A Lean Software Engineering System for the Department of Defense

By

Troy Hamilton

B.S.B.A. International Business and Finance,
The American University, Washington DC (1993)

Submitted to the System Design and Management Program of the
Sloan School of Management and School of Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
IN
ENGINEERING AND MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1999

© 1999 Troy Hamilton
All Rights Reserved

The author grants MIT permission to reproduce and distribute publicly paper and
electronic copies of this thesis document in whole or in part.

Signature of Author.......................................................... Troy Hamilton
System Design and Management Program
January 1999

Certified by................................................................. Daniel Frey
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by.............................................................. Thomas L. Magnanti
Dean of School of Engineering
Institute Professor
Co-Director, System Design and Management Program
Abstract of Thesis

A Lean Software Engineering System for the Department of Defense

by

Troy Hamilton

Submitted to the System Design and Management Program of the Sloan School of Management and School of Engineering on January 15, 1999 in partial fulfillment of the requirements for the degree of Master of Science in Engineering and Management

Quality software engineering is crucial for the Department of Defense. The ability to engineer software that meets cost, schedule, and technical goals is a continuous challenge for both the commercial and government sector. This thesis presents an engineering model based on lean principles. The lean principles provide a foundation for a system that is based on value, communication, teamwork, efficient use of resources, elimination of waste and continuous process improvement. This system is flexible and can be tailored to meet the needs of projects of varying size and complexity. The model is intended to serve as a template for organizations to evolve their software engineering system to meet the needs of their customers.

Thesis Advisor: Assistant Professor Daniel Frey

Title: A Lean Software Engineering System for the Department of Defense
Acknowledgments

The completion of this thesis and associated course work is not complete without recognizing those who made it possible. First, I would like to thank my wife, Tammy J. Hamilton M.D. for her unconditional support and serving as my role model throughout the years. To Joshua Logan Hamilton, my son born in the middle of this academic adventure, for providing continued inspiration on pursuit of my intellectual goals. To Misha Hamilton, truly a man's best friend during the late night marathon sessions and relaxing walks around town. A well deserved thank you to my family for providing a foundation of support that allowed me to focus on MIT work while not having to worry about family obligations.

To the MIT Sloan School of Management and School of Engineering I say thank you for providing an environment and cohort second to no other institution. Thank you to the SDM program and staff for giving me this opportunity and being flexible over the two years. To all my professors and advisors for sharing a wealth of knowledge and exciting me about new frontiers in engineering and management. I would like to express gratitude to my advisor Dan Frey for bringing focus and value into both my course work and thesis. Special indebtedness to Lockheed Martin Corporation for sponsorship and support of the SDM program. Lastly, I would like to acknowledge my classmates, a very distinguished group from which I learned something from each and every one of you this past two years. Thank you.
Table of Contents

ABSTRACT OF THESIS ......................................................... 1

ACKNOWLEDGMENTS .......................................................... 2

TABLE OF CONTENTS .......................................................... 3

TABLE OF FIGURES ............................................................ 6

TABLE OF TABLES ............................................................. 8

LIST OF ACRONYMS ............................................................ 9

OVERVIEW OF THESIS ........................................................ 12

SECTION 1 INTRODUCTION .................................................... 14

SECTION 2 BACKGROUND AND MOTIVATION .................................. 18

  SECTION 2.1 SYSTEMS: WORKING DEFINITIONS ............................ 19
  SECTION 2.2 SYSTEMS ENGINEERING ....................................... 21
  SECTION 2.3 SYSTEMS THINKING ........................................... 21
  SECTION 2.4 SYSTEM DYNAMICS: THE REWORK CYCLE ..................... 25

SECTION 3 LEAN SOFTWARE ENGINEERING SYSTEM OBJECTIVES ............ 27

SECTION 4 LEAN PRINCIPLES ................................................... 29

  SECTION 4.1 LEAN VALUE .................................................... 33
  SECTION 4.2 LEAN TEAMWORK ............................................. 33
  SECTION 4.3 LEAN COMMUNICATION ....................................... 35
  SECTION 4.4 LEAN EFFICIENT USE OF RESOURCES AND ELIMINATION OF WASTE ......................................................... 36
Table of Figures

FIGURE 1-1 THESIS OVERVIEW ................................................................. 13
FIGURE 2-1 SYSTEMS THINKING – PROBLEM EVALUATION EXAMPLE – PROJECT LATE .............. 23
FIGURE 2-2 SYSTEMS THINKING VIEWPOINT ............................................... 24
FIGURE 2-3 THE REWORK CYCLE .............................................................. 26
FIGURE 3-1 THE LEAN PUZZLE ................................................................. 28
FIGURE 6-1 THE VALUE STREAM ............................................................... 42
FIGURE 7-1 WATERFALL MODEL ............................................................... 49
FIGURE 7-2 CODE AND FIX ..................................................................... 50
FIGURE 7-3 SPIRAL MODEL ...................................................................... 52
FIGURE 7-4 EVOLUTIONARY PROTOTYPE MODEL ....................................... 54
FIGURE 7-5 LEAN SOFTWARE ENGINEERING SYSTEM .................................... 55
FIGURE 7-6 CONTINUOUS RISK MANAGEMENT SYSTEM .................................. 58
FIGURE 7-7 RELATIVE COST OF ERROR CORRECTION [ADAPTED FROM DAVIS 1996] ............. 60
FIGURE 7-8 PROTOTYPING PROCESS ......................................................... 62
FIGURE 7-9 TIME BOXING CONCEPT ....................................................... 67
FIGURE 7-10 RECHTIN'S SYSTEM ARCHITECTURE CHALLENGES ......................... 73
FIGURE 7-11 LEAN STREAMLINE DOCUMENTATION .................................... 79
FIGURE 7-12 CONCEPTUALIZATION TIME BOX ......................................... 83
FIGURE 7-13 DEVELOPMENT TIME BOX ................................................... 88
FIGURE 7-14 EVOLUTIONARY TIME BOX .................................................. 95
FIGURE 8-1 PROJECT J PROCESS DEVELOPMENT MODEL BEFORE LEAN .................... 100
FIGURE 8-2 SYSTEM J LEAN DEVELOPMENT MODEL .................................... 108
FIGURE 8-3 PROJECT J BUILD CYCLE ....................................................... 109
FIGURE 8-4 OUTBOARD RAPID PROTOTYPE TRACK ..................................... 110
FIGURE 8-5 IN-LINE RAPID PROTOTYPE TRACK .......................................... 111
Table of Tables

SECTION 7.4.11 INTEGRATED DEVELOPMENT ENVIRONMENT ................................................................. 76

TABLE 7-1 CONCEPTUALIZATION TIME BOX ACTIVITIES ................................................................. 84

TABLE 7-2 DEVELOPMENT TIME BOX ACTIVITIES ............................................................................. 89

TABLE 7-3 EVOLUTIONARY TIME BOX ACTIVITIES ............................................................................ 95
List of Acronyms

ACM       Association for Computing Machinery
ADF       Architecture Development Folder
ARC       Army Reuse Center
ATA       Army Technical Architecture
C3I       Command, Control, Communication and Intelligence
C4        Command, Control, Communication and Computer
CASE      Computer Aided Software Engineering
CDRL      Contract Data Requirements List
COE       Common Operating Environment
COTS      Commercial-off-the-shelf
CSCI      Computer Software Configuration Item
CWTH      Code Walk Through
DAWG      Design Analysis Working Group
DBDD      Database Design Document
DID       Data Information Document
DoD       Department of Defense
DRR       Detailed Design Review
DSM       Design Structure Matrices
DSMC      Defense Systems Management College
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCL</td>
<td>Functional Capability List</td>
</tr>
<tr>
<td>FRR</td>
<td>Functional Capability List Requirements</td>
</tr>
<tr>
<td>HM</td>
<td>Hardening Methodology</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>ICD</td>
<td>Interface Control Document</td>
</tr>
<tr>
<td>IDD</td>
<td>Interface Design Document</td>
</tr>
<tr>
<td>IMVP</td>
<td>International Motor Vehicle Program</td>
</tr>
<tr>
<td>IPT</td>
<td>Integrated Product Team</td>
</tr>
<tr>
<td>IRPM</td>
<td>In line Rapid Development Methodology</td>
</tr>
<tr>
<td>IRS</td>
<td>Interface Requirements Specification</td>
</tr>
<tr>
<td>LSES</td>
<td>Lean Software Engineering System</td>
</tr>
<tr>
<td>MAC</td>
<td>Macintosh Computer</td>
</tr>
<tr>
<td>MIL-STD</td>
<td>Military Standard</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>MRB</td>
<td>Materials Review Board</td>
</tr>
<tr>
<td>MS</td>
<td>Microsoft</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>ORPM</td>
<td>Outboard Rapid Prototype Methodology</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PEO</td>
<td>Program Executive Office</td>
</tr>
<tr>
<td>RCL</td>
<td>Reuse Capability Level</td>
</tr>
<tr>
<td>RDM</td>
<td>Rapid Development Methodology</td>
</tr>
</tbody>
</table>
ROSE  Rational Rose Object Oriented Tool Set
RTM  Requirements and Traceability Management Tool Set
SDD  Software Design Document
SDF  Software Development Folder
SDM  System Design and management Program
SE  System Engineering
SEL  Software Engineering Laboratory
SRS  System requirements Specification
SSDD  System/Sub-System Design Document
SSS  System Segment Specification
TAWG  Task Analysis Working Group
TDS  Textron Defense Systems
UNIX  Unix Operating System
WP  Work Package
WWW  World Wide Web
Overview of Thesis

The goal of this thesis is to present a software engineering system based on lean principles and systems thinking. Figure 1-1 provides a flow chart of the sections. Section 1 introduces the issues and driving factors behind the research. Section 2 discusses systems and systems thinking. The importance of thinking in circles (loops of cause and effect relationships) versus a linear view is highlighted and supported with examples. Sections 3 through 6 examine the lean paradigm. Lean system objectives, principles and key elements are analyzed within the framework of software engineering. Section 7 recognizes the lean software engineering system vision, describes alternative strategies and describes the key characteristics of the model. Section 8 contains a Department of Defense case study with the application and tailoring of the lean system to meet the needs and desires of the customer.
Section 1 Introduction

This thesis deals with the challenges of developing large-scale complex systems. Specifically, the system defined within this thesis is for development of software for the Department of Defense based on lean principles\textsuperscript{1}. Although this thesis focuses on the Department of Defense environment, the concepts and principles can be abstracted and tailored for the commercial software development community. The system is based on a set of lean principles, concurrent engineering methods, continuous risk management activities, multiple system development methodologies, and an iteration based approach over time.

The DoD spent nearly 43 billion dollars on computer information systems in 1995.\textsuperscript{2} DoD acquisition accounted for 35 billion dollars spent on the development of software to support operational DoD systems (Weapons, Command, Control, Communication, Intelligence) while supporting information systems; financial, human resource and information exchange requirements and other non-weapon type systems accounted for 7 billion.\textsuperscript{3} While DoD software needs have grown much more than corresponding hardware needs, the cost for software has grown by a factor of 3 for software and only a factor of two for

\textsuperscript{3} Ibid, 1994.
hardware. This data presents a major challenge for the DoD for software development to address the cost and corresponding schedule and quality problems.

The Department of Defense has defined Military Standards (MIL-STD's) as a minimum set of regulations for developing software. The Defense System Software Development Standard (MIL-STD 2167A) is for developing "mission critical" software, it defines a set of minimum standards and guidelines that must be followed and verified in software development. MIL-STD-498 supersedes MIL-STD-2167A allowing for more tailoring and flexibility for implementation of software development regulations. The Lean Software Engineering System incorporates the 2167A and 498 requirements into the model design and processes.

Why do military standards exist? The Defense Systems Management College (DSMC) has identified critical reasons for establishing military standards for mission critical systems:

- Systems are large, complex, usually involving hundreds or thousands of engineers
- Many government software development programs involve large numbers of contractors
- Today's systems are technically complex

---

• System integration issues for "Joint" military efforts that require multiple systems converging into system of systems

• Complex system requirements that are often "cutting edge" or "state of the art"

• Consequences of system failure can be life threatening and very costly

• Government program managers are often less experienced than commercial program managers in software engineering

• Government program offices are not always directly involved in the detailed technical decisions made by contractors

Thus the need to develop these complex software systems while adhering to the MIL-STD’s presents a positive and advantageous "development system environment" but what has occurred is a defense contractor industry of projects that fail to achieve the following goals:

• Cost: Meeting or completing below the planned cost proposal

• Schedule: Meeting or completing before the planned schedule proposal

• Technical: Meeting all defined functional system requirements of the customer

These goals have been established to quantify the four essential parts to evaluating a system development activity. The goals cover both technical and

---

management areas of system development while focusing on the ultimate goal, a satisfied customer.

Section 2 Background and Motivation

The ability to develop large-scale complex software systems presents major challenges for both commercial and government engineering professionals. Albert Lederer and Jayesh Prasad report two-thirds of all software projects substantially overrun their original schedule estimates. Large complex systems overrun 25% to 50% of the time, and as projects grow in size and complexity, schedule slip grows proportionally. A 1979 study by the Comptroller general studied nine Mission Critical Department of Defense programs totaling 6.8 million U.S. dollars resulted in 98% unusable software with only 2% of the software usable as delivered. Although this study is 19 years old, it highlights the infancy of software engineering and the need to focus on quality and sound engineering discipline. A more recent survey conducted by MIT and Pugh-Roberts associated reports the majority of development efforts fail to achieve cost and schedule objectives. The American Programmer states “thousands of software managers commit career suicide each year.” These managers do not have clear notions or mental models on how to develop software. Today, many

---

software-engineering programs exceed the total 6.8 million amount but still encounter some of the same challenges that prohibit the attainment of the project goals.

These examples highlight the need for a paradigm shift in the software-engineering arena. I propose that this paradigm shift is the "The Lean Software Engineering System (LSES)."

The need to develop software faster has caused the industry to seek quick fix and fad initiatives that do not have long term gains for the learning organization. Declining budgets and the competition to get software to market before the next company or to meet military concept of operations for future defense requirements is pushing engineers and program managers to migrate toward shorter development cycles while meeting stringent engineering standards.

**Section 2.1 Systems: Working Definitions**

This thesis defines and describes a system for engineering software. Webster's New International Dictionary defines as system as:

- An aggregation or assemblage of objects united by some form of regular interaction or interdependence; a group of diverse units so combined by nature or art as to form an integrated whole; to function, operate, or move in unison and, often, in obedience to some form of control, an organic or
organized whole; as, to view the universe as a system; the solar system; a new telegraph system.\textsuperscript{13}

A systems approach is used for the evolution of the Lean Software Engineering System. Simon Ramo, as quoted in Gibson describes the systems approach as:

- "In the systems approach, concentration is on the analysis and design of the whole, as distinct from...the components or parts...The systems approach relates technology to the need, the social to the technological aspects; it starts by insisting on a clear understanding of exactly what the problem is and the goal that should dominate the solution and lead to the criteria for evaluating alternative avenues...The systems approach is the application of logic and common sense on a sophisticated technological basis...It provides for simulation and modeling so as to make possible predicting the performance before the entire system is brought into being. And it makes feasible the selection of the best approach from the many alternatives.\textsuperscript{14}

The systems approach is a process for the evolution from the time a need is identified through the time the system is finally decommissioned. Drew describes the systems approach as "a approach striving to be logical, consistent, [and] objective...in analyzing systems and solving problems. It recognizes the need to

\textsuperscript{13} Webster's New International Dictionary, Unabridged, 2nd Ed.
make compromises and tradeoffs among system factors. It facilitates the selection of the best approach from the many alternatives.\textsuperscript{15}

**Section 2.2 Systems Engineering**

System Engineering (SE) is not a traditional engineering discipline. Instead of focusing on a technical specialty, system engineering ensures all aspects of the system are integrated together in an optimal manner and function together as an integrated unit within some defined boundary. The systems approach is used in the formulation of the Lean Software Engineering System for the Department of Defense.

**Section 2.3 Systems Thinking**

Today's need to develop large-scale complex systems requires one to view the whole versus single elements of the system. Systems' thinking is based on the spirit of two disciplines:

1. "Seeing interrelationship rather than linear cause-effect chains"

2. "Seeing processes of change rather than snapshots"

\textsuperscript{15} Drew, Donald R.; *Systems Dynamics: Modeling and Applications*, Blacksburg, Virginia, p. 5.
The lean software engineering system embraces this type of thinking in the structure and process flow of the system. The integration of systems thinking into the design of the model and how the model is to be implemented brings balance and an ability to organize and understand technical and management issues that arise during complex software engineering activities.

One can view the world of complexity from three perspectives, events, patterns of behavior or in a system structure. Below are two examples of the three views of system structure:
Figure 2-1 illustrates a systems view of evaluating problems and situations by seeing the cause and effects of multiple decisions on a particular subject. Once system thinkers understand the interrelationships and the processes flows,
patterns of behavior can be identified and finally the events causing this system
dynamic viewpoint\textsuperscript{16}.

\textbf{Figure 2-2 Systems Thinking Viewpoint}\textsuperscript{17}

System thinkers must have the ability to step back and attain the vision of the
system including issues concerning how the system works, issues challenging
the ability of the system to function as designed and to develop strategies for
dealing with the issues concerning the system development effort. Instead of

\textsuperscript{16} Lynesis, Jim, MIT - System and Project Management Course Packet, September 16, 1997.
analyzing only one component of the system, the systems thinker will take into account the effect of his or her decision on the whole system. Systems thinkers will formulate strategies to deal with the complex issues of system development based on a prioritization criterion for the entire system and implement the strategies based on the ability to leverage the most gain toward specific system requirements. All decision making on the system development activities should directly affect the needs of the development effort and ultimately the customer needs.

Section 2.4 System Dynamics: The Rework Cycle

A key concept within systems thinking is the rework cycle. MIT Professor Jay Forrester17, the father of system dynamics, discovered that many organizations fail to identify or address the concept of rework. First, rework is not planned into the overall development schedule and results in many projects failing to achieve cost, schedule, technical and success goals. Second, non-lean development systems do not identify and correct the problem of rework. The lean software engineering system presented here strives to eliminate the rework cycle through the implementation of lean principles. Even though the lean system strives to eliminate rework, the need to use the data extracted from this system thinking is an invaluable data set for successful large-scale complex system development. The rework cycle consists of a collection of stocks and activity flows for each

---

chunk of work of a development project. The first stock consists of work to be completed over time x. The second stock is work completed over some time y. The third stock is the undiscovered rework that has occurred over time. The forth stock is known or acknowledged rework that is accounted for in future project planning. The graphical representation represents the rework cycle.

Figure 2-3 The Rework Cycle

Rework System Drivers

Resources → Work Being Executed → Production → Quality

Work to be Done

Work Really Done

Acknowledged Rework

Undiscovered Rework

Rework Discovery

Section 3 Lean Software Engineering System Objectives

The primary objective of this thesis is to develop a Lean Software Engineering System based on the Lean Paradigm. The Lean Software Engineering System will enable engineering teams (including both commercial and government organizations) to better achieve the three previously defined goals of large-scale complex system development, cost, schedule, and technical success. The Lean Software Engineering System will provide a model that accomplishes the following objectives:

- Integration of lean principles into the software development process
- Promotion and harmonization of the systems thinking (wholistic) paradigm at all levels of the software development process
- Implementation of a risk management system to identify, analyze, plan, track, control and communicate the technical and management risk in the software development process
- Application of Rapid Prototype Product Development Methodologies
- Application of Crystallization Product Development System Methodologies
- Utilization of an Iteration Approach to system development (Spiral Development)
Figure 3-1 The Lean Puzzle

- Systems Thinking (Circles and Feedback)
- Continuous Risk Management
- Lean Principles
- Rapid Prototyping
  Crystallization
  Product Driven
- Time Boxing
  Reuse
Section 4 Lean Principles

What is lean? The most basic definition of lean is the shortening of cycle times and the elimination of waste. The lean paradigm strives for transformation to a value oriented organization implementing teamwork, communication, efficient use of resources, elimination of waste and continuous process improvement. Much of the academic research in lean initiatives is focused on manufacturing automobiles and other types of hardware products. This thesis proposes abstracting the same principles and creating a lean system for developing software products. The same problems that hamper automobile production also occur in the software development arena. The current Department of Defense software development industry is really a "craft" industry for product development. Each project is unique but commonality of components does exist. For example, a common language, C++, is often used with approved standards for mission critical systems. The requirement for inter-operability among differing systems often requires the reuse of common software components that make up the differing systems. The Software Engineering Institute report on "State of Product Line Practice" presents research indicating a shift from craft to Product Line development. A product line is a group of products sharing a common set of functionality for a particular customer, concept of operations or purpose. The

DoD is also recognizing the product line approach to reuse cost, schedule and technical risks. A DoD directive, regulation 5000.1 and 5000.2-R is focusing on reuse and commercial practices such as product line approach to software acquisition. Success stories on product line approaches have dramatically reduced cycle times in commercial industries. Motorola created a family of one-way pagers with 80% reuse of software and hardware resulting in a four-fold development cycle time reduction. Other commercial implementations show similar results in industries ranging from air traffic control systems (Raytheon), engines (Cummins), telecommunication systems (AT&T, Ericsson, Kokis, Lucent Technologies), and information systems (Buzzelo).\textsuperscript{20} The recognition of the craft-based approach has been addressed with many different solutions for software development. The approaches include process-oriented (sequential flow of computation using traditional programming language), data-oriented (based on data files and data base technologies), and object-oriented (based on classes and objects that encapsulate different behavior modes and states). These approaches have not elevated the software engineering industry from the craft based approach. A Lean Aircraft report on lean software/hardware development states the "root cause of these problems is a lack of science for software development."\textsuperscript{21} The same analysis using a software development methodology called cleanroom engineering addresses quality and productivity challenges by applying rigorous engineering practices to achieve "intellectual control over the

\textsuperscript{20} CMU-SEI-1998
Finally, the lean software engineering system will employ a process flow that breaks the system under consideration into small manageable chunks developed by small, focused project teams. The teams work in a rapid prototype process flow and eventually come back and optimize the developed software in a crystallization process flow to harden the final product. Small-integrated product teams are employed. An integrated development environment provides a framework for communication and sharing of data and information on a continual basis. The following discussion describes the principles upon which this lean system is based throughout the following thesis.

The shift to lean software development has shown benefits on NASA Software Engineering Laboratory (SEL) improvement programs for software development of mission critical systems. The SEL over a four year period “through the implementation of reusable generalized architectures has led to a 300% increase in software reuse per system and overall cost reduction of 55% during the next four years”.23 The shift to lean can also show other potential long term benefits to the software development discipline. These benefits include reduction of development times throughout the entire development process, the ability to dynamically keep pace with technological and customer change requests, higher productivity leading to cost reduction, and continuous risk reduction of management and technological issues through prototypes and reuse.

---

22 LEAN 95-01, p. 45.
The software engineering system defined in this thesis is based on a set of lean principles. James P. Womack, Daniel T. Jones, and Daniel Roos together with MIT's International Motor Vehicle Program (IMVP) define the lean paradigm as a set of principles based on the follow elements:

- Value
- Teamwork
- Communication
- Efficient Use of Resources and Elimination of Waste (Muda)
- Continuous Improvement\(^\text{24}\)

This thesis will incorporate these principles into the basis of the software development environment and strive to achieve the same type of results that automobile production industry has attained in some cases to date.

The Department of Commerce reports the DoD aerospace industry shipments declined 11 percent in 1993 and 1994. The aerospace industry has historically gained half of its revenues from the DoD. The same report states that commercial industry is also experiencing declines that will force better development approaches that are cost effective and embrace quality.\(^\text{25}\)

\(^{25}\) LEAN 95-01, p. 76.
Section 4.1 Lean Value

Merriam-Webster Dictionary defines value "as a relative worth, utility, or importance"\(^{26}\)

The lean paradigm defines value by the product or services best prescribed to meet the needs and desires of the customer. Value must be defined as the minimal set of products and services that enable a software engineering team to meet cost, schedule, and technical goals of system development.

Section 4.2 Lean Teamwork

Merriam-Webster defines teamwork as "work done by several associates with each doing a part but all subordinating personal prominence to the efficiency of the whole."\(^{27}\)

The lean team is one comprised of small numbers of personnel (as required depending on the size and complexity of the product) and a commitment for the entire life of the product development activity. The small teams are comprised of multi-functional disciplines to comprise a truly integrated product development team. These small teams will not report to the functional organizational during

\(^{26}\) Merriam-Webster WWW Dictionary
\(^{27}\) Merriam-Webster WWW Dictionary
their time on this project development life cycle, but hey remain affiliated with the functional group for future activities.

Research performed by the Lean Aircraft Initiative reports success on the migration from traditional matrix organizations to product based teams. The premise of the product teams is that "individuals will retain their area of expertise and broaden their understanding of their functional area expertise with fellow team members." This leads to a team of integrators instead of generalist who do not have a functional area of expertise. Textron Defense Systems (TDS) reorganized based on this product team focus and reported attaining or exceeding all financial goals for the 1993 period. TDS experienced a sales decline of 15% and work force reduction of 850 people in 18 months but showed increases in profits, cash, and return on investment for the organization.29

NASA SEL performed experiments on software development projects that placed emphasis on leveraging human discipline and teamwork. The SEL discovered that great leverage lies in process improvement opportunities that involve the discipline of the engineer to perform some action in the development process. The SEL used a methodology called "cleanroom" to examine the results of human leverage in software development. The study showed that defects in software were lower for smaller projects (less than 50 thousand lines of code developed in-house) versus large-scale joint-contractor developed systems

(Greater than 400 thousand lines of code). The goal of the experiment was not to adopt of "cleanroom", but the need to empower teams and leverage aspects of software development that can be standardized and eventually optimized for NASA acquisition efforts. The SEL has reduced error rates on Flight Dynamic Division software development efforts by 85% over a 15-year reporting period due to leveraging of the human element and employee empowerment of structured processes.\textsuperscript{30}

\textit{Section 4.3 Lean Communication}

Merriam-Webster defines communication as "a process by which information is exchanged between individuals through a common system of symbols, signs, or behavior"\textsuperscript{31}

The lean system promotes communication among product sub-teams and within each individual team. Good communication will help mitigate the risk of design problems and other engineering challenges that arise in the development process. Individual team members should not hesitate to identify potential conflicts as early as possible. This early identification of risk will have an overall aggregate positive effect on the entire process. Another lean paradigm shift is the involvement of large numbers of team members early in the process to interact and understand the real customer need and to have input on technical

\textsuperscript{29} LEAN 94-01, p. 33.
\textsuperscript{30} Pajerski, 1996, p. 5.
\textsuperscript{31} Merriam-Webster WWW Dictionary
decision making before the detailed work is started. The lean software
development system should employ an integrated information knowledge
center that provides a common database for all data and relevant information
about the product accessible by all product teams. The integrated development
system will be discussed in more detail under system key characteristics.

Section 4.4 Lean Efficient Use of Resources and Elimination of Waste

Merriam-Webster defines efficiency as "something productive of desired effects;
especially productive without waste."

The lean paradigm strives to best utilize resources accomplishing tasks along the
value stream. Once the customer defines value up front and resources are best
allocated to satisfy the engineering and managerial tasks required meeting those
value needs, resources will be used efficiently. Efficient use of resources is a
continuous effort that needs to be managed and optimized to best meet the
development goals. The utilization of resources in an optimized manner will lead
to elimination of waste.

Section 4.5 Lean Continuous Improvement

1. the act or process of evolving

---

32 Merriam-Webster WWW Dictionary
2. a: the state of being refined; especially: enhanced value or excellence

   b: an instance of such improvement: something that enhances value or excellence

The lean principle of continuous improvement is directly related to goal of perfection. All lean organizations strive to "perfect" or best achieve the goals and desires of their customers. A system for continuous improvement for both technical and managerial goals should be established within the organization. The continuous improvement system should be communicated and highlighted to all members of the engineering team, both engineering team and customer.
Section 5 Lean Elements

The MIT International Motor Vehicle Program\textsuperscript{33} has identified the following essential elements for lean transformation. The list represents four categories of elements that form the essence of the lean software engineering system. Only the elements directly related to software engineering are mentioned. Elements only related to manufacturing have been omitted from this thesis.

Section 5.1 Lean Development Process

- Multi-functional teams
- Integrated product teams
- Well defined development processes
- Supplier (vendor) participation

Prototypes for risk mitigation and cycle time reduction

Section 5.2 Lean Management Organization

- Broad tasking of direct employees
- Few layers of management
- Decision authority delegated to as few as possible
- General management has different basic function
- Employees viewed as fixed cost
• Management works to balance workload
• Significant training investment
• Few skill classes/flexible job assignments
• Advancement mainly in skills/challenges - not job titles
• Bonuses tied to team/company performance

Section 5.3 Lean Customer/Supplier Relations

• Continuing evaluation of customer needs
• Rapid changes in product when need identified
• Customer interface people part of development teams
• Extensive training of customer interface people
• Shared development when possible
• Long term relations
• Joint cost reduction when possible
• Broader responsibilities to vendors when possible
• Single sourcing used sparingly

33 Womack, 1993.
Section 6 A Road Map to Lean Transformation

The first step in shifting to a lean oriented software development organization begins with defining value. As previously defined, value is the set of products or services that best meet the needs of the customer and the engineering development team. The value set is the goals at every level of the process that the team is striving to achieve without adding or subtracting additional goals along the life cycle.

The second step in lean transformation is the development of the value stream. The value stream for lean software engineering is defined as the minimal set of essential products (code, design models, trade studies, etc.) or services (mission analysis, requirements analysis, system integration, test, etc.) required for engineering the software system over the entire life cycle (conceptualization to decommission). A customer-focused approach is instituted to establish the "customer-determined products needs" and not "internal-driven activities" of the software development team to evolve the value stream. The value stream is developed and implemented using true integrated product teams totally focused on developing x product for y system. The traditional matrix and internal-need based organization is replaced by this product-focused team working within the defined parameters of the value stream. When product teams and organization deviate from the customer defined value stream, the "leaness" of system is
violated. Womack and Jones defined two types of waste\textsuperscript{35}; Type 1 Muda, or unavoidable activities to accomplish the value-stream activity or Type 2 Muda, total non-value to the system development effort. A third type of waste, defined here as Type 3 Muda, or undiscovered rework is the unplanned work that evolves and often encompasses a development effort due to product deliveries or services performed that do not have total quality results. The undiscovered rework cycle should go away once a total lean paradigm shift is achieved due to adherence with lean principles and ultimately "perfection" in the system development effort. Figure 6-1 shows a graphical view of a defined value stream.

\textsuperscript{34} Womack, James P. And Jones, Daniel T.; \textit{Lean Thinking}, Simon and Shuster, 1996, p. 29.
\textsuperscript{35} Womack, 1996, p.20.
The third step in the lean transformation road map is flow. An organization establishes flow after the value of the customer is defined and the defined value is allocated along a value stream over the defined build period. Flow is essentially an optimization of the lean system development path. The flow of the value stream steps must be choreographed to meet the customer-defined schedule that includes the required engineering tasks as determined by the methodologies or science. The utilization of Design Structure Matrices (DSM)\textsuperscript{36}

for organizing products and services and schedule management approaches such as Goldratt's Critical Chain\textsuperscript{37} allows the optimized flow to be formulated, managed and controlled throughout the product development life cycle. The optimized flow transformation addresses the infamous "batch and queue" effects that occur in a non-lean organization. This phenomenon is also observed in software development as well. Department of Defense software development projects are usually "craft" oriented development efforts. Quite often, batch and queuing does arise when moving from one development kernel to the next development kernel. For example, a non-lean software development organization may have defined a requirement analysis kernel with an ultimate deliverable of sub-system level requirement specifications. A requirements review meeting is established out in time x. Since systems sub-system's requirement specifications are different, trying to schedule all requirement reviews during the same period will cause three major challenges, batching, queuing, and resource misallocation. Requirement reviews involve the chief engineer and many important customer interfaces. The ability for many of the same people to review and put quality inputs into the requirement review exercise is not realistic. A lean organization will prevent this "batch and queue" event from occurring along the value stream. The book "Lean Thinking" discusses a major challenge for implementing flow in a lean organization. The transition from a matrix organization to a product-focused organization is a key constraint to achieving the value stream.

The fourth step on the road to lean transformation is the concept of pull. The transition to lean will allow the customer to determine their need for a particular product or system. The lean paradigm defines value, the value stream, and appropriate flow of activities all leading to an evolution of customer pull. Customer pull is the enabling of the "real need" of the product both in terms of delivery time and functionality to be determined by the customer. The pull philosophy contrasts with the "push" philosophy by the system developer determining the real need in terms of delivery time and functionality. Often, the system developer is late due to non-value activities or functionality is driven by some technological direction determined by the engineering team that may or may not be in the best interest of the ultimate customer need.

The fifth process in lean transformation is perfection. Perfection is something the lean organization will always strive for as long as it exists. This step has no stop date. Once a firm achieves the first four steps of the lean roadmap, the organization will always continue to achieve perfection in terms of the following variables:

- Continuous effort reduction
- Continuous time reduction
- Continuous cost reduction
- Continuous mistake reduction
If an organization continues to strive for perfection and concentrates on continual progress toward perfection goals while always focusing on attaining the ultimate desires of the customer, the lean initiative will be achieved.

The following section describes the lean software engineering system. The vision and corresponding system structure are discussed and supporting process elements supporting the lean paradigm identified.
Section 7 A Lean Software Engineering System

The following section describes the lean software engineering system. The vision and corresponding system structure are discussed and supporting process elements supporting the lean paradigm identified.

Section 7.1 Vision

The development of large-scale complex software systems for the Department of Defense is under going an ever-growing shift to streamlined deliveries. The days of unlimited budgets and long time lines for development cycles are disappearing with schedule driven customer requirements for deliveries. The lean paradigm shift may enable Department of Defense software acquisition process to meet the current needs of the Department of Defense customer without sacrificing the required engineering science and reliable final products of mission critical software. The following discussion describes a lean software engineering system that can be tailored to meet the needs of differing programs and projects. The intent is to layout a framework for a lean engineering system abstracted from the lean principles identified by MIT's International Motor Vehicles Program (IMVP). The model incorporates systems thinking approach by utilizing feedback loops in each process chunk. The lean system also spirals by delivering multiple
iteration of products to the end customer. The lean model also iterates from a prototype environment to a crystallization or hardening environment. The rapid prototype environment allows the engineering team to build the system and deliver it to the end user very quickly while mitigating risk. The crystallization environment allows the development team to optimize and formalize the final product. The crystallization environment is necessary to support the mission critical element of the product. The military standards and stringent test requirements of the final product at time x will be achieved using this approach.

Finally, a shift in focus from documentation-focused development to products and services focused development is initialized by chunking work into manageable pieces and developing discrete time boxes for customer interactions and demonstrations of functionality to end users. The model description will identify the key features and describe the foundation for the engineering methodology.

Section 7.2 Software Engineering Model Alternatives

This section will briefly identify alternative models for developing software. The purpose is to educate the reader on the strengths and weaknesses of the lean engineering system. The lean engineering system derived its shape from a variety of the different model approaches to achieve a lean paradigm.
Section 7.2.1 Alternative: Waterfall Method

The waterfall method\textsuperscript{38} has the following characteristics:

- Progresses through orderly sequence of steps
- Holds reviews at the end of each phase to determine approval to next phase
- Document driven
- Phases are discontinuous (No overlap)

**Strengths**

- Potential for highly reliable system produced
- Large system growth
- Management Visibility

**Weaknesses**

- Requirements must be defined and understood without open risk items
- Architecture must be defined and understood without open risk items
- Batch and queue occurrences during phases for completion of milestone
technical review meetings tat require all sub-system components complete at
the same time
- Low customer visibility on progress
- Risk mitigation is low since commitment is made to move on to the next
process area with the approved requirements or design, etc.

Section 7.2.2 Alternative: Code and Fix Model

The code and fix method\textsuperscript{39} has the following characteristics:

- Start coding immediately
- No real documentation or specifications (back of the envelope)
- No quality assurance, configuration management or requirements traceability

\textsuperscript{39} Ibid, pp. 156-157.
Strengths

- Immediate progress to customer on coded system
- Little expertise required on engineering science principles and methods
- Low overhead

Weaknesses

- Dangerous for sophisticated projects with many interfaces
- System usually not optimized
- Maintenance nightmare for design changes and code modifications
- No management visibility on progress/status
- No large growth potential for system

Figure 7-2 Code and Fix

Section 7.2.3 Alternatives: Spiral Model

The spiral method\textsuperscript{40} has the following characteristics:

Risk oriented model
Breaks development effort into mini projects based on risk

Evolves into a waterfall execution model

**Strengths**

- Works well with poorly understood requirements and poorly defined architecture
- Potential for highly reliable system
- Potential high system growth potential
- Risk identification and management is high

**Weaknesses**

- Eventual evolution to waterfall slows process
- Requires high sophistication of engineering team
- Requires strong continual leadership from technical and management team
- Schedule changes constrained within predetermined long term plan

---

Section 7.2.4 Alternative: Prototype Model

The prototype method\(^\text{41}\) has the following characteristics:

- Provides sets of consistent functionality in each increment to the verification environment for on-going integration
- Early results from testable functionality reduces risk

• Early stakeholder feedback/buy-in

• Technical risk contained in short development time-frames

Strengths

• Customer interface and buy in
• Technical risk mitigation
• Risk management
• Time to customer cycle is short
• Ability to re-plan/reengineer of design and code at anytime
• Management visibility is high

Weaknesses

• Need stable system architecture
• Pre-defined schedule milestones can be difficult
• Lower reliability of system
• Continued large-scale documentation capture of engineering elements that will be out of date during the next iteration of the development effort
Section 7.3 A Lean Software Engineering Model

The following sections describe the key features of the lean software engineering system. The discussion begins with key system features and details of the system structure and flow over time. The lean principles used to develop the lean system are emphasized as well as the systems thinking required for understanding the basis for the overall system development effort.
Section 7.4 Key Characteristics of Lean Software Engineering Model

The preceding discussion describes key factors highlighting the software development community's need for a lean software engineering system. This section describes the key characteristics that enable the lean paradigm shift for software development. The lean software engineering system establishes a set of required process characteristics that emphasize the need for a lean development system. The process is focused on building products and providing services directly related to the specific needs of the customer in the Department of Defense software acquisition environment. The lean system places great emphasis on customer interaction throughout the entire process. The lean software engineering system provides a forum for information exchange that benefits both parties in terms of value over time. The customer is involved with all aspects of the life cycle using the time boxing approach for information exchange between engineering team and end user.

Section 7.4.1 Risk Management System

The lean software development system embraces risk once it is identified. The lean organization views risk as an opportunity to mitigate a future problem of greater magnitude. The Software Engineering Institute has depicted a risk
management paradigm\textsuperscript{42} of continuous activities that is utilized over the life cycle of the project. The risk model contains five functions based on seven principles. The five functions are:

- **Identify**: Inspect and discover risks before they become problems
- **Analyze**: Transform risk data into decision-making information. Evaluate impact, probability, and time frame, classify risks, and prioritize risks.
- **Plan**: Translate risk information into decisions and mitigating actions (both present and future) and implement those actions
- **Track**: Monitor risk indicators and mitigation activities
- **Control**: Correct for deviations from the risk mitigation plans
- **Communicate**: Provide information and feedback internal (product teams) and external (customers) to the project on the risk activities, current risks and emerging risks

The seven principles forming the foundation of the risk mitigation paradigm are:

- **Wholistic perspective**
- **Forward-looking view**
- **Open communications**
- **Integrated management**
- **Continuous process improvement**

\textsuperscript{42} Van Scoy, Roger L.; Software Development Risk: Opportunity, Not Problem, Technical Report
- Shared product vision

- Teamwork

The risk principles are also the same or very similar as the lean principles and integrate well into the lean software engineering system.

Figure 7-6 Continuous Risk Management System

Section 7.4.2 Evolutionary Prototype Methodology
The purpose of evolutionary prototyping is to develop the system in increments allowing for changes to be easily incorporated after customer feedback. Evolutionary prototyping does not propose "throwing away" the prototype code or design. The goal of this type of prototyping is to build an initial prototype system that the user can execute and provide valuable feedback to the development team. The engineering team will iterate on the prototype until it is at an acceptable level for evaluation by the end user. Once the feedback is analyzed, the development team will make the decision whether to refine the prototype further or promote the prototype to the crystallization methodology.

The use of prototyping in software development can uncover defects and potential technical challenges for the development effort. A DoD study for the Defense Science Board Task Force shows the relative cost for error detection grows as the engineering effort matures.43 The analysis concluded, "the hardest part of the software effort is setting the exact requirements." Another study on the relative cost of error correction performed by Alan Davis supports the Defense Science Board Task Force study of relative cost growth. Figure 7-7 shows the phase and relative cost of correction.44

---

Figure 7-7 Relative Cost of Error Correction [Adapted from Davis 1996]

The data supports the need to understand the requirements early in the process to avoid cost and subsequent schedule over runs. The implementation of prototyping can support this requirement. Prototyping can allow the engineering team to validate the requirements through small development efforts with continuous customer feedback and lessons learned. A study performed by Boehm at TRW shows that 54% of all quality problems found on software development projects occurred after code and unit test phases. The study also discovered 83% of the errors were attributable to the requirements and design phase.\textsuperscript{45}

Supporting prototyping in software development can have a beneficial effect on the development of large-scale complex systems. The ability to enhance the validation of requirements early in the development life cycle will contribute to cost, schedule, and technical goal achievement. The Defense Science Task Force Board concluded the following:

"We believe that users can not, with any amount of effort and wisdom, accurately describe the operational requirements for a substantial software system without testing by real operators in an operational environment, and iteration on the specification. The systems built today are just too complex for the mind of man to foresee all the ramifications purely by exercise of the analytic imagination."\(^{46}\)

The first International Workshop on Rapid System Prototyping held in the Research Triangle Park identified two primary reasons for embracing prototyping in software development. First, prototyping can "assist" in understanding the "what" the software is intended to do rather than "how" the software is implemented early in the development process and second, the potential reliability gains that can come from a prototype implementation throughout the entire software-engineering life cycle.\(^{47}\)


\(^{47}\) Urban, Joseph E.; Software Prototyping and Requirements Engineering, Data and Analysis Center for Software, June 1992, p. 4.
Evolutionary Prototype Methodology has both positive and negative aspects.

The positive results include:

- Potential schedule reduction and cycle time reduction
- High visibility to program progression through product and service demonstrations
- Potential "Bang for your Buck effect" on achieving a working product the customer desires
- Potential for increased learning curve over time by development team on real customer needs

Potential negative outcomes include:

- Management and customer will set unrealistic schedules and budgets
• Non-value prototype activities (if processes are not well defined and team leaders do not stay focused on time boxing requirements)

• Unrealistic system performance expectations (this will be achieved in the crystallization phase)

• Poor Design (Again, the system will be optimized during the crystallization phase)

• Poor maintainability (The lean software engineering system will incorporate continuous integration and control of the system through the integrated development environment)

Section 7.4.3 Crystallization Development Methodology

Crystallization is a hybrid version of evolutionary prototype methodology that incorporates the feedback from the customer back into the development effort while allowing the development team to refine, redesign and optimize the design and code as required. If the customer deems the prototype product acceptable, the product will go through crystallization concurrently during the next build (Build n + 1) activity. This concurrent effort allows the development team to build the next set of functionality (Build n + 1) in a prototype environment and the customer feedback functionality in the crystallization environment (Build n).

This hardening of the software is a lean initiative that allows for mission critical software to "graduate" to an acceptable level of maturity. Mission critical
software requires rigorous testing and reliability requirements satisfaction. This hardening approach allows for both rigor and early customer input that would mitigate risk over the life of the development effort. The concurrent approach will also reduce cycle times on all phases of the development life cycle.

Section 7.4.4 Time Box Concept

Time boxes\textsuperscript{48} are established at the macro and micro level for of the lean software system model. Three macro-level boxes provide the overall framework of the system. The three boxes consist of a the following macro-level time boxes:

- System Time Box
- Sub-System Time Box
- Validation Time Box

These time boxes are developed to capture three important boundaries required to make the development process flow along in a logical and streamlined fashion. The system level time box captures the conceptualization phase of the system, the requirements analysis phase of the process and results in a system architecture framework that serves as the foundation for the evolving sub-system configurations or Computer Software Configuration Items (CSCI). Other key
activities include packaging of products and services into micro-level time boxes, technical and programmatic risk identification and prioritization of the developing activities for the build.

The sub-system time box is the second macro-level development period for detailed design, code generation, debugging, testing and integration. Two types of development methodologies are executed, rapid prototypes and crystallization or hardening of the product. In addition to current build activities, preliminary system level analysis and design activities are evaluated along with analysis on customer requirements for the next build or iteration. This concurrent activity will reduce the cycle times for the system level work on the next spiral of the process. The concurrent activity keeps the customer close to the process and the developer can use the customer inputs as valuable data points for mitigating system level architecture design risks from both a technology shift and understanding of the requirements at the system level.

The third macro-level time box is the evaluation period. This period is used for evaluation of the delivered software product by the end user. The developer also begins to evaluate the final product versus the planned product by traceability exercises to original requirements and review of lessons learned throughout each phase of the process. The lessons learned data is critical for both future risk mitigation activity and migration to best engineering practice and managerial process. Time boxing is also utilized within each process kernel. Time boxing at

---

48 McConnel, 1996, P 575.
this "working" level provides a structured approach bound by a specific time frame to demonstrate functionality to the end users rather than producing some voluminous paper document. Time boxes also bring focus to the development teams in terms of activities and time frame for the development activity. It is important for the engineering organization to form small development teams to carry out this time box work packages and to define attainable goals within the time frame. A forum to demonstrate the developed functionality to end users will replace the big milestone meetings defined in non-lean development environments that often take multiple days and focus on paper documents and slide presentations. Quite often, the information exchange is one-way without much technical exchange between the right people. One of the key lean principles is elimination of multiple layers of non-contributors within organizations. The old milestone meetings often involve high level people who do not require the same types of technical exchange of data as the engineers in the "trenches." Each project needs to decide who are the appropriate individuals to attend technical exchange meetings. The technical and/or managerial information being presented should determine who attends the meeting.
Time boxing supports software quality by providing an environment to answer three key questions: “who are the users?,” “what is important to them,” and “how do their priorities relate to the way you build, package, and support products?”

Through continuous customer interaction and quick turn around of deliverable products, the quality of the product to the customer will likely be high in terms of meeting the end users needs.

---

Section 7.4.5 Reuse-Driven Philosophy

The lean software development system desires to leverage off existing standards and proven software components previously developed that meet the needs of the current customer. A lean organization is not interested in "reinventing the wheel" and will strive to reduce cycle time with the use of existing components whenever possible. The need for government systems to migrate to common platforms and inter-operability of data among systems will support the reuse-driven philosophy. Embracing reuse products into the system design also mitigates risk of the component functioning as designed. Testing the product in the integrated system is still required and potential increase of risk is lowered and potential unknown rework is minimized. Besides not reinventing the wheel, the transformation to reuse of software will enable Department of Defense software systems to better embrace industry standards for components and better designs through integration of modular chunks of software.

A Carnegie Mellon University Software Engineering Institute case study of Celsius Tech identifies the potential benefits of software reuse. Celsius Tech is a Swedish developer of command and control information systems for defense forces around the world. In December 1985, Celsius Tech won two major contracts for the same time period. Celsius Tech could not satisfactory develop both systems at the same time with its constraints on staff and resources. Celsius Tech decided to leverage off its in-depth domain expertise and
developed an architecture that would serve as the foundation for both projects. Since this reuse strategy has been adapted, Celsius Tech has developed over 55 variations of this command and control architecture. Celsius Tech has successfully leveraged 80% exact reuse for each system and achieved delivery times an order of magnitude better than previous efforts.\textsuperscript{50} The reuse strategy has enabled Celsius Tech to remain competitive globally while reducing its workforce and achieving greater reliability in system performance and ultimately achieving greater customer satisfaction.

In January 1996, the Army Reuse Center (ARC) was tasked to conduct surveys of the state of the army reuse policy. The purpose of the survey was to determine the current reuse capability levels for Army Command, Control, Communication, and Computer (C4) programs. The study looked at the capability levels of five areas:

- Reuse Management
- Reuse Education and technology Transfer
- Domain Analysis
- Domain Implementation
- Reusable Asset Acquisition

The survey results are calculated using a denominator of 47 (number of participating projects) and a numerator based on adding all scores of Program Executive Offices (PEO) responding to particular survey sections. The reuse management survey results of 10 program executive offices ranged from 60.3 Reuse Capability Level (RCL) to 3.2 RCL with an average of 25.8 RCL with a maximum possible score of 75 RCL. The Reuse Education and Technology Transfer resulted in a range of 16.0 RCL to 0.0 RCL with an average of 5.8 RCL out of a possible 90 RCL maximum score. The Domain Analysis scores ranged from 33.2 RCL to 11.8 RCL and an average of 25.6 RCL. The total possible maximum score for Domain Analysis is 49 RCL. The Domain Implementation analysis shows a range of 53.5 RCL to 0.0 RCL and an average of 28.7 RCL. There is no upper limit for Domain Implementation or Reusable Asset Acquisition since total points are assessed on a variable number developed by each PEO. The Reusable Asset Acquisition response ranges from 115.3 RCL to 0.0 RCL with an average of 35.2 RCL.\textsuperscript{51} The Army was reasonably motivated by the proceeding results that reuse "...should be an integral part of a comprehensive software engineering process that produces quality, cost-effective systems."

Two important initiatives supporting reuse by the DoD and Army include the implementation of the Common Operating Environment (COE) and the Army Technical Architecture (ATA). These initiatives coupled with declining budgets and flexibility of MIL-STD's for software engineering are creating an environment

\textsuperscript{51} Department of the Army: Reuse Technology Assessment Report for the Department of the Army, 1996, p. 5.
that promotes product lines and reuse policies for achieving cost, schedule, and technical goals for DoD software engineering.

**Section 7.4.6 Continuous Integration and Testing**

The reason for putting together an engineering development team to develop a system for a customer is achievement of system functionality. The most reliable measure of success is the actual execution of the system functionality on a continual basis. The lean software engineering system provides an environment for daily integration of software. Risks are identified early in the life cycle and immediate risk mitigation measures are identified, analyzed, planned, tracked and controlled by the development team.

The requirement to integrate and test on a daily basis will bring structure and discipline to each time box development team. A synergy will evolve as the component interface risks are removed and the system evolves on a daily basis.

**Section 7.4.7 Architecture-Based Synthesis**

The architecture of any system represents the framework for the development team to derive the details of the sub-systems and components. The relationships, interfaces, functions, forms, and shape form the core for making the system fulfill customer requirements. Together with the system architecture
synthesis is the need to understand the domain of the system under consideration. Domain knowledge will enable the development team to shape and structure the system in an optimal configuration to best meet system requirements. The need to understand and comprehend the needs of the end user is essential to building a system. Domain engineering enables the development team to form the boundaries and structure of the system and the product behavior can be extracted and captured by the team. The flow of information and interfaces are determined by continual domain analysis. Eberhardt Rechtin recommends the following architectural techniques to accomplish the architectural needs of the system:

- Size the architecture to a manageable configuration
- Partition the system into achievable and modular pieces
- Identify risk areas among modular interfaces of the sub-systems
- Watch Key Characteristic areas along the development life cycle and mitigate as soon as possible
- Keep the system in equilibrium

The tensions in system architecture are addressed on a continual basis in the lean software engineering system. Rechtin best depicts this architecture driven synthesis in the following graphical representation\(^5\)

Section 7.4.8 Continuous Process Improvement through Systems Thinking

Systems thinking places great emphasis on thinking in feedback loops. The lean software engineering system also places great value on systems thinking through continuous process improvement. One of the goals of the system is the elimination of waste and attaining better information to achieve perfection and a total quality system for the end user. The development teams should constantly
be looking for process improvement lessons learned during execution of each
time box work package. The systems thinking approach will shorten cycle times,
strengthen the quality initiative, eliminate waste in terms of engineering effort and
non-value stream activities such as "gold capping" (engineering beyond the
specification) or risky technological solutions.

**Section 7.4.9 Small Development Teams and Customer Interaction**

The lean paradigm shift for organization structure is on integrated product teams.
The lean software engineering system institutes small focus teams that perform
activities within the time box defined time frames. The customer is involved in
the time box demonstrations of functionality and information exchange between
both parties. The inclusion of the customer is important to mitigate the potential
of non-domain area experts on the engineering team. The Association for
Computing Machinery (ACM) reports usability objectives (A usable software
system is one that enables effective and efficient competition of tasks within a
given system boundary) are not usually based on the customer need or setting.
"One common consequence is the assignment of responsibility for usability to
people who do not have appropriate training, or who are trained in behavioral
sciences rather than in more product-oriented fields such as design or
engineering" do not address the "usability" of the software system at a effective
or efficient level acceptable by the customer. The integrated cross functional team can address this issue by having the right people involved from inception to decommission of the system. These integrated product teams will have defined missions to accomplish within each prototype as defined in the time box work package. The team structure will encompass a cross-functional talent spectrum including both engineering and customer teammates. The make up of the teams will be determined by the needs of the team during different parts of the life cycle. However, a core group will be on the team from concept to decommission. This is critical for the product development effort to be successful. Inter-changeable team members without the product history will slow down the development process and effect the development of the product in terms of customer needs. If people are coming and going throughout the development process, the "human knowledge center" will be lost. The human knowledge center is something that can not be replaced by any process of CASE tool. The human element is critical to lean transformation and ultimately lean success.

Section 7.4.10 Continuous System Evolution Planning

The short build cycles and short development activities within the lean software engineering system provide for continuous short and long term planning. The needs of the customer in the Department of Defense often change as fast as the

---

original requirements are captured in a system requirements document or specification. The lean system accounts for this dynamic environment by providing continuous customer involvement and system evolution activities throughout the entire life cycle of the model. Only before each build is accomplished, is the requirement baseline established. This is possible since the work scope is appropriately shaped to meet the end users needs at time x. The short cycle times will enable the developers to turn software over to the end user quicker and feedback loops of information exchange will eventually lead to a better system in a shorter time frame.

Section 7.4.11 Integrated Development Environment

The lean software engineering system requires an integrated development environment for information creation, exchange, and storage. The integrated development environment is structured around the project knowledge center. The knowledge center contains all relevant information and data for optimization of the development process. For example, in software development, the need to create software development folders that contain the different sub-system source code, test results, white papers, etc. are required to capture the development results in some controlled fashion. A requirements tool is required to model the system and provide trace-ability of the system for verification of customer need. Databases with change information is required to present a history of the
evolving system over time as well as data points for analysis on trades and other technological challenges. Integrated e-mail, product team web pages, schedules and other relevant information will serve as an infrastructure for achieving the lean principles of communication and team work.

Section 7.4.12 Lean Documentation

Non-lean Department of Defense software development organizations are driven by documentation. All military development standards state in the Data Information Document (DID) what, when, how, and where these documents are to evolve. The lean paradigm shift will have a big impact in this area as the focus shifts from documentation to products and services. All data will be captured and accessible on line in the software development folders and architecture development folders through report generation or via the tool interfaces. The value of generating formal documentation with the short development periods is not a great benefit to the customer or the development team. When a system is fully crystallized at a formal version delivery, the formal documentation will be generated and presented to the customer as an contractual deliverable. Just as the system is evolving by prototypes and customer feedback to be hardened at maturity, documentation will go through a similar process until it is at a mature state for delivery and will provide value to the customer.
Lean documentation should be in a format compatible with both the customer and the CASE tool environments. The data should be accessible across differing platforms (UNIX, PC, and MAC) with a common interface product. The World Wide Web (WWW) and Hypertext Markup Language (HTML) are examples of cross platform data exchange. When formal documentation is generated at version, a document generation tool will extract the data from its native CASE tool environment and populate the document template in the MIL-STD format. Since the data can be extracted from the CASE tool by the engineering team, formal technical documentation generation is not recommended.

The focus on CASE tools for information will reduce the redundant activities of document generation and return the engineering team focus to products and services. Research shows that an integrated CASE tool environment can reduce cycle time and errors during the development effort. United Technologies Corporation improved its software development process through Pictures-to-Code standardized process. United Technologies was able to reduce cycle time by 40% and error rates by 60%.55 The cost of development and schedule were each reduced 50% by embracing the tool set and standardized processes.56 The use of automatic code generation CASE tools has shown great potential for cycle time reduction, reuse, and error decline. United Technologies, General Electric, McDonnell Douglas, and Aerospatiale show productivity increases of 66%, 35%,

25%, and 55% respectively.\textsuperscript{57} These productivity results show great potential for other development efforts to gain advantages from low risk shifts.

Figure 7-11 Lean Streamline Documentation

Section 7.4.13 Streamlined Development Control and Management

The objectives for streamlined control and management are to empower the teams and allow for demonstration of functionality via time boxing and demonstrations. The large milestone meetings lasting for multiple days are not recommended. This type of control pushes the focus to building systems, not

\textsuperscript{57} Ibid, 1997, p. 80.
reporting about them in overhead presentations or fancy documentation. The technical management requirements will flow down to the primary points of contacts working the real issues of the development effort and not passed up the command line. The full utilization of the development tools and development environment will also provide for a controlled environment without too much restriction in terms of bottlenecks from non-value activities such as submission of engineering data through multiple functional organizations for control and approval. Each individual team based on an approved control scheme agreed upon by the customer can achieve this same function. As a general rule, prototype development control should remain as low as possible and empowered to the engineering development teams. The crystallization of the software will require control to rise to high organizational levels to ensure data integrity and customer approval of the system.

The use of formal engineering technical reviews serves as the process for assessment and evaluation of software products. However, these reviews can be expensive and time consuming for valuable engineering and management personnel. Russell reports that formal engineering reviews have run development programs 35% - 20% of their overhead budget. The same study also concluded that scheduling these meetings into the development process lengthened the development effort almost a third in one particular analysis. Another study in the journal for Empirical Software Engineering concluded that

---

traditional technical review meetings are costly and are successful at filtering out "false positives" but the formal meetings did not show great differences in error or quality detection. The subjects were surveyed and preferred the "live" meetings as they perceived greater quality benefit from face to face interaction although the study did not show any evidence of that perception.\textsuperscript{60} The use of time boxes and employee empowerment may provide the best of both worlds in terms of information exchange between developers and customer and a specific focus on a small amount of data for a particular time box product or service.

A Lean Aircraft Initiative case study performed on operator inspection certification in the manufacturing area shows potential high returns for employee empowerment and reduction of technical reviews meetings. A certification program was initiated to achieve the following goals:

- To provide immediate direction of operator, machine, and process errors
- To provide an alternative to 100\% in-line inspection by quality personnel
- To provide greater incentive to manufacturing personnel to identify parts accurately
- To provide feedback to operators to prevent re-occurring errors.

The result of one company's certification program shows a 66\% reduction in Material Review Board (MRB) actions for all products and a 90\% reduction for

\textsuperscript{59} Russell, 1991.
military products. Another company reports shop lead times decreased from 12.1 weeks to 7.3 weeks because of the certification program and employee empowerment.

Section 7.5 Conceptualization Time Box Activities

The purpose of the conceptualization time box is to develop the system architecture, establish a requirement's base line, constitute specific objectives for the development effort (time box work packages) and identify the technical and managerial risk elements.

The following activities represent a framework to guide the development team. Depending on the development methodology (Object Oriented, Structured Analysis, etc.) the activities are to be tailored to meet each program's needs. These activities are presented as a starting point of the tailoring and synthesis process.

The following table lists a framework for activities to be executed during the Conceptualization Time Box Activity. The list of activities should be tailored to meet the needs of individual programs.

---

**Figure 7-12 Conceptualization Time Box**

- **Conceptualization Level Time Box**
  - **Customer Need**
    - Start & End
    - System Arch
    - TBWP Priority
    - Risk Plan
    - Schedule
  - **Concept Activity**
  - **Requirements Activity**
  - **System Arch Activity**

**Key Elements**
- Time Box Work Package (TBWP) Generation
- DSM Developed
- Technical & Project Risk ID
- Project Schedule Development
- Project Prioritization Est.
- Domain Analysis
- Trade Studies
- Project Strategy/Planning
Table 7-1 Conceptualization Time Box Activities

<table>
<thead>
<tr>
<th>ACTIVITY</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptualization</td>
<td>• Validate and refine mission objectives and requirements. Identify and capture the functional, performance, operation and support requirements</td>
</tr>
<tr>
<td>(Macro Level Time Box)</td>
<td>• Develop Operations Concept</td>
</tr>
<tr>
<td></td>
<td>• Identify all external interfaces to the system</td>
</tr>
<tr>
<td></td>
<td>• Identify performance constraints</td>
</tr>
<tr>
<td></td>
<td>• Identify/Communicate Risks</td>
</tr>
<tr>
<td></td>
<td>• Conduct Concept Time Box review with customer</td>
</tr>
<tr>
<td>Requirements</td>
<td>• Develop system requirements, data</td>
</tr>
<tr>
<td>(Micro Level Time Box)</td>
<td>• Iterate on system constraints and scenarios</td>
</tr>
<tr>
<td></td>
<td>• Expand, decompose and integrate system requirements</td>
</tr>
<tr>
<td></td>
<td>• Create system requirements base line</td>
</tr>
<tr>
<td></td>
<td>• Allocate requirements to system scenarios</td>
</tr>
<tr>
<td></td>
<td>• Identify requirements risk issues</td>
</tr>
<tr>
<td></td>
<td>• Establish verification program</td>
</tr>
<tr>
<td></td>
<td>• Develop system test plan</td>
</tr>
<tr>
<td></td>
<td>• Define methods for requirements verification</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Develop set of external requirements</td>
</tr>
<tr>
<td></td>
<td>• Start conceptual data model</td>
</tr>
<tr>
<td></td>
<td>• Identify key characteristics</td>
</tr>
<tr>
<td></td>
<td>• Capture approved base-line</td>
</tr>
<tr>
<td></td>
<td>• Identify/Communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box demonstrations</td>
</tr>
<tr>
<td>Architecture</td>
<td>• Verify all required data from conceptualization and requirements activities is available for architecture analysis</td>
</tr>
<tr>
<td>(Micro Level Time Box)</td>
<td>• Plan architecture selection and evaluation criteria</td>
</tr>
<tr>
<td></td>
<td>• Create and evaluate candidate architecture and select optimal architecture based on customer requirements</td>
</tr>
<tr>
<td></td>
<td>• Refine architecture</td>
</tr>
<tr>
<td></td>
<td>• Determine the configurations of development and test environments in order to anticipate requirements or constraints associated with development which may differ from an operational setting (The lean development</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>environment will strive to mirror the operational environment as close as possible). This mirroring of the real operational environment will mitigate risk associated with engineering the wrong system for the wrong environment</td>
</tr>
<tr>
<td></td>
<td>• Update requirements as necessary</td>
</tr>
<tr>
<td></td>
<td>• Update risk management system</td>
</tr>
<tr>
<td></td>
<td>• Identify/Communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box demonstrations</td>
</tr>
<tr>
<td>Planning and Goal Development</td>
<td>• Ensure requirement base-line satisfies customer needs</td>
</tr>
<tr>
<td>(Micro Level Time Box)</td>
<td>• Prioritize requirements based on customer need</td>
</tr>
<tr>
<td></td>
<td>• Rank requirements based on technical complexity</td>
</tr>
<tr>
<td></td>
<td>• Development Design Structure Matrices to represent complicated requirement and task dependencies</td>
</tr>
<tr>
<td></td>
<td>• Develop time box work packages to bundle work into manageable development packages</td>
</tr>
<tr>
<td></td>
<td>• Evolve Master and detailed schedules from DSM</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>and Time Box data</td>
</tr>
<tr>
<td></td>
<td>• Identify/Communicate risks into risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box demonstrations</td>
</tr>
</tbody>
</table>

**Section 7.6 Development Time Box Activities**

The following table lists a framework for activities to be executed during the Development Time Box Activity. The list of activities should be tailored to meet the needs of individual programs.
Figure 7-13 Development Time Box

Customer Need

Time Box #1  Time Box #2  Time Box …n

Development Time Box

Multiple Prototype Efforts

Code

Detailed Design

Debug

Test

Prototype Daily Integration Environment

Multiple Concurrent Prototype Efforts

Capture Lessons Learned

Advanced Conceptualization for Build n + 1 Continuous Risk Management System Control Through Demos and Tools/IDE

Build Delivery

Version Delivery at time x

Capture Lessons Learned

Multiple Concurrent Crystallization Efforts

Code

Detailed Design

Debug

Test

Crystallization Daily Integration Environment
<table>
<thead>
<tr>
<th>TIME BOX ACTIVITY</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed Design (Macro Level Time Box)</td>
<td>• Build software requirements model in Requirements Tool</td>
</tr>
<tr>
<td>System Requirements (Micro Level Time Box)</td>
<td>• Perform analysis in the context of the chosen development methodology (Object Oriented or Structured Analysis)</td>
</tr>
<tr>
<td></td>
<td>• Analyze models to allocate requirements to components</td>
</tr>
<tr>
<td></td>
<td>• Refine models as required</td>
</tr>
<tr>
<td></td>
<td>• Provide inputs to Data Dictionary</td>
</tr>
<tr>
<td></td>
<td>• Analyze system scenarios/modes and derive requirements as appropriate</td>
</tr>
<tr>
<td></td>
<td>• Restate, refine and allocate requirements to each component</td>
</tr>
<tr>
<td></td>
<td>• Continue to perform verification activities</td>
</tr>
<tr>
<td></td>
<td>• Determine reuse for each component</td>
</tr>
<tr>
<td></td>
<td>• Trace all requirements (system and software) to software components in requirements tools</td>
</tr>
<tr>
<td></td>
<td>• Confirm software requirements support</td>
</tr>
<tr>
<td></td>
<td>operational requirements defined during</td>
</tr>
<tr>
<td>TIME BOX ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>conceptualization phase</td>
</tr>
<tr>
<td></td>
<td>• Define component testing, performance requirements and resource allocations</td>
</tr>
<tr>
<td></td>
<td>• Continue updating conceptual data model, data architecture and data dictionary</td>
</tr>
<tr>
<td></td>
<td>• Create data migration strategy if applicable</td>
</tr>
<tr>
<td></td>
<td>• Continue to update requirements trace-ability in requirements tool</td>
</tr>
<tr>
<td></td>
<td>• Continue to update operations concept, system requirements, architecture design as necessary</td>
</tr>
<tr>
<td></td>
<td>• Develop software test/demo material</td>
</tr>
<tr>
<td></td>
<td>• Continue to update/communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box demonstrations</td>
</tr>
<tr>
<td>Development</td>
<td>• Analyze software requirements and system level architecture to derive software components</td>
</tr>
<tr>
<td>(Macro Level Time Box)</td>
<td>• Identify critical components (key characteristics) of software architecture</td>
</tr>
<tr>
<td>Software Design</td>
<td>• Identify interfaces between components and communication protocols</td>
</tr>
<tr>
<td>(Micro Level Time Box)</td>
<td></td>
</tr>
<tr>
<td>TIME BOX ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>• Decompose components to develop structure</td>
</tr>
<tr>
<td></td>
<td>• Continue to trace requirement changes in requirement tool database</td>
</tr>
<tr>
<td></td>
<td>• Perform trades as required to evaluate potential use of COTS and/or GOTS products for software components</td>
</tr>
<tr>
<td></td>
<td>• Identify any potential reuse component candidates for the software architecture</td>
</tr>
<tr>
<td></td>
<td>• Develop and validate any algorithms associated with key components</td>
</tr>
<tr>
<td></td>
<td>• Identify data flow channels</td>
</tr>
<tr>
<td></td>
<td>• Assign components to logical and physical architecture entities</td>
</tr>
<tr>
<td></td>
<td>• Design logical database (including components that maintain and access data including table definitions)</td>
</tr>
<tr>
<td></td>
<td>• Design physical database (physical structure, fields, records, and physical allocation of the files)</td>
</tr>
<tr>
<td></td>
<td>• Conduct performance analysis on critical components</td>
</tr>
<tr>
<td></td>
<td>• Revalidate software architecture against</td>
</tr>
<tr>
<td>TIME BOX ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>operational concepts</td>
</tr>
<tr>
<td></td>
<td>• Revalidate software architecture against software requirements</td>
</tr>
<tr>
<td></td>
<td>• Continue to identify/Communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Capture all design components in architecture and software</td>
</tr>
<tr>
<td></td>
<td>software development folders</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box</td>
</tr>
<tr>
<td>Code (Micro Level</td>
<td>• Integrate COTS, GOTS and Reuse items</td>
</tr>
<tr>
<td>Time Box)</td>
<td>• Create and update source code</td>
</tr>
<tr>
<td></td>
<td>• Populate database and update database design and data dictionary</td>
</tr>
<tr>
<td></td>
<td>• Continue to identify/communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box</td>
</tr>
<tr>
<td>Debug (Micro Level</td>
<td>• Conduct continuous peer reviews within product teams</td>
</tr>
<tr>
<td>Time Box)</td>
<td>• Conduct continuous peer internal unit testing</td>
</tr>
<tr>
<td></td>
<td>• Conduct continuous integration drops</td>
</tr>
<tr>
<td>TIME BOX ACTIVITY (Micro Level Time Box)</td>
<td>TASK</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------</td>
</tr>
</tbody>
</table>
| Test                                   | • Prepare unit tests  
• Conduct unit tests  
• Capture all design components in architecture and software development folders  
• Continue to identify/communicate risks to risk management system  
• Continue demonstrating products and services via time box demonstrations |
| Integration (Micro Level Time Box)    | • Verify test environment reflects operational environment as close as possible  
• Install, populate and maintain verification database  
• Continue developing test procedures  
• Generate input data as required  
• Continue obtaining hardware and software required to conduct testing and integration activities |
<table>
<thead>
<tr>
<th>TIME BOX ACTIVITY</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Verify requirements on a continuous basis</td>
</tr>
<tr>
<td></td>
<td>• Generate problem reports as soon as discrepancies are identified</td>
</tr>
<tr>
<td></td>
<td>• Capture all integration results in architecture and software development folders</td>
</tr>
<tr>
<td></td>
<td>• Continue to identify/communicate risks to risk management system</td>
</tr>
<tr>
<td></td>
<td>• Continue demonstrating products and services via time box demonstrations</td>
</tr>
</tbody>
</table>

**Section 7.7 Evaluation Time Box Activities**

The following table lists a framework for activities to be executed during the Evaluation Time Box Activity. The list of activities should be tailored to meet the needs of individual programs.
Figure 7-14 Evolutionary Time Box

Table 7-3 Evolutionary Time Box Activities

<table>
<thead>
<tr>
<th>ACTIVITY</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>• Execute test procedures using prototype and</td>
</tr>
<tr>
<td>(Macro Level Time Box)</td>
<td>crystallized software deliverable product</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>TASK</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>• Evaluate by execution, not documentation</td>
</tr>
<tr>
<td></td>
<td>• Provide detailed feedback to development team on problems using problem reports</td>
</tr>
<tr>
<td></td>
<td>• Development team ensures traceability and verification of requirements database</td>
</tr>
<tr>
<td></td>
<td>• Customer and engineering team planning for Build n + 1</td>
</tr>
</tbody>
</table>
Section 8 A Lean Software Engineering System Example

The following discussion represents an example of the Lean Software Engineering System tailored to meet the needs of a Department of Defense customer. The model has been developed to address the following problems that have evolved over two years. The problems identified on the program are all potentially solvable by implementation of the lean paradigm. The transition for this particular organization will require a major culture change and strong leadership from senior management. The model and lean recommendations represent a first step in that direction.

This case study examines a large-scale complex software development effort for the Department of Defense. The project began in the fall of 1996 when the customer approved for the next generation of a command and control software system to be developed and deployed to support future digitization efforts of the Department of Defense. This system will be referred to as Project J and System M for purpose of anonymity of the engineering organization and Department of Defense customer.

System M has a mission to automate the decision support information system that supports the tactical commander and supporting operation team during war and peace time operations. The two major functions of the system are command
and control for the maneuver of battlefield functional areas by planning, coordinating and controlling ground battle, and providing automated command and control capability for armor, infantry, aviation, engineer, Military Police (MP), Special Operation Forces (SOF), and chemical operations. The second major functional capability of System M is control of the force level information. Control of force level information requires capabilities for integrating maneuver battlefield functional areas with other battlefield functional areas and providing command and control for the commander and supporting staff at and between echelons.

To date, the development effort has not gone well in terms of the three goals of system development: cost, schedule, and technical success. The program is delinquent in all three areas and the customer has requested a new streamlined approach to support mission requirements. The project development team is approximately 150 people including all technical, management and support personnel. The project advertises using all the latest and best-practice methodologies and tools. These methodologies and tools include spiral development, object oriented design and analysis, Rational Rose\textsuperscript{63} Modeling Tool, Requirements and Traceability Management (RTM)\textsuperscript{64} Tool, Integrated Product teams (IPT's), - risk mitigation approaches, and so on, but as we review the project, the need for a paradigm shift to lean is be evident. The current system development process is defined as a spiral process. In reality, the project is employing a modified waterfall approach with mini-projects leading to a final

\textsuperscript{63} Rational Corporation, www.rational.com
\textsuperscript{64} I-Concept Corporation, www.iconcept.com
product. The project is struggling with this dysfunctional understanding of what is thought (Spiral) and what is reality (Waterfall). This confusion and lack of direction coupled with the problems identified below set the stage for implementation of the lean software engineering system as a potential solution that will meet the goals of the customer for Project J. A review of the project uncovered that major elements of the Spiral model are not be used. There is no risk management system for technical problems. No system level integration occurred until the sub-system segments were complete at the end on code and unit test. No lessons learned data is incorporated into the development process. A further analysis of the Object Oriented Model Design verified against the configuration management “controlled” version of the code reveals different design characteristics. A third verification against the requirement traceability tool shows further discrepancies of the planned design versus the delivered design.
Section 8.1 Case Study Problem Identification

The major problem and challenge areas identified for Project J is as follows:

- Program development strategy states a spiral development process but a waterfall process was really occurring
- Program stated integrated product teams is the structure but a pseudo functional matrix organization exists
- The program is experiencing high turnover rates (55% for 1993) and approximately 90% since October 1996.
• The appraisal and reward system is individual based and not team oriented
• No real risk management system exists
• No accountability for following the defined process is occurring
• Very little customer interface at the “working level”
• Long process cycles occurring for each process kernel
• Lack of training on sophisticated CASE tools
• Development focus on documentation versus products and services
• Lack of integrated knowledge center for exchange and sharing of information
• Lack of adherence to defined system architecture and associated models (No top-down approach)
• Lack of acknowledgment for defined interfaces (Bottom-up engineering)
• Lack of accountability by individual integrated product teams on the system view
• Many bottlenecks occurring in process (Quality Assurance, Configuration management, and customer response to changes)
• Batch and queue effects occurring in wasted down time by some IPT's while waiting for all IPT's to complete process kernel
• Overall lack of communication within and between teams
• Poor planning on scope of work and associated goals
• Lack of integration of products on a continual basis
Section 8.2 Case Study: The Lean Solution

The following is a summary of the lean initiative for the System M Development model and control mechanisms that will be implemented by the engineering team for and during the development of the functionality of the stated requirements.

The System J development model is a tailored system, incorporating rapid development and prototype methodologies mapped to an evolutionary spiral process. Figure 8-2 presents a high level process view of the differing integrated development methodologies/processes working together to produce a test acceptable software product for each build. The Controlling Body for the definition and authorization of the methodology is the Technical Analysis Working Group (TAWG) and the Design Analysis Working Group (DAWG) is the Controlling Body for the implementation of the processes and Work Packages (WP). The four development processes depicted in the figure are listed below and each is summarized later in this Section.

- Outboard Rapid Prototype Methodology (ORPM)
- In line Rapid Prototype Methodology (IRPM)
- Rapid Development Methodology (RDM)
- Hardening Methodology (HM)

a) Controlling Bodies
1. The Task Analysis Working Group (TAWG)

The TAWG reviews and approves Work Packages (WP's) that identify the allocated requirements from the System Segment Specification (SSS) to specific builds and defines the development process to be utilized for requirement satisfaction. The TAWG is chaired by the government and comprised of government and contractor representatives. The government chair has sole authority for the decisions levied by the TAWG. Furthermore, the TAWG is the only System M technical body that has the authority to allocate, re-allocate, not allocate or delete individual requirements from a build or version. The TAWG shall approve a build plan that is the summation of all the SSS allocated and approved for a specific build within a version.

2. Design Analysis Working Group (DAWG)

Unless otherwise directed by the TAWG, the DAWG has oversight responsibility and approval authority of the technical output and results of implementation with respect to TAWG authorized WP’s. The DAWG is chaired by the contractor and comprised of government and contractor representatives. The Government representative must approve any DAWG modifications to the derived requirements (i.e. virtual lean documentation). The DAWG will authorize, within a given WP, progression from process kernel to process kernel based upon their exit criteria and results. The DAWG’s authority is limited to the scope of work defined within a given WP and to the interim Control Gates within the Prototype and Rapid Development Process.
b) Development Methodologies and Processes

1. Outboard Rapid Prototype Methodology (ORPM)

ORPM activity is utilized to mitigate version risk. The ORPM is intended for future build requirements and functionality determined “high risk” due to requirement uncertainty, architecture or technology challenges. Functional Capability List (FCL) is substituted for formal derived requirements. No formal testing or code walk through is conducted unless the ORPM effort is deemed successful and the TAWG directs the results of the ORPM to be incorporated into the current Build. TAWG acceptance and approval of the demonstration results (i.e., exit criteria) is the Control Gate for the prototype process.

2. The In line Rapid Prototype Methodology (IRPM)

IRPM is used for current build requirements deemed “high risk”. The same approach, criteria and analysis as described in ORPM are required but the peer review and code walk through is required. In addition, structured testing using the code test tool set is required during IRPM if the software is going to be integrated into the current build. This ensures stable software is introduced into the system base line for testing and integration. A pre-integration environment will be utilized for testing of work package development software throughout the IRPM life cycle.
3. The Rapid Development Methodology (RDM)

RDM is utilized for current build requirements defined, understood and quantified as “low risk” by the TAWG. The Preliminary System Analysis (PSA) results along with the Work Packages (WP) will serve as initial inputs for system level design and architecture data. The RDM utilizes a streamlined five-kernel process. The first kernel is an optional process (as designated in the WP) to further define the SSS, analyze reuse candidates, assess COTS products, or other subject areas as appropriate. The white paper will capture the results of the analysis. This white paper will be reviewed by the DAWG. The SSS requirement derivation process will result in an FCL approved by the DAWG in a FCL Requirements Review (FRR). The software design is also approved by the DAWG during a Detailed Design Review (DDR). Peer reviews and a code walk through is required for all RDM work packages. In addition, structured testing using the Code test tool set is required during RDM. A pre-integration environment is used for testing of work package development software throughout the RDM life cycle.

4. The Hardening Methodology (HM)

HM is the process of formally deriving the FCL’s developed during ORPM, IRPM, and RDM. All previously developed work packages will go through the HM when the system level functionality is deemed “fully achieved” by the customer during testing of the previous build. The hardening process allows the contractor to achieve the following:
a) Approval from the customer office on SSS and Derived Requirement satisfaction

Optimization of design as required Optimization: of code as required

Updates to the Software Development Folder (SDF) and Architecture Development Folder (ADF) as required

Populate the Design Blueprint with appropriate data to a sustainable and maintainable level for the customer to support the System M system. The HM process provides interim Control Gates to review and approve (TAWG level approval) the maturation of the SSS and it's associated design. The entry criteria of HM is the approved exit criteria developed for ORPM, IRPM and/or RDM along with the Preliminary System Analysis (PSA) for the upcoming Build/Version.

The HM process matures the requirements and design up to a fully compliant set of documentation and design as delineated via the hard copy CDRLs and their corresponding DD1423 Forms and Data Item Descriptions (DIDs/Dis). HM has within it the following Control Gates to monitor and approve the requirements, design and implementation maturity progression:

- TAWG RR - Requirements Review
- DAWG DR - Design Review
- DAWG CWTH - Code Walk-through
The HM is the final process for "closing out" System level requirements from future build plans and future required work for the contractor to implement in future PSA and TAWG analysis efforts.
This high level model is also decomposed to reflect the detailed needs of the project.

Each of the four development tracks is further depicted in the following figures.

**Figure 8-2 System J Lean Development Model**
Figure 8-3 Project J Build Cycle

Initial Reqs. → Mission & System Requirements Analysis

System Design

Test Planning

Begin Cycle Build N

Work Package Devel.

PRs from field and build tests

ORPM

IRPM

RDM

HM

Problem Correction

Build Integration

TAWG

TRR

PRs & patches deferred to next build

PRs resolved by TAWG for resolution

Repeat Cycle Build N+1

TRR (Build)

SSAT (Vers.)

PPT (Build)

Tailored PPT

ABCS Integration

ABCS Integration (NJ)

Product Delivery

Tailored Product

Legend:

TAWG Oversight

109
Figure 8-4 Outboard Rapid Prototype Track

Project J
Outboard Rapid Prototyping Track

Build $N$


Demo Checklist

(or)

System Design

Mission & System Analysis

DAWG Control

Build $N+1$

*ORPM = Outboard Rapid Prototyping Methodology
Figure 8-5 In-Line Rapid Prototype Track

Project J
In-Line Rapid Prototyping Track

IRPM*

Mission Analysis RP → Rqts. Analysis RP → Design IRP → Code IRP → Demo

Peer Review → Code Walkthru

Pre-Integration

DAWG Control

Build N

*IRPM = In-line Rapid Prototyping Methodology
Figure 8-6 Rapid Development Track

Project J
Rapid Development Track
-3-

Build N

RDM*


Pre-Integration

RTM Capture for Build

Peer Review ➔ Code Walkthru

DAWG Control

*RDM = Rapid Development Methodology
Section 8.3 Project J Audit Results

This software-engineering framework is established to address the previously stated cost, schedule, and technical challenges. The System J project has experienced significant schedule, cost and technical problems since the project began development. To date, the program has had two software build deliveries. The first resulted in 15% successful requirement achievement of system level requirements. The second build delivery resulted in 55% requirement success of
system level requirements after testing. The first build was originally planned for June 1997 and was delivered in September 1997. The second build was originally planned for May 1998 and was delivered in December 1998.

Figure 8-8 Build Delivery Schedule Results in Calendar Months

<table>
<thead>
<tr>
<th></th>
<th>Build 1</th>
<th>Build 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planned</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Actual</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Desired</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
The customer has now imposed a six-month development window on the project. The customer has new operational support requirements that require each software build be available every six months. The customer requested that a new streamlined development process be conceived to address the new time constraint and the past challenges the project has been experiencing. This new streamlined system is presented in Section 8-2. A process audit was conducted on the engineering team for Project J. The survey was conducted on six software product integrated product teams, the system engineering IPT, integrated logistics support IPT, and integration and test IPT. No audit was conducted on the management team for this analysis. The total possible score for the survey is 100 points. The total project average is 69.7 points. The
software IPT's average score is 68 points and the system engineering, integrated logistics support, and integration and test IPT's average score is 73 points.

**Figure 8-10 Project J Process Audit Results**

The survey addressed organization, current assignment, and process related items. The survey highlighted some critical findings listed below:

**Organizational:**
- Confusing work direction due to multiple managers with conflicting direction
- Unclear or unknown about the IPT mission charter
- None or very little knowledge of program master schedule
- No knowledge of risk management system

**Current Assignment:**
- Direction usually given in verbal or email format
• Almost no written direction of work assignments

Process:

• Uncertainty as to location of critical project data supporting their work
• Use of "Non-approved" process procedures
• Inability to discuss his or her specific process area
• Unable to discuss or identify his or her process kernel procedures
• Very little knowledge about change/control process
Section 9 Conclusions

Software engineering productivity is a critical element in both the Department of Defense and commercial sectors. This thesis presents shifting to lean principles to improve cycle time reduction and overall quality in the software development process. A lean software engineering solution is presented to address the three goals of software development: cost, schedule, and technical performance. The foundation of the lean software engineering model is value, teamwork, communication, efficient use of resources, elimination of waste, and continuous improvement.

A lean software-engineering model is presented with detailed analysis on the key characteristics highlighted and supported with Department of Defense, commercial, and academic software research activities. The research has shown success in reuse driven initiatives, prototyping activities, and a movement toward product line solutions. Alternative lean themes such as time boxing, virtual documentation, streamline management and control, and iteration in the development process are presented and supported through empirical research and case study analysis.

Lean principles are not hard to understand, but it is difficult to start the change process. The existing infrastructure is still in place and incentives to enable the
lean change need to be established. The current state of defense budgets and the requirement to keep pace with changing technologies and customer needs is further reason to consider the lean shift. Software engineering can benefit from lean thinking. The lean shift together with effective engineering and management practices can evolve the industry into a robust and customer oriented system.
Bibliography


Department of the Army: Reuse Technology Assessment Report for the Department of the Army, 1996.


Karat, John and Dayton, Tom; *Practical Education for Improving Software Usability*, Association for Computing Machinery, 1995.


Womack, James P. And Jones, Daniel T.; Lean Thinking, Simon and Shuster, 1996.