Reusable Code

Question: Is reusable code beneficial in the development of RAD applications?

Answer: Yes. RAD must rely on a series of practices that improve development efficiency. Reusable code provides a library of resources that can be applied to the software development process. Most code packaged would have already been tested and proven in other previous systems, improving the development effectiveness.

The SHARP Team strongly relied on reusable code to decrease overall development time of the project. Reusable code included third party code packages, OS based dynamic link libraries, and team developed repeating modules. Much of the code relied on centralized or repetitive modules to decrease the total number of lines of code and increase overall program operating efficiency. SHARP relied on reusable Error Checking Modules, Database Access Modules, Keystroke Validation Modules, and other proprietary modules to decrease the development requirements of the project.

Many DoD development teams are so fragmented that they do not share code with other development groups. The DoD would be better served if it would create an archive or library of reusable code available for standardized integration into other projects. Such a library would provide an invaluable resource, decrease development time, and encourage standardized component development.

e. Subject Matter Experts

Question: How critical are subject matter experts in the requirements, developmental, and testing phases of software engineering, and if they are critical, why are more contractors not using them?

Answer: Subject Matter Experts are critical to the success of any project development. SHARP relied on a series of subject matter experts to provide quality assurance to every aspect of the process. SMEs composed the requirements, developed the prototype, were actively involved in the testing and evaluation process, and were finally part of the deployed user group. SMEs ensured that the final project professionally automated the aviation environment. Without access to SMEs or without SME participation on the development the SHARP Project would have been making assumptions at requirements and potentially provided a product that may have not meet the ultimate needs of Naval Aviation.

The SHARP Team has made its Subject Matter Experts available to other development team to encourage fleet interaction in other development processes.

f. Streamlining DoD Software Development

Question: How can the Department of Defense (DoD) streamline its software development protocol to more efficiently produce a product?

Answer: The DoD continues to rely on outdates software development practices for the engineering of small and midsize defense projects. The newest DoD Standard – DII COE – is not recognized by civilian accreditation organizations, is congested with overhead requirements, and difficult to understand and comply with. It is unfortunate that the greatest military in the world has lost to the civilian sector its ability to develop defense tools, and mandates that those civilian companies that wish to do business with the military comply with an inflexible requirement that stifles efficiency. Many of its standards are fraught with loopholes, lack supervision and compliance authority, and are based on archaic development principles. The SHARP Team conducted an extensive search of potential development practices certified by the DoD, but quickly realized that all of the existing standards would stifle the development of the SHARP Product.

The DoD needs to review its development standard to seek out current and applicable practices that are compliant with proven civilian practices. Attitudes and comments from the DII COE staff stated a lack of desire to comply with accepted commercial standards. Such an atmosphere is unprofessional and places the DoD at risk of continuing to fall in the advancing practice of software engineering.

g. Uniform Service Development

Question: To what extent should uniform service personnel serve on development teams?

Answer: The SHARP Team demonstrated the ability for Uniform Service Personnel to design and engineer a successful project. That success was reliant on the subject matter expertise, military training in leadership and team building, and education

and training in the field of software engineering of team members. Team members demonstrated the unique ability to adapt and overcome a series of conflicts, lack of resources, and criticism from members of the chain-of-command. Uniform service members developed the SHARP Project out of pride for their aviation community, using personal experience, and professional relationships to build a product that they could use when they deployed to flight operations. Uniform service members have a professional desire to ensure the success of the their projects.

h. Uniform Service Education

Question: Should more uniform service personnel be assigned to advance degree education concurrent to their operational commitment, and how can this be facilitated?

Answer: Yes. Uniform service members should be afforded every opportunity to attend postgraduate degree education concurrent to their operational commitments. Due to operational manning shortfalls, many service specialties are unable to take service members out of their communities for two years of education, despite post-education extended service commitments. Many members have opted to pursue post-graduate education through civilian school at their own cost, in fields that do not directly benefit the Navy. Such a pursuit is a waste of military resources, as each member of the military is considered a resource.

SHARP Team members were provided access to Naval Postgraduate School via distance learning channels to pursue a continuing education. The knowledge taught in the evening was directly applied the next morning in the project development environment. The Navy received a direct benefit from the education process through a

more quality project, and service members received the satisfaction of increasing their own personal knowledge. This symbiotic relationship increases the strength of the military knowledge base, increases the morale of uniform service members, and increases the combat readiness of the military by providing a smarter, more intellectual, more professional sailor to defend the nation.

The U. S. Navy, streamlined though the Naval Postgraduate School, should afford more service members the opportunity to pursue postgraduate education through non-traditional means at no monetary expense to themselves. These service members will pay the military back in folds by providing a skilled and educated resource, trained in the unique military environment, and dedicated to national service. These service members will demonstrate their desire to serve by their desire to pursue such an education through non-traditional means during operational commitments. This practice will save the military financial resources by not having to transfer members to postgraduate locations and maintaining optimal manning in critical billets.

C. RECOMMENDATIONS

The Department of Defense continues to struggle with current technology and has built up walls of requirements that stifle aggressive development. For the DoD to remain competitive, it must change the infrastructure that governs software development. The DoD must become an active participant in commercial standards development, and be willing to adapt to new technological changes and methods. The DII COE must pursue commercial certification.

The Department of Defense needs to re-embrace Rapid Applications Development. In times of conflict, developers were able to build products faster,

cheaper, and without bureaucratic overhead. We have lost that edge and hindered developers with excessive oversight and administrative burdens. Many products wind up being produced over budget, over time, and obsolete to rapidly changing technology. The DoD needs to review its development practices to find methods that are more efficient otherwise, they continue to squander precious resources.

The DoD needs to get away from building monolithic large systems that would be better managed as a series of smaller systems. Smaller systems could be more efficiently developed, modified, and redistributed to the fleet through existing channels. If one portion of the program became obsolete, that minor module could be updated and redistributed without requiring a rewrite of the entire system, in keeping with the Three-Year Rule.

The DoD needs to make more subject matter experts available to development teams, to ensure a quality product is developed the first time that meets the needs of the soldier and sailor. DoD and contract development teams need to take advantage real subject matter experts when they engineer a product. You can not ask a high-ranking officer what he thinks about a new software tool and expect to get a reliable answer, when the junior sailor is the one who specializes in the tool.

Where applicable, COTS technology should be incorporated into the Department of Defense to provide a commercially proven automation solution. The DoD should also admit that COTS is not the "Silver Bullet" to all solutions. In cases where there is no commercially applicable product, the DoD needs to look towards exclusive development of unique software solutions. It is unacceptable to the soldier or sailor to provide him with a product that is not based on his concept-of-operation. A new solution should

improve his work environment. Solutions should fit the working environment, not change the work environment to fit the solution.

Uniform service members have expressed desires to their pursue postgraduate education, even during operational commitments. The DoD needs to afford the opportunity to more service members to pursue their education through as many channels as possible. The Naval Postgraduate School should provide as many curriculum choices as possible, via distance learning methods, to Naval Officers deployed worldwide. These curriculums could be tailored such that Officers would be able to apply their military training to their assigned missions. The DoD also needs to institute an annual bonus structure to reward Officers that complete their postgraduate education and remain in military service.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A - QUOTES

A. 1996 Naval Audit Service Report on Naval Aviation

"The Naval Audit Service report on Naval Tactical Air (TACAIR) and Anti-Submarine Warfare (ASW) Flying Hours Program (FHP) contained a recommendation that CNO (N889), together with Commander, Naval Air Force, U. S. Pacific Fleet (COMNAVAIRLANT) and Commander, Naval Air Force, U. S. Atlantic Fleet (COMNAVAIRLANT), update the squadron training and readiness matrix for each aircraft type, document basis for flying hour estimates, and use the updates matrix hours to determine TACAIR/ASW FHP. The training and readiness matrixes joint instruction was to be validated by the Commanders-in Chief and CNO (N889). The instruction was to be used as the basis for programming FHP funds for TACAIR/ASW"[86]

---

86 *"Flight Hour Program Audit"*, Naval Audit Service, Job #950040, 15 Apr 1996.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B - LOOK UP LISTS

A. PMA List

- AAW Anti-Air Warfare
- AMW Amphibious Warfare
- ASU Anti-Surface Warfare
- C2W Command and Control Warfare
- CCC Command, Control, and Communications
- FAC Forward Air Control
- FSO Fleet Support Operations
- INT Intelligence
- LOG Logistics
- MIW Mine Interdiction Warfare
- MOB Mobility
- MOS Missions of State
- NCO Non-Combative Operations
- NSW Naval Strike Warfare
- STW Strike Warfare
- WAT Weapons and Tactics

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C - SOFTWARE ENGINEERING TECHNIQUES

While the following definitions may serve as a broad representation of the most popular models of development, they in no way represent all of the potential models that can or would be employed by software engineers.

1. Build and Fix Approach

The Build and Fix Approach is a very simplistic model of the developer building the entire product, delivering it to the client who then in-turn requests changes. The developer would repeatedly exchange the product with the client with incremental changes, until the client feels that the software can be used productively, as depicted in Figure C.1.

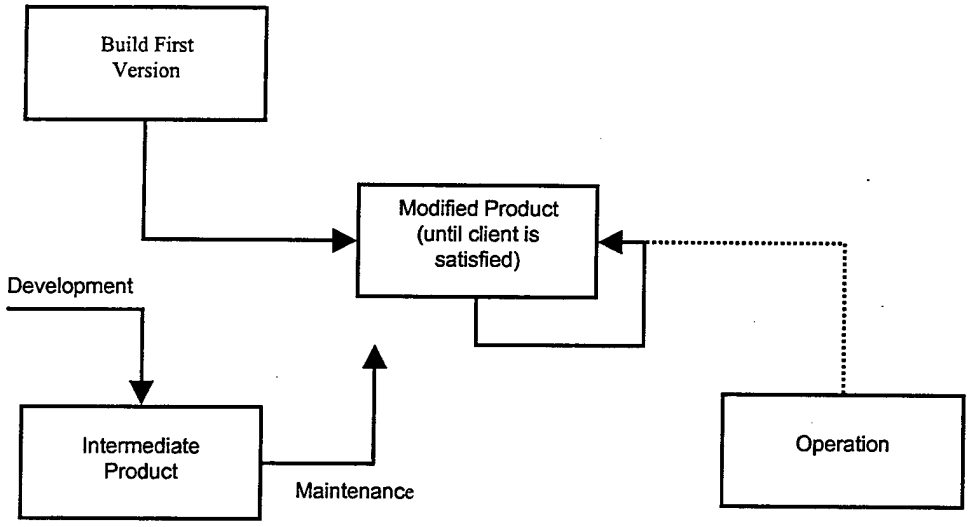


Figure C.1 Build and Fix Approach Diagram[87]

The Build and Fix Approach can also be refereed to as the "Hunt and Peck Method", relating to the chicken in the barnyard that constantly pecks at the ground looking for food. If the chicken pecks long enough, he just might find something to eat;

---

87 David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

if a developer keeps changing the system long enough, he just might meet the customer's requirements.

This method is highly informal, has no real requirements gathering or analysis phase, relies on the customer for testing and evaluation, and can result in great inefficiencies in product completion. The reliance on project completion falls upon the customer to state when they are satisfied. With no real completion goal established, it becomes evident that no real time line for stage completion can be set for this method. An IEEE Workshop Abstract noted that:

> "Large organizations throughout industry face immense problems when they have to adapt their acquired software systems with millions of lines of code to rapidly changing requirements. Far too often, such systems have evolved from an uncoordinated build-and-fix attitude towards software development and suffer from a lack of methodical support during maintenance. The original design intents of the software systems are obfuscated, or worse, have disappeared altogether. It takes immense effort to implement and test changes as the effects on other software modules and the impact on future reuse are hard to predict."[88]

Due to the complexities seen with large system developments, the Build and Fix Approach does not work well and is not recommended against other approaches. In the case of smaller, less formal projects, where an intimate relationship exists between developer and client, the Build-and-Fix Approach may serve well, such as in the case of in-house development.

---

88 Rudolf K. Keller, Background Paper for the DEXA '98 International Workshop on Large-Scale Software Composition held in conjunction with the 9th International Conference on Database and Expert Systems Applications (DEXA'98) Vienna, Austria, http://www.cs.ttu.edu/fase/v8n02.txt, August 24-28, 1998.

2. Stagewise Development

The concept of Stagewise Development serves as the basis for most formal developmental models. Each step of the development process is divided up into distinctly contained units or stages, then executed independently, as depicted in Figure C.2. This approach requires that the developer reviews each step, execute, complete it, verify the results, and then evaluate if the results warrant advancing onto the next step. In the event that changes are required, the developer would return to the phase that required the change, and then progress through each subsequent phase until development completion.[89]

[89] Brian Foote, Briefing on *"A Fractal Model of the Lifecycles of Reusable Objects"* for the Workshop on Objects, Iteration, and Development Process Standards, http://www.laputan.org/talks/fracwash.html, Washington, D. C., 1993.

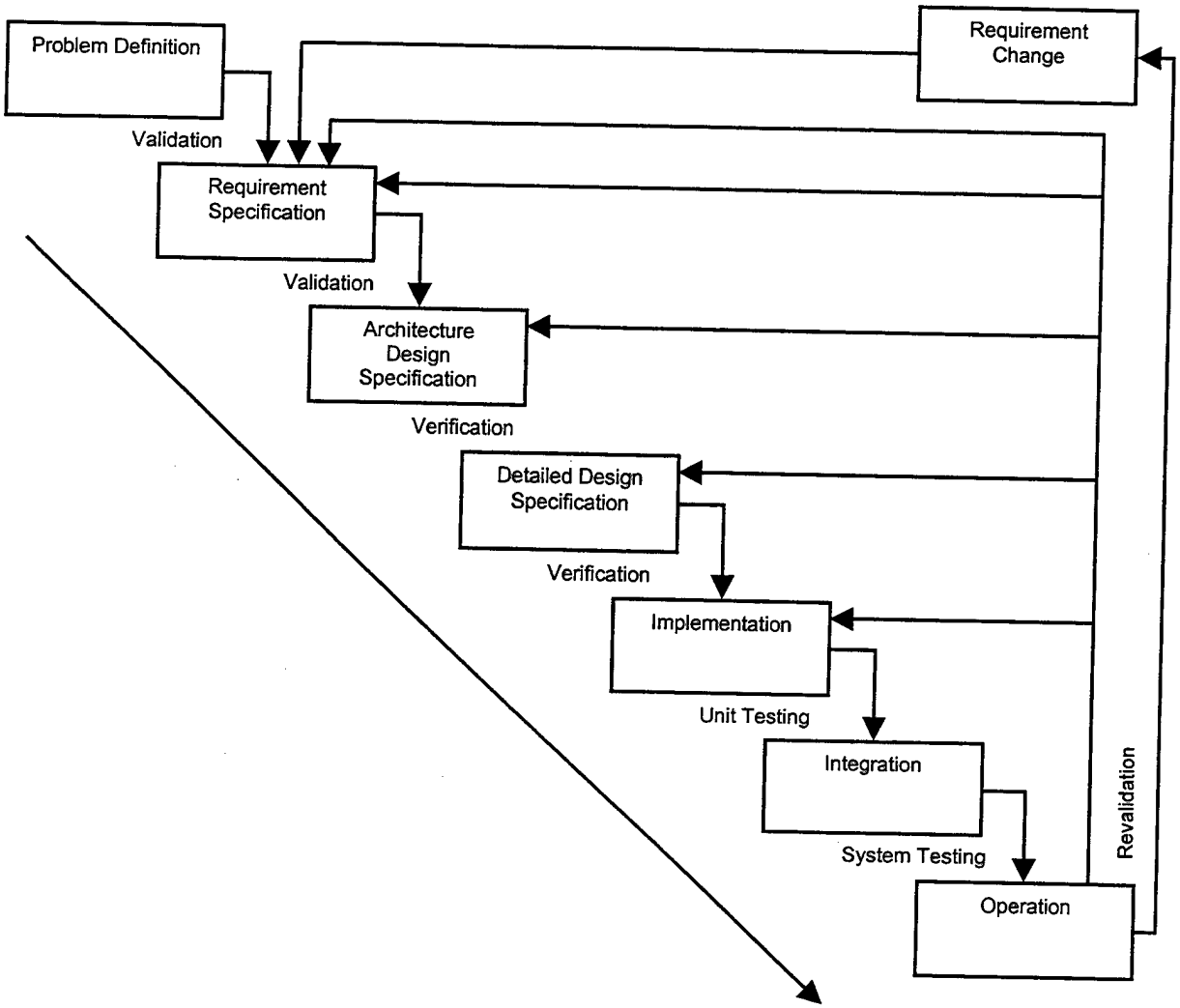


Figure C.2 Stagewise Development Diagram[90]

As depicted in Figure C.2, Stagewise Development is a progressive ladder of evolution from the requirements phase to the operation phase, with validation or verification checks certifying advancement into the next phase. Further examples and definitions of other development methods and models will demonstrate the relationship between other models and Stagewise Development. While the concept of Stagewise

---

90 David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

Development is applicable in the realm of software development, it is also widely used throughout other avenues of engineering. In a recent Environmental Impact Assessment for a Hazardous Waste Management Study in Estonia, the context of Stagewise Development was used as;

> The objective of the feasibility study was to prepare a plan for a stagewise development of a comprehensive nation-wide hazardous waste management system for Estonia for the period 1994 to 2009, based on environmental, legal and institutional considerations.[91]

3. Waterfall Model

The Waterfall Model is one of the most popular derivations of the Stagewise Development Method. In 1970, Winston Royce first introduced the Waterfall Model, fashioned after the concept of stagewise development and based on the other process models in use at the time.[92] It is composed of all of the traditional units of Stagewise Development with the addition of a single level of feedback between each given stage and the preceding one, as depicted in Figure C.3

---

91 Estonian Ministry of the Environment Estonian Environment Fund 1994-1995, http://www.sei.se/seit/english/listproj.html, Software Engineering Institute.

92 A. Macro, *"Software Engineering: Concepts and Management"*, Prentice Hall, 1990.

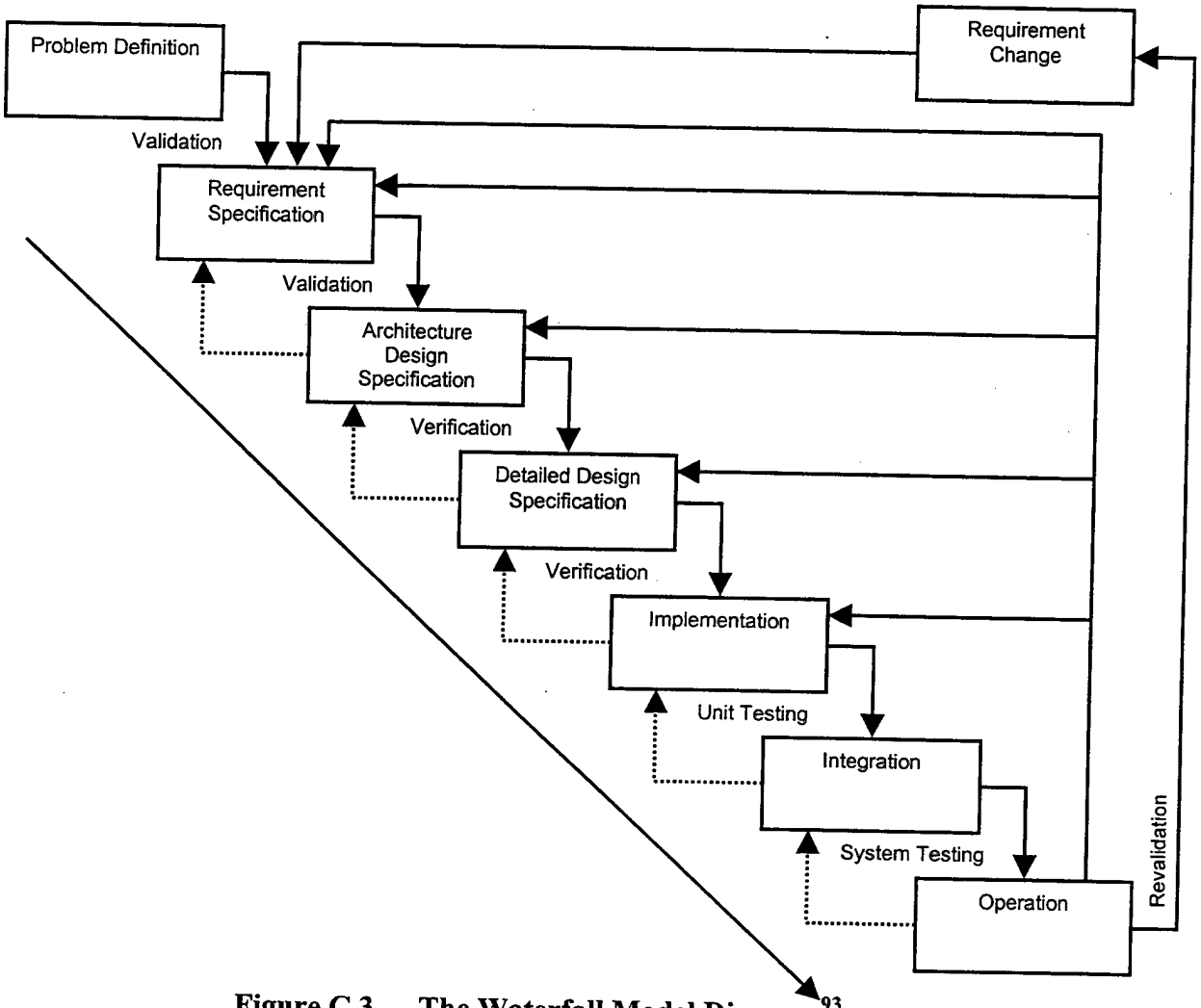


Figure C.3 The Waterfall Model Diagram[93]

The Waterfall Model requires a complete, validated specification of the required functions, interfaces, and performance before the design of the system is actually undertaken, due to the non-return flow of the waterfall. It is extremely popular throughout industry for its rigid format and reliance on structure and formal specifications from the onset. It carries a strong emphasis on fully elaborated

[93] David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

documentation from the earliest of stages of the requirements and design phase. Each phase of development is culminated with a validation or verification study to ensure completeness.[94]

The strength in the Waterfall Model lies in its usefulness as a management tool towards the software development process. Each phase is defined by a set of functions, goals, milestones, and deliverables, making the process highly visible and the project easier to track. Since requirements and specifications are determined at the onset, the project manager is better able to determine his resource needs and establish schedules. Conducive to its formal flow, risk management practices can be instituted through the process.

The weakness of the Waterfall Model is the fact that it does not flex well to changes in requirements. The deeper the project advances through the development phase, the more costly in time and resources it is to revert back and adopt the new changes. Secondly, a commercial weakness in the process is that a working model of the system is not visible until late in the process, prohibiting prototype distribution of the system to potential clients. Thirdly, real projects seldom flow sequentially. Despite the best efforts of any development team to collect and document requirements, new requirements will surface, old requirements will need to be redefined and refined to be better understood by the development team, and technology will mandate changes in the platform. The Waterfall Model does not lend itself well to development in the Rapid Application Development Environment.[95]

---

[94] James A. DeBardelaben, Lecture Notes of *"Cost Modeling for Embedded Digital Systems Design Module 57"*, Pennsylvania State University, Pittsburgh, Pennsylvania, 15 Sept 1998.

Many contract developers favor the Waterfall Model in that they are able to gather the requirements at the start of the process, develop the system, test it, and if it satisfies the requirements as stated at the beginning, their contract is complete. If, by the time the development process reaches the client, it meets the requirements stated at the beginning of the process, then the contractor has succeeded in his obligation. Such a model does not permit flexible requirements, but reduces the burden on the contractor. DOD-STD-267A/498, once considered the standard guiding software development for the Department of Defense, recognized nine distinct stages of development, almost identical to the Waterfall Model, with the exception of the addition of a precursor "System Concept Analysis Phase" and a post development "Maintenance Phase".[96] This document has since been superceded by numerous other DoD standards.

4. Test Development

The next progression from the Waterfall Model is referred to as the Test Development Model in which the Test Plan is developed in parallel with each phase of development, as depicted in Figure C.4. At each phase of developed, the team intimately involved with that model would composes the Test Plan for the final phase of System Testing. While the Test Development Model does not supercede the need for a balanced Test Plan, it does strengthen the Test Plan by further refining it through the development.

---

95 Laurance Leef, Dr., *UN-10006-1999-11-20, "Rapid Prototyping or the Waterfall Model: Which Path for Legal XML"*, http://www.legalxml.org/DocumentRepository/UnofficialNote/Clear/UN-10006_1999_11_20.html, Legal XML, 20 Nov 1999.

96 DOD-STD-2167A, *"Defense System Software Development"*, 04 Jun 1985.

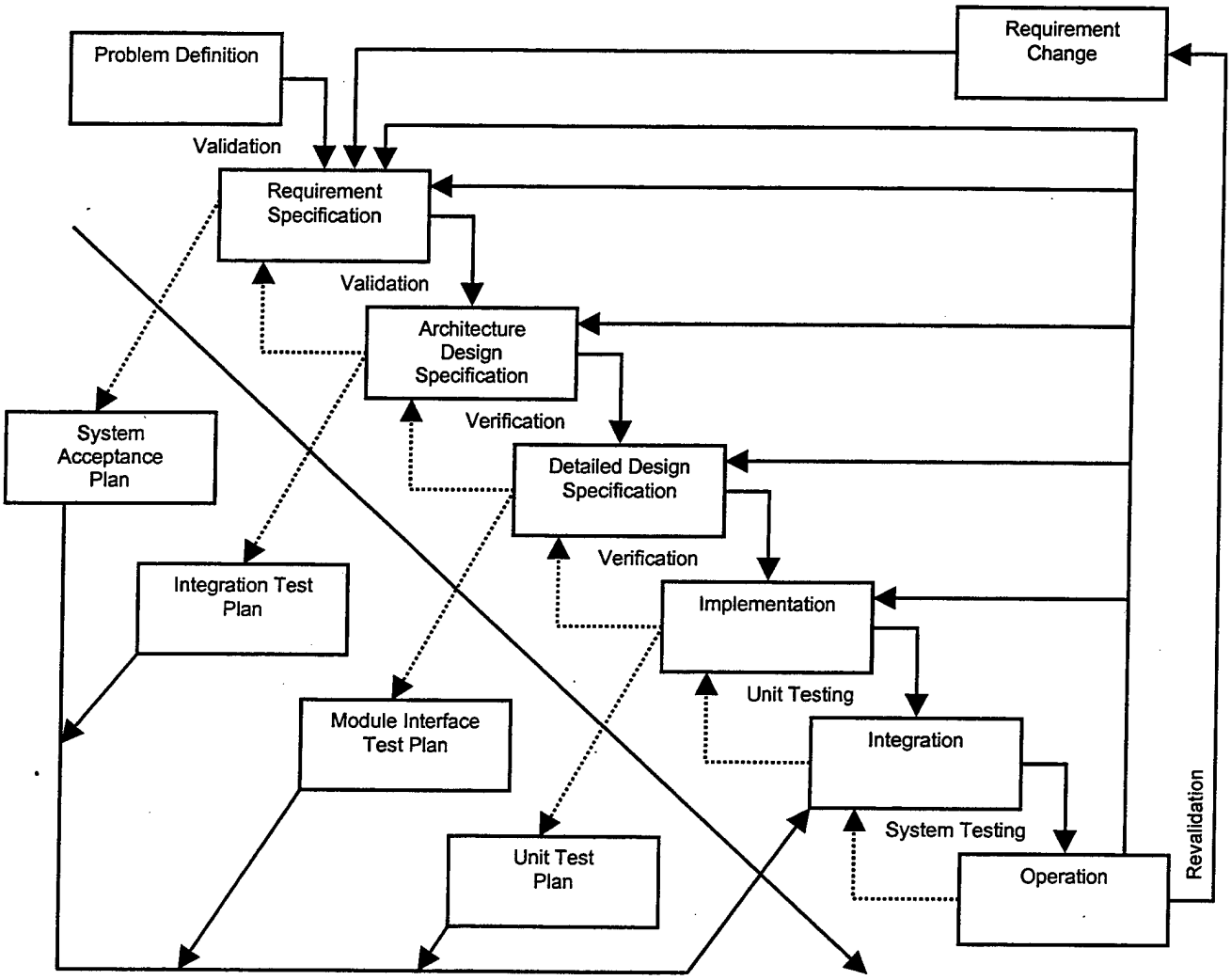


Figure C.4 Test Development Diagram[97]

A strong Test Plan is critical in the evaluation phase of a completed product. Simply determining a Boolean result from a specific requirement does not accurately verify if a requirement has been met. Many requirements have logic driven results that can generate a wide number of responses, interactions, and outcomes dependent on an array of inputs and impulses. If an initial Test Plan is generated in the start of development during the requirements search and specification stage, and then further

refined through each stage of development, it could be ensured that a more accurate test model be authored. Once authored, this Test Plan could be held in reserve until the integration stage for final system testing. Such a concept ensures that methodologies and mentalities used at the onset of development are used to evaluate a final product. Such a concept would serve as a viable tool in the SHARP development.

5. Exploratory Programming

A kin to the simplistic Build and Fix Approach is the Exploratory Programming Technique that incorporates a repetitive building phase after each test of the operation. This approach serves as the traditional baseline for Rapid Application Development (RAD) by incorporating cyclic development. In contrast to the Build and Fix Approach that relies on feedback from the customer for changes to the requirements, the Exploratory Programming method has a Use Phase embedded in its protocol, as depicted by Figure C.5. This approach does not require a detailed requirement specification, but does require direct interaction with the client to progress through each iteration of the development.

---

[97] David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

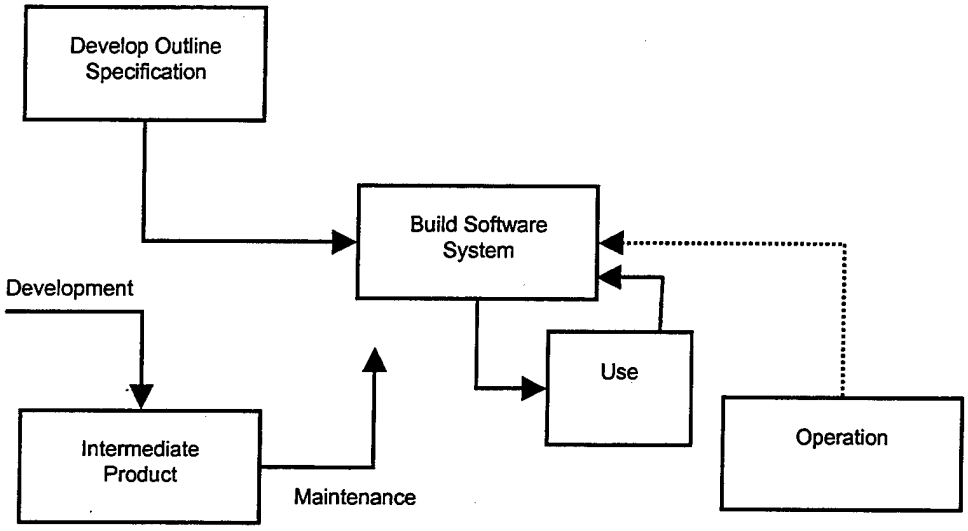


Figure C.5 Exploratory Programming Diagram[98]

For products of small scale or where the subject matter experts are a part of the development team, the Exploratory Programming method increases programmer productivity by giving the development team immediate feedback to all design changes. The "pause-free" interaction allows the developer to concentrate on the task at hand to meet the client's current criticism and requirement. This interaction can also create a great distraction to some developers due to the frequent interruptions and reinterpretation of requirements.[99]

6. Prototyping Model

The Prototyping Model is an extension of the Waterfall Model with the addition of an initial Prototype Phase prior to the Requirements Specification Phase. For the purpose of the Prototype Phase, developers may elect to utilize any of the applicable RAD Models; Exploratory Programming, Build and Fix, Spiral Model (defined later in

---

98 David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

this chapter), or other RAD Models. The Prototype Phases would be used to define or demonstrate the proposed requirements for workability, the proposed concepts for development and applied technology to the system, as well as the applicability of the overall system. Pointing out errors, flaws, shortcomings, or misdirection in requirements, their interpretations, or development, in the earliest possible stage benefits the overall efficiency of a project by limiting the amount of rework. Such a Prototype Phase can decrease the overall workload of the product by reducing the number of requirements that need to be reviewed and revised after system design has commenced. Removal of excessive, unwanted, or unnecessary modules from the production can be accomplished before time is spent on its development, thereby increasing the Risk Management Factors by highlighting potential shortcomings in the overall design and methodology of the system.

7. Incremental Model

The Incremental Model serves as the precursor to the Spiral Model, built upon the Experimental Model, in which the developer would build the first increment or module of the system, submit it for review by the client, then develop the next increment successively until project completion, as depicted by Figure C.6.

---

[99] Urs Holzle, Dissertation of *"Adaptive Optimization for SELF: Reconciling High Performance with Exploratory Programming"*, Stanford University, Palo Alto, CA, Aug 1994.

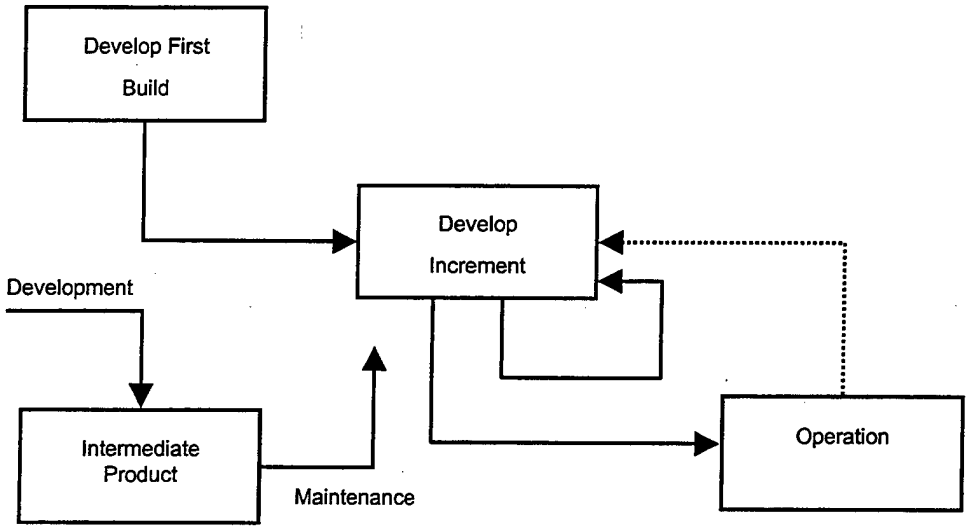


Figure C.6 Incremental Model Diagram[100]

Development of the Increment Phase can be accomplished using any of the Stagewise Development Models, including the Waterfall Model.

8. Spiral Model

For the last two decades, Barry Boehm has been recognized as a pioneer in Software Engineering for his models and techniques of software development and risk management. One of his most widely adopted accomplishments is that of the Spiral Model of software development, which he formalized in 1988.[101] The Spiral Model is a progressive cyclic version of a stagewise development model, which begins each cycle of the spiral by performing the next level of elaboration of the prospective system's requirements, as depicted by Figure C.7

---

100 David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

101 Barry Boehm, *"A Spiral Model of Software Development and Enhancement"*, Computer, May 1998, p. 61-72.

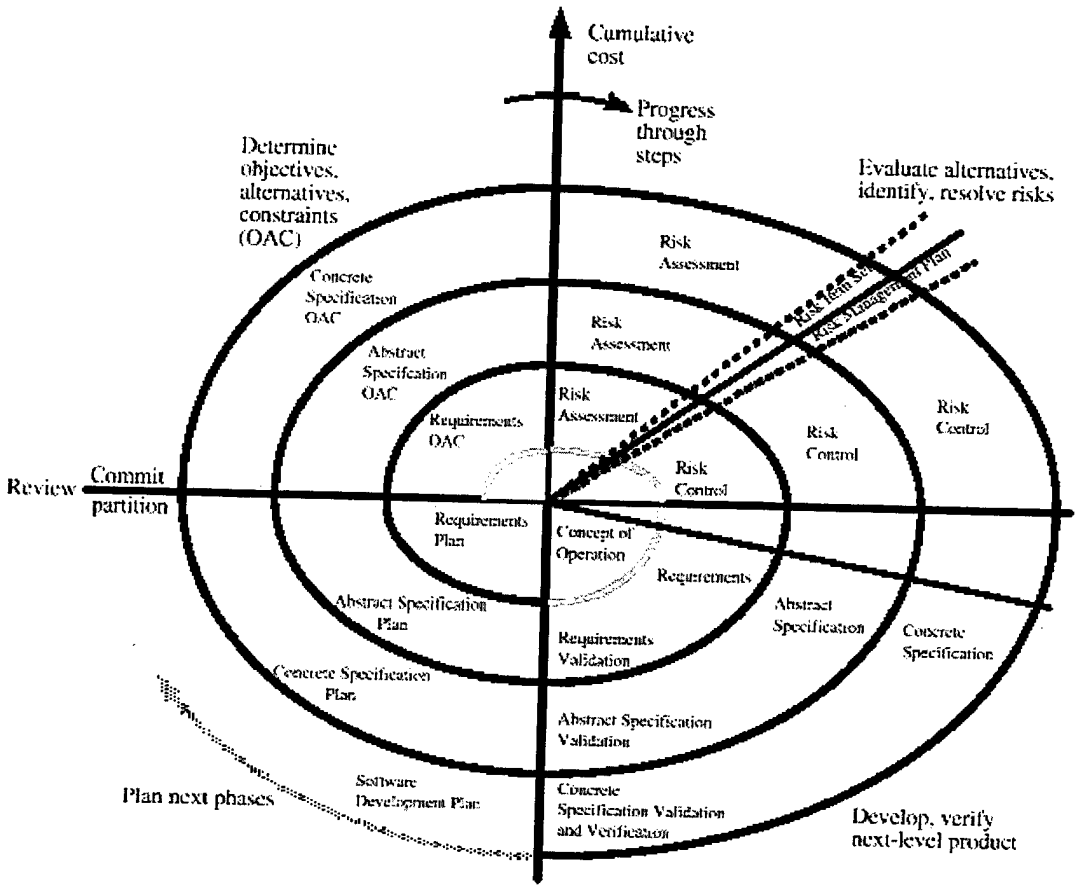


Figure C.7 Spiral Model Diagram[102]

The Spiral Model is built on the fundamental concept of a repetitive or cyclic development platform, with increasing system complexity or refinement with each revisit to the phases of the cycle. The Spiral Model consists of four phases;[103]

Planning – Determine objectives, gather requirements and alternatives, and establish constraints.

Risk Analysis – Evaluate alternatives, identify and resolve risks.

Engineering – Develop prototypes or modules of the product.

---

102 David F. Redmiles, *"ICS 121 Software Tools and Methods, Lifecycle Models, Class Notes"*, http://www.ics.uci.edu/~redmiles/ics121-FQ99, University of California, Irvine, 1999.

103 Roger S. Pressman, *"Software Engineering: A Practitioner's Approach"*, McGraw-Hill, 1992.

Evaluation – Test and evaluate product based on Test Plan and requirement specifications.

Initially, from the center of the spiral, the requirements are researched and composed, then risk analysis is used to determine the degree of difficulty, uncertainty, and potential faults in the requirements. The development team would then engineer a prototype or working model of the system in a simplistic form, then finally have it evaluated by the testing team or even the client for completeness against the requirement specifications. If, at this point in the cycle, the model does not meet the expectations of the development team or the client, then requirements can be revised and refined. Potentially new requirements can be inspired from the prototype developed. At the end of this phase, the decision needs to be made to accept the product as is, or that the requirements justify to continue through another cycle of evolution. Once the requirements have been revised from the current cycle and the results warrant continuation in development, the process starts again to the next level, hopefully resulting in its completion with a more concrete product.

One the strength of the Spiral Model is its flexibility in requirements and evolutionary design, in that design can be done in increments and retailored as system the requirements change. Due to the cyclic evaluation of the product, client and developer feedback can be collected at established points and periods, permitting an opportunity for positive measures or correction to be taken to rectify errors. Secondly, the Spiral Model established a risk management phase in the cycle to identify and manage risk early in the software development process[104]. If the risk can not be mitigated, a decision can be made

[104] R. Charette, *"Large-Scale Project Management is Risk Management"*, IEEE Software, 1996.

early in the process to reformat the requirement or even terminate the project before great resources are expended on the development. Thirdly, the Spiral Model encourages incremental design, giving the client a greater opportunity to receive a working prototype of the system early in the development process, permitting him to make previsions to his environment to accept the final system.

One of the weaknesses of the Spiral Model is that, without proper management, it stands the potential to lacks milestones, as development cycles began to merge together and depart series from its corresponding cycle.[105] It is essential that each stage moves incrementally through the process and then from one cycle up to the next. Without proper management, a spiral development can become trapped in one level of the cycle and not progress up to the next level. Regardless of what software design model is selected for the development process, failure to provide sufficient management oversight will result in a loss in development inefficiencies.

9. Legacy and Reuse Software Life Cycle

Due to the growing pressure to increase development productivity and efficiency, many development teams are taking a serious look at the concept of software reuse and redevelopment of legacy systems over the costly venture of building new systems from scratch. Some software cost evaluation models have predicted savings of up to 30% when implementing software source-code reuse management.[106] In an attempt to properly manage such an undertaking, the Legacy and Reuse Software Life Cycle Approach was drafted to model the two-way flow of development, as the requirements

---

105 Barry Boehm, Software Risk Management, IEEE Computer Society Press, 1989.

drive the development and previously developed software drives the requirements, as depicted in Figure C.8.

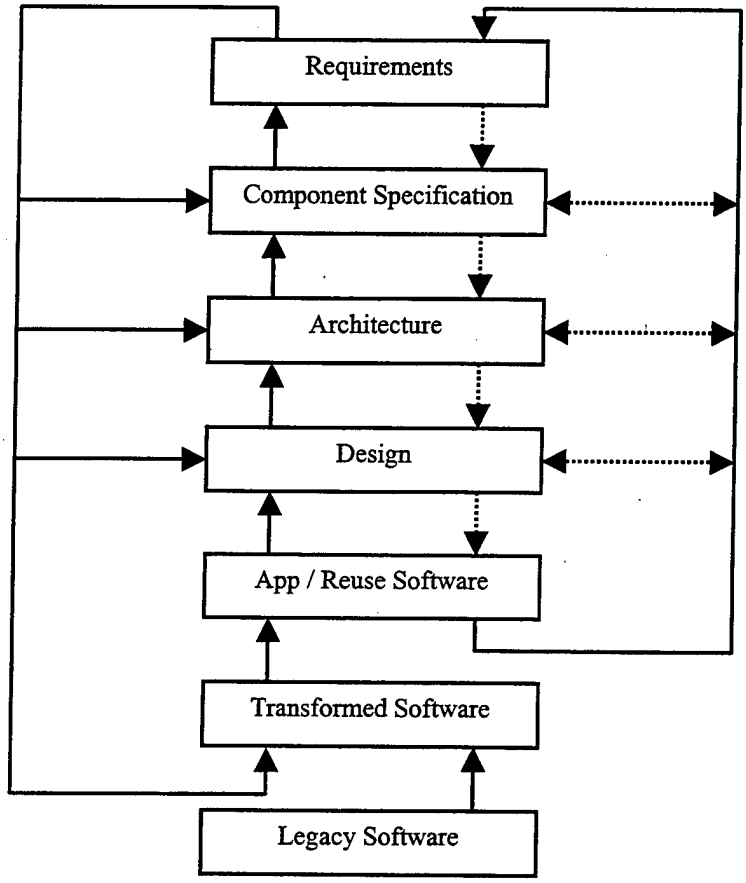


Figure C.8 Legacy and Reuse Software Life Cycle[107]

The primary success of any software reuse plan requires a searchable software repository with sufficient depth to contain all potential variants of applicable code. Secondly, the actions of the software development team requires constant attention to software reuse including standardize code semantics, reverse engineering of

---

106 Frank McGrath, *"16 Best Practices: Management and Technical Practices with High ROI in Development and Sustainment of Large Scale Software Intensive Systems"*, Software Program Managers Network, 1996.

107 James A. DeBardelaben, Lecture Notes of *"Cost Modeling for Embedded Digital Systems Design Module 57"*, Pennsylvania State University, Pittsburgh, Pennsylvania, 15 Sept 1998.

development, and an open architecture of engineering, all encouraged by management actions and applicable support tools. The Reuse Lifecycle relies on a traditional stagewise development technique with a reverse flow of development in which legacy software influences the transition software, which in turn influences the development of the reuse software chosen, backwards to the design and eventually driving the requirements. In a parallel effort, the requirements are driving the design, and then the reuse software selected for the development. DoD Military Standard 498, "Software Development and Documentation," dated December of 1994 specifically denoted the Department of Defense's efforts for software reuse.[108]

[108] DOD Military Standard 498, *"Software Development and Documentation"*, Dec 1994.

APPENDIX D - DOD SOFTWARE ENGINEERING REQUIREMENTS

While the following reviews of DoD Software Engineering Requirements may serve as a broad representation of the most relevant requirement instructions, they in no way represent all of the potential models that can or would be employed by software development managers.

1. DOD-STD-2167A

In an attempt to support the development, design, and acquisition of mission-critical software systems, the Department of Defense established DOD-STD-2167A – Defense System Software Development, approved February 29th, 1988.[109] 2167A formalized a stagewise development process including phases for:

- System Requirements Analysis and Design
- Software Requirements Analysis
- Preliminary Design
- Detailed Design
- Coding and CSU Testing
- CSC Integration and Testing
- CSCI testing
- System Integration and Testing
- Maintenance

Standard 2167A was based on the rigid flow of the Waterfall Model for its developmental process. Previously, software development was accomplished with a "footloose and fancy-free" attitude, with little oversight or guidance to regulation and management. With the introduction of 2167A, system design teams were required to adhere to a structured evolution flow of development, with phases for requirement determination and system testing. The 2167A document was designed to follow the design and development process of a product through its entire lifecycle of the system.

Shortly after the document was released and into the early 1990's, DOD-STD-2167A became recognized throughout the world as a de-facto software development standard. The fact that a compelling number of national and international groups were willing to follow one standard of development served only to strengthen the authority of the DoD Standard. Despite the growing international support, DOD-STD-2167A was destined to its own demise due to its inflexibility related to object-orientated design, excessive documentation, no guidance on management indicators, and the need to incorporate new development techniques such as reuse and reengineering.[110] DOD-STD-2167A was cancelled on December 5th, 1994[111] when it was superseded by MIL-STD-498.

2. DOD-STD-7935A

In parallel with the release of DOD-STD-2167A, the Department of Defense composed DOD-STD-7935A – the Department of Defense Automated Information System Documentation Standards, published for release on October 31st, 1988.[112] DOD-STD-7935A was authored as a standard to provided guidelines for the development and revision of the documentation for Automated Information System (AIS) or applicable software, specified in eleven types of documents that may be produced throughout the life cycle of development:

- DS Database Specification

---

109 DOD-STD-2167A, *"Defense System Software Development"*, 29 Feb 1988.

110 George A. Newberry, Mag. USAF, *"Changes from DOD-STD-2167A to MIL-STD-498"*, http://www.stsc.hill.af.mil/crosstalk/1995/apr/Changes.asp, SAF/AQKS, USN, Apr 1995.

111 DOD Military Standard 498, *"Software Development and Documentation"*, 08 Nov 1994.

112 DOD-STD-7935A, *"Department of Defense Automated Information System Documentation Standards"*, 31 Oct 1988.

- EM End User Manual
- FD Functional Description
- IP Implementation Procedures
- MM Maintenance Manual
- OM Computer Operation Manual
- PT Test Plan
- RT Test Analysis Report
- SS System/Subsystem Specification
- UM Users Manual
- US Software Unit Specification

As a guidance standard, 7935A provided an invaluable resource to management by establishing concrete templates for development documentation. Project managers were able to establish development goals and benchmarks through the process of formal document submissions, and with the progressive nature and order of the documents, establish protocols for testing and evaluating the advancement of the system. While the success of 7935A was found in its standardization features, its demise was also found in its inflexibility to changing technological requirements and proprietary document formats. As with its sister, DOD-STD-2167A, DOD-STD-7935A was cancelled on December 5th, 1994 when it was superseded by MIL-STD-498.[113] While the documentation standards of this instruction may have been superseded by the more flexible structure of MIL-STD-498, the context and content of these documents and their formats still serve as an invaluable resource template for development engineers to design and tailor their own papers.

[113] DOD Military Standard 498, *"Software Development and Documentation"*, 08 Nov 1994.

3. MIL-STD-498

MIL-STD-498, also referred to as Military Standard 498 – Software Development and Documentation, was approved November 8th, 1994,[114] with four primary objectives:

- Merge DOD-STD-2167A, used for weapon system, with DOD-STD-7935A, used for automated information systems, creating a single software development standard for DoD.
- Resolve issues raised in the use of these standards.
- Ensure compatibility with current DoD directives, instructions, and other standards.
- Provide a basis for United States implementation of ISO/IEC 12207, Software Life Cycle Process.[115]

MIL-STD-498 was composed by a team of DoD Software Engineering Experts including the Office of the Undersecretary of Defense (Acquisition), the Director of Defense Research Engineering, and the Joint Logistics Commanders on a board chaired by the Space and Naval Warfare Command (SPAWAR) called the Harmonization Working Group (HWG). Representatives of all four armed forces, DoD agencies, other federal agencies, members of industry, and various foreign defense allies participated in the working group.

MIL-STD-498 was developed as a stopgap solution to the lack of a military standard for software development and corresponding commercial standard. The document was issued for an interim period of two years, before it would be reviewed for an applicable commercial standard.

---

114 DOD Military Standard 498, *"Software Development and Documentation"*, 08 Nov 1994.

115 Jane Radatz, Myran Olson, Stuart Campbell – Logicon, MIL-STD-498, Logicon, San Diego, CA, 1994.

One of the key pieces of MIL-STD-498 was the model of developing systems in multiple stages or "builds". Each of the builds would incorporate a specified subset of the planned capabilities of the software, closely resembling the Spiral Model. A second piece of the document called for disestablishing the practice of formal reviews, due to great number of man-hours lost preparing for the review, and establishing a more frequent informal review process. A third part of the document called for a decrease in the emphasis for documentation and increasing the capabilities of computer-aided software engineering (CASE) tools to prototype the development of the system. The document is further broken down into 22 Data Item Descriptions (DIDs) including:

- Software Development Plan (SDP)
- Software Test Plan (STP)
- Software Installation Plan (SIP)
- Software Requirements Specification (SRS)
- Software Design Descriptions (SDD)
- Software Test Descriptions (STD)
- Software User Manual (SUM)

MIL-STD-498 was cancelled May 27th, 1998 with the DoD acceptance of IEEE/EIA 12207.[116]

4. IEEE/EIA or ISO/IEC 12207

In 1995, the Electronic Industries Association (EIA) and the Institute of Electrical and Electronic Engineers (IEEE) jointly established a commercial software development standard to serve as a precursor to military software development, referred to as Version 12207. This new version was designed to serve as a strategic solution to:

- Represent the best commercial practices.

---

116 James W. Moore, Perry R. DeWeese, Dennis Rilling, *"U. S. Software Lifecycle Process Standard"*, http://stsc.hill.af.mil/crosstalk/1997/jul/lifecycle.asp, Space and Naval Warfare System Center, Jul 1997.

- Be suitable for application to the complex requirements of Defense acquisition.
- Be compatible with those of the emerging global marketplace for software.[117]

The U. S. Department of Defense formally adopted 12207 on the 9th of December 1997 to serve as a framework for software development. Standard 12207 included six important advances over existing software lifecycle standards:

- Coverage of the entire lifecycle constraints and development process.
- Flexible approach to recording process and product data to be handled by computer–aided software-engineering tools.
- Incorporates specific references to existing U. S. standards.
- Provides a set of process and data objectives that guide adaptation of the standard for unusual situations.
- Compatible with the ISO 9000 approach to quality systems, quality management, and quality assurance.
- Fully compliant with the international version of the standard, permitting U. S. companies to develop a single set of enterprise processes applicable to both global and domestic business.[118]

The concept of a flexible approach to documentation can be one of the more contentious points of the standard. Many of the previous standards specifically detailed format and content of documentation, and the management principles of the documentation process. Many organizations have built a large support complex around

---

117 James W. Moore, Perry R. DeWeese, Dennis Rilling, *"U. S. Software Lifecycle Process Standard"*, http://stsc.hill.af.mil/crosstalk/1997/jul/lifecycle.asp, Space and Naval Warfare System Center, Jul 1997.

118 James W. Moore, Perry R. DeWeese, Dennis Rilling, *"U. S. Software Lifecycle Process Standard"*, http://stsc.hill.af.mil/crosstalk/1997/jul/lifecycle.asp, Space and Naval Warfare System Center, Jul 1997.

the concept of documentation management, goal management, and the formal evaluation and review process. 12207 returned flexibility and autonomy back to the development team to design and manage a product development using a principle of guidance and recommendation.

One of the unforeseen results of 12207 was the fact that, without proper oversight and management, development teams in large commands could face standardization problems, in that one unit may require strong documentation and evaluations, while another unit may permit a lesser level of documentation and evaluation management oversight. This difference can hinder the concept of reusable systems, as outlined in the Legacy and Reuse Software Life Cycle Approach.

5. DII COE

The DII COE - Defense Information Infrastructure Common Operating Environment was established under the direction of the Secretary of Defense on August 22nd, 1996,[119] to create a unified structure for the development of software hardware and software in the joint environment. DII COE is a collection of reusable software components, a software infrastructure for supporting mission-area applications, guidelines, standards, and specifications, and an architecture and approach for program management.[120] DII COE is not a single document, but rather a collection of fluid documents that cover all of the facets of system development for products under submission to the Defense Information System Agency (DISA). The cornerstone of DII COE is its eight levels or degrees of interoperability compliance for development, as:

---

[119] Office of the Secretary of Defense, *"Subj: Implementation of the DoD Joint Technical Architecture"*, 22 Aug 1996.

• Level 1	Standard Compliance	14 Requirements
• Level 2	Network Compliance	15 Requirements
• Level 3	Platform Compliance	17 Requirements
• Level 4	Bootstrap Compliance	16 Requirements
• Level 5	Minimal DII Compliance	117 Requirements
• Level 6	Intermediate DII Compliance	63 Requirements
• Level 7	Interoperable Compliance	35 Requirements
• Level 8	Full DII Compliance	22 Requirements[121]

The DII COE is managed in parallel with the CM-165-60-03 - Configuration Management Software and Documentation Delivery Requirements, Version 3.0, released March 8th 1998.[122]

In a phone interview with the Chief Engineer of DII COE, Mr. Kenneth Wheeler, I was informed that DII COE was designed to serve as an architecture or approach for developing interoperable systems and establishing a baseline of compliance. While the DII COE standard is not yet widely accepted in industry, Mr. Wheeler is attempting to change that by imposing his influence on defense contractors. All defense contractors who intend to do business with DISA units will be required to be DII COE compliant with all product submissions. His intention is that defense contractors will find it more efficient to adopt the DII COE concept from their military divisions through to their entire company, and then influence those whom they do business with to also adopt the standard.[123]

---

120 Assistant Secretary of Defense, *"Subj: Implementation of Defense Information Infrastructure Common Operating Environment Compliance"*, 23 May 1997.

121 DII COE Integration and Runtime Specification Version 4.0, Appendix B: *Compliance Checklist*, 25 Oct 1999.

122 CM-165-60-03, *"Configuration Management Software And Documentation Delivery Requirements"*, Version 3.0, 08 Mar 1998.

123 Phone Con, 13 Jul 2000, Mr. Kenneth Wheeler, DII COE Chief Engineer, DII COE Agency Joint Office, Falls Church VA., 703-681-2304.

As per the DII COE instruction[124] and the promulgating letter from the Office of the Secretary of Defense,[125] the requirement to follow the DII COE standard is mandated only to C4I systems and the interfaces of other key assets (e.g., weapon system, sensors, office automation systems, etc.) with C4I systems. While the decision for other branches and agencies to follow the DII COE compliance schedule is voluntary, there by association, compliance by contractors is also voluntary. Without compelling authority outside of the C4I environment, the DII COE instruction will remain a limited or even "stovepipe" [def] standard.

Mr. Wheeler stated in out interview that the DII COE mandate is "driven by technology, tempered by the reality of the limitation of the existing systems." He mentioned that any standard needs to be bounded by production reality. While technology would dictate the installation of fiber-optic networks across all military installations and vessels, to increase data transfer rates and reliance, reality imposes a limitation to what the military could impose. The expense of installing a base wide network would be too cost prohibitive. The physical limitations of retrofitting and installing a fiber-optic network on all naval vessels would be physically prohibitive. Mr. Wheeler intends to establish a standards based development system that promotes integration and development cooperation through his organization's software tools, run time specifications, and management principles. His and the DII COE success will be a realized through the "commertization"[def] of the software environment. The DII COE

---

124 DII COE Integration and Runtime Specification Version 4.0, Appendix B: *Compliance Checklist*, 25 Oct 1999.

125 Office of the Secretary of Defense, *"Subj: Implementation of the DoD Joint Technical Architecture"*, 22 Aug 1996.

standard is not recognized or accredited by IEEE or any other internationally recognized agency, nor does Mr. Wheeler have any intention to seek accreditation from any agency. When asked why Mr. Wheeler had not made any attempt to seek such accreditation or recognition, he stated that "the return on investment is not there."[126]

6. DoD Regulation 5000.2-R

DoD Reg 5000.2-R, Change 4, the Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs, was signed for release on May 11th, 1999 by both the Under Secretary of Defense (Acquisition and Technology); Director, Operational Test and Evaluation; and the Assistant Secretary of Defense (Command, Control, Communications, and Intelligence). The 5000.2 establishes a simplified and flexible management framework for translating mission needs into stable, affordable, and well-managed MDAPs and MAIS Acquisition Programs, as well as sets forth mandatory procedures for MDAPs and MAISs.[127]

The scope of the DoD Reg 5000.2 applies to all major defense acquisition programs or other programs as mandated by Congress, with the exception of highly sensitive classified programs, cryptologic, and intelligence programs which shall follow the guidance for other programs, or other programs as waived. A major program is defined as:

- Not a highly sensitive classified program.

---

126 Phone Con, 13 Jul 2000, Mr. Kenneth Wheeler, DII COE Chief Engineer, DII COE Agency Joint Office, Falls Church, VA., 703-681-2304.

127 DOD Reg 5000.2-R, Change 4, *"The Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition"*, 11 May 1999.

- Estimated by the USD(A&T) to require an eventual total expenditure for research, development, test and evaluation of more than 355 million in fiscal year (FY) 1996 constant dollars.

- Total procurement cost of more than 2.135 billion in FY 1996 constant dollars.

The 5000.2 is broken up into six phases or parts:

1. The Acquisition Management Process – Defining and establishing the system management process, the acquisition phases and accomplishments, and milestone establishments.

2. Program Definition – Product support, requirements, alternatives, affordability, and supportability.

3. Program Structure – Program goals, acquisition strategy, test and evaluation, and life cycle planning.

4. Program Design – Integrated process and project development, and system engineering.

5. Program Assessments and Decision Reviews – Establishment of the Defense Acquisition Board, Joint Requirements Oversight Council Review Procedures, Cost Analysis Improvement Group Procedures, and various other boards and councils.

6. Periodic Reporting - Cost, Schedule, and Performance Program Reports, Test and Evaluation Reports, and Contract Management Reports.

Under the direction of the 5000.2, DoD major software acquisitions managers have new direction for the purchase and maintenance of commercially and government developed products. Due to the high dollar value definition of major accusation projects, many development projects actually fall outside of the scope of the requirement. In the case of some potentially major accusation projects, some contractors will actually subdivide their projects in an attempt to avoid the requirements of the 5000.2. Due to the abilities to avoid the DoD Regulation 5000.2, many development teams have opted to use other accredited and non-accredited standards.

DoD Regulation 5000.2-R, Change 4, remains in effect, under periodic review.

7. DoD Directive 5200.40

DoD Directive 5200.40, "DoD Information Technology Security Certification and Accreditation Process" or DITSCAP, was signed for approval on December 30th, 1997, by then active Secretary of Defense for C3I to implement policies, assign responsibilities, and prescribe procedures for certification and accreditation of information technology. The scope of DITSCAP covers automation, information systems, networks, and sites in the Department of Defense.[128]

DITSCAP is designed to standardizes the certification and accreditation (C&A) process to ensure information systems are properly guarded from unwanted intrusion, as well as establish a life-cycle management approach to the C&A and reaccredidation process of DoD IT. The authority of DITSCAP is extremely broad as stated in the instruction, that it:

> "Shall apply to the acquisition, operation, and sustainment of any DoD system that collects, stores, transmits, or processes unclassified or classified information. It applies to any IT or information system life cycle, including the development of new IT systems, the incorporation of IT systems in to an infrastructure, the incorporation of IT systems outside the infrastructure, the development of prototype IT systems, the reconfiguration or upgrading of existing systems, and legacy systems."

This sweeping authority extends to all military departments, defense agencies, DoD field activities, their contractors, and agents, in coordination with the National Security Agency (NSA). The DITSCAP process shall consist of four phases of development, including:

---

128 DoD Directive 5200.40, *"DoD Information Technology Security Certification and Accreditation Process (DITSCAP)"*, 30 Dec 1997.

- Phase 1 — Definition – The documentation of the system mission, environment, and architecture, defining of the threat, levels of effort, and certification authority.
- Phase 2 — Verification – The verification of compliance with previously agreed security requirements.
- Phase 3 — Validation – The evaluation of a fully integrated system to validate system operation in a specified computing environment with an acceptable level of residual risk.
- Phase 4 — Post Accreditation – Actives to monitor system management and operation to ensure an acceptable level of residual risk is preserved.

Through the integration of the four phases of DITSCAP, it is intended that this standard will protect the DII by presenting an infrastructure-centric approach for certification and accreditation. DoD Directive 5200.40 remains in effect under annual review.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Berzins, V., *Unpublished Class Notes for CS4500*, Naval Postgraduate School, Monterey, CA.

2. Clark, Edmund M. and Wing, Jeannette M., *"Formal Methods: State of the Art and Future Directions,"* Carnegie Mellon University, a part of the NPS CS 4500 Class Notes.

3. Dampier, David A., *"A Model for Merging Different Versions of a PSDL Program,"* Naval Postgraduate School, Monterey, CA., 1990.

4. Ibrahim, Osman Mohamed, *"A Model and Decision Support Mechanism for Software Requirements Engineering,"* Doctoral Dissertation, Naval Postgraduate School, Monterey, CA, 1996.

5. *"Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development"*, 1994 Monterey Workshop Notes, Naval Postgraduate School, Monterey, CA, 1994.

6. *"Increasing the Practical Impact of Formal Methods for Computer-Aided Software Development"*, 1995 Monterey Workshop Notes, Naval Postgraduate School, Monterey, CA, 1995.

7. Maiden, Neil A. and Ncube, Cornelius, *"Acquiring COTS Software Selection Requirements."* IEEE Software, March / April 1998.

8. Osmundson, John, *Unpublished Class Notes for IS4300,* Naval Postgraduate School, Monterey, CA, 1999.

9. Reed, John S., *"Critical Success Factors in Software Projects,"* IEEE Software, May / June 1999.

10. Shing, Man-Tak, *Unpublished Class Notes forCS3460,* Naval Postgraduate School, Monterey, CA, 1999.

11. Shing, Man-Tak, *Unpublished Class Notes forCS4596,* Naval Postgraduate School, Monterey, CA, 1999.

12. Troyka, Lynn Quitman, *"Handbook for Writers"*, Simon and Schuster Publishing, Englewood Cliffs, N. J., 1998.

13. White, L. *"The Development of a Rapid Prototyping Environment"*, Master's Thesis, Naval Postgraduate School, Monterey, CA, 1989.

THIS PAGE INTENTIONALLY LEFT BLANK

ABBREVIATIONS

AAW	Anti-Air Warfare
ACTC	Aircrew Training Continuum or Air Combat Training Continuum
AIRLANT	Naval Air Force, United Stated Atlantic Fleet
AIRPAC	Naval Air Force, United Stated Pacific Fleet
AIS	Automated Information System
AMW	Amphibious Warfare
AOA	Analysis of Alternatives
AQKS	Air Force Acquisition – Office "Kilo" – Deputy Assistant of the Secretary for Logistics - Software
ASU	Anti-Surface Warfare
ASW	Anti-Submarine Warfare
ATRRIS	Aviation Training, Readiness, and Requirements Information System
C&A	Certification and Accreditation
C2W	Command and Control Warfare
C3I	Command, Control, Communications, and Intelligence
C4I	Command, Control, Communications, Computers, & Intelligence
CAG	Carrier Air Group
CANDE	Computer Aided NAVFLIRS Data Entry
CASE	Computer-Aided Software Engineering
CCC	Command, Control, and Communications
CD-W	Compact Disk Writer
CDR	Commander
CM	Configuration Management
CNAL	Commander, Naval Air Force, United Stated Atlantic Fleet
CNAP	Commander, Naval Air Force, United Stated Pacific Fleet
CNO	Chief of Naval Operations
COMNAVAIRLANT	Commander, Naval Air Force, United States Atlantic Fleet

COMNAVAIRLANTINST
Commander, Naval Air Force, United States Atlantic Fleet, Instruction

COMNAVAIRPAC
Commander, Naval Air Force, United States Pacific Fleet

COMNAVAIRPACINST
Commander, Naval Air Force, United States Pacific Fleet, Instruction

COTS	Commercial off the Shelf
CPF	Commander Pacific Fleet
DID	Data Item Descriptions
DII COE	Defense Information Infrastructure Common Operating Environment
DISA	Defense Information System Agency
DITSCAP	DOD Information Technology Security Certification and Accreditation Process
DOD, DoD	Department of Defense
EIA	Electronic Industries Association
ERP	Enterprise Resource Planning
FAC	Forward Air Control
FHP	Flight-Hours Program
FPO	Fleet Post Office
FSO	Fleet Support
GPS	Global Position Satellite (System)
HFQMB	Human Factors Quality Management Board
HWG	Harmonization Working Group
IAF	Israeli Air Force
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INT	Intelligence
IPT	Integrated Process Team
ISO	International Organization for Standardization
IT	Information Technology

JMETL	Joint Mission Essential Task List
LAN	Local Area Network
LCDR	Lieutenant Commander
LOG	Logistics
LT	Lieutenant
MAIS	Major Automated Information System
MAJ	Major
MAW	Marine Air Wing
Mb	Megabyte
MCO	Marine Corps Order
MDAP	Mandatory Procedures for Major Defense Acquisition Programs
MDB	Microsoft Database
MIW	Mine Interdiction Warfare
MOB	Mobility
MOS	Missions of State
MS	Microsoft
MSDE	Microsoft Data Engine
MSVB	Microsoft Visual Basic
MTF	Message Test Format
NALCOMIS	Naval Aviation Logistics Command Management Information System
NAVAIR	Chief of Naval Air Operations
NAVFLIRS	Naval Aircraft Flight Record
NATOPS	Naval Air Training and Operations Procedures Standardization
NCA	National Command Authority
NCO	Non-Combative Operations
NFO	Naval Flight Officer
NPS	Naval Postgraduate School
NSA	National Security Agency

NSW	Naval Strike Warfare
NTTL	Naval Tactical Task List
NWP	Naval Warfare Publication
OMA	Organizational Maintenance Activities
ORM	Operational Risk Management
OPNAVINST	Chief of Naval Operations Instruction
OS	Operating System
P4 + WARTS	People, Planes, Parts, and Petrol, plus Weapons, Adversaries, Ranges, TAD, and Simulators
PC	Personal Computer
PMA	Primary Mission Area
POE	Projected Operational Environment
PQS	Personnel Qualification Standards
RAD	Rapid Application Development
ROC	Required Operational Capability
SAF	Secretary of the Air Force
SARA	Squadron Assistance Risk Assessment
SDD	Software Design Specification
SDP	Software Development Plan
SE	Software Engineering
SHARP	Sierra Hotel Aviation Reporting Program
SIP	Software Installation Plan
SIPRNET	Secret (Secure) Internet Protocol Router Network
SMART-R	Squadron Management, Automated Risk Tolerance and Reporting System
SME	Subject Matter Expert
SOP	Standard Operating Procedures
SORTS	Status of Resources and Training System
SPAWAR	Space and Naval Warfare System Center
SQL	Structured Query Languages

SQOM	Squadron Operations Management System
SRS	Software (System) Requirements Specification
STD	Software test Description
STP	Software Test Plan
STW	Strike Warfare
SUM	Software User Manual
T & R	Training and Readiness
TACAIR	Tactical Air
TAD	Temporary Assigned Duty
TRMS	TYCOM Readiness Management System
TYCOM	Type Commander. Commander Naval Air Force United Stated Pacific Fleet and the Commander Naval Air Force United States Atlantic Fleet
UAV	Unmanned Aerial Vehicle
UJTL	Universal Joint Task List
UNTL	Universal Naval Task List
USD	Under-Secretary of Defense
USAF	United States Air Force
USCG	United States Coast Guard
USMC	United States Marine Corps
USN	United States Navy
VB	Visual Basic
WAT	Weapons and Tactics
WTM	Wing Training Manual
WWW	World Wide Web

THIS PAGE INTENTIONALLY LEFT BLANK

DEFINITIONS

Adversaries (P4+WARTS)	Sufficient number of adversaries or opponent aircraft available to complete squadron training requirements, as required by the type WTM.
Causality Principle	The principle that: cause must always proceed effect; cause always relates effect.
Commertization	Making a product commercially viable[129]
COTS	Commercial off the Shelf - Commercial items customarily used for non-governmental purposes and offered for sale, lease, or license to the general public; An item evolved from such an item that will be available within sufficient time; Items that are standard modifications available in the commercial marketplace or are minor modifications; Any non-developmental item developed exclusively at private expense and competitively sold in substantial quantities to non-federal governments.
Crewstation	The position of responsibility that a crewmember would have on an aircraft; i.e. Pilot, Crew Chief, or Navigator.
Development Field	A visual language term to represent the graphical area of development for the placement of visual objects.
Detachment	Sub-units of a parent command, planning to or actually deployed away from the parent command.
Determinism Principle	The principle that: if one knows the state to an infinite accuracy of a system at one point in tine, one would be able to predict the state of that system with infinite accuracy at any other time, past or future.
Grease Boards	Large glass panes that can be inscribed or painted from behind to make borders, columns, and rows, and then written on from the front to display information. For the purpose of Training and Readiness computations, users would inscribe a matrix of qualifications by crewmember, and then enter the dates in which each of the qualifications would expire.

[129] Phone Con, 13 Jul 2000, Mr. Kenneth Wheeler, DII COE Chief Engineer, DII COE Agency Joint Office, Falls Church VA., 703-681-2304.

Homeguard	The parent unit for detachments.
Legacy System	A computer system or application program which continues to be used because of the cost of replacing or designing it and often despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic, and difficult to modify.[130]
P4 + WARTS	The encompassing concept of all factors that effect readiness, as People, Planes, Parts, and Petrol, plus Weapons, Adversaries, Ranges, TAD, and Simulators
Parts (P4+WARTS)	Access to sufficient inventory of aircraft parts required to maintain planes, as denoted by the ROC POE.
People (P4+WARTS)	The combination of all personnel assigned to squadrons, either as pilots, aircrewmen, maintainers, or other support personnel, in the correct numbers, as denoted by the Personnel Manual.[131]
Petrol (P4+WARTS)	Sufficient budget to purchase aviation fuel, which in turn provides flight-hours to accomplish training, operational, and overhead missions, as determined by the OP-20 Report[132]. Overhead missions are those flights not included in training or operational events.
Planes (P4+WARTS)	The aircraft assigned to a squadron, capable of completing assigned missions, as assigned by the Universal Naval Task List (UNTL)[133], in the correct numbers as denoted by the T & R[134] and Projected Operational Environment (POE) and Required

---

130 Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

131 OPNAVINST 1016.J, *"Manual of Naval Total Force Manpower Policies and Procedures"*, 01 Jun 1998.

132 OP-20 Report, Flying Hour Program Execution Plan, on file, AIRPAC N8 Office, Various.

133 OPNAVINST 3500.38 / MCO 3500.26 / USCG CMDTINST M3500.1 *"Universal Naval Task List (UNTL)"*, 30 Sep 1996.

134 COMNAVAIRLANTINST 3500.63E, COMNAVAIRPACINST 3500.67E – *"Squadron Training and Readiness"*, 24 Mar 2000, Enclosure (23).

Operational Capability (ROC)[135] Instruction, referred to as the ROC POE.

Ranges (P4+WARTS)
: Sufficient numbers and types of ranges required to facilitate squadron training requirements.

Simulators (P4+WARTS)
: Adequate simulator training devices equipped and available to meet the training requirements of the squadron, as noted in the type WTM.

Stovepipe
: An item with a limited funneling line of data, authority, access, requirements, or goal. Spoken of in derogatory terms.

TAD (P4+WARTS)
: Temporary Assignment for Duty. Sufficient funding to provide temporary duty assignments for squadron personnel to accomplish training events away from home guard, as outlined by the parent command's TAD budget forecast.

Type Commander
: TYCOM. The Commander of Naval Air Force United Stated Pacific Fleet and the Commander of Naval Air Force United States Atlantic Fleet

Type Wing
: Unique commanding units that encompass all subcommands or squadrons of a specific type of aircraft, as in an F/a-18 Hornet Wing, an H-60F Sea Hawk Wing, or EA-6B TACAMO Wing.

Weapons (P4+WARTS)
: A sufficient allocation of weapons, armaments, and expenditures to complete squadron training requirements, as outlined in the type Wing Training Manuals (WTM) and the T & R.

Visual Programming Language
: Any programming language that allows the user to specify a program in a two-(or more) dimensional way. Conventional textual languages are not considered two-dimensional since the compiler or interpreter processes then as one-dimensional streams of characters. A VPL allows programming with visual expressions – spatial arrangements of textual and graphical symbols

---

[135] OPNAVINST (C) 3501.2J, *"Projected Operational Environment (POE) and Required Operational Capabilities (ROC) For Aviation Units"*, various.

VPLs may be further classified, according to the type and extent of visual expression used, into icon-based languages, form-based languages and diagram languages. Visual programming environments provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction.

NOTE: Visual Basic, Visual C++ and the entire Microsoft Visual Family are not, despite their names, visual programming languages. They are textual languages which use a graphical GUI builder to make programming interfaces easier.[136]

136 Denis Howe, *"The Free On-Line Dictionary of Computing"*, http://wombat.doc.ic.ac.uk/, Denis Howe, 1993-1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Professor Luqi, Code CS/LQ 4
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr. Dan Boger 1
   Chairman, Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr. Oleg Kiselyov 1
   Naval Postgraduate School
   Monterey, CA 93943

6. Dr. Richard Reihle 1
   Naval Postgraduate School
   Monterey, CA 93943

7. Captain Paul Young, U. S. Navy 1
   Naval Postgraduate School
   Monterey, CA 93943

8. Lieutenant Chris Williamson, U. S. Navy 1
   2502 Whispering Palms Loop
   Chula Vista, CA, 91915-1402

9. Commander Mark Burgunder, U. S. Navy – Retired 1
   4292 Hortensia Street
   San Diego, CA, 92103