Identifying Lean Practices for Deriving Software Requirements

By

Brian J. Ippolito

B.S. Aerospace Engineering
Illinois Institute of Technology, 1992

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in
ENGINEERING AND MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2000

© 2000 Massachusetts Institute of Technology
All rights reserved

Signature of Author
Brian J. Ippolito
System Design and Management
December 22, 1999

Certified by
Earll Murman
Ford Professor of Engineering
Department of Aeronautics & Astronautics

Accepted by
Thomas A. Kochan
LFM/SDM Co-Director
George M. Bunker Professor of Management

Accepted by
Paul A. Lagace
LFM/SDM Co-Director
Professor of Aeronautics & Astronautics and Engineering Systems
Identifying Lean Practices for Deriving Software Requirements
by
Brian Ippolito

Submitted to the System Design and Management Program
on 22 December in Partial Fulfillment of the
Requirements for the Degree of Master of Science in Engineering and Management

Abstract

Lean principles focus on employing value added activities to reduce product development cycle
time, increase quality, and reduce cost. Lean originated in the automotive industry and has since
been centered in the manufacturing domain. Lessons learned on implementing Lean initiatives
have been captured by the in the Lean Aerospace Initiative (LAI) and incorporated into the Lean
Enterprise Model (LEM) (http://lean.mit.edu/public/index.html). To the author’s knowledge,
this is the first research effort specifically designed to apply the Lean principles and the Lean
Enterprise Model to the aerospace software requirement derivation process.

Data supporting this research is the result of a comprehensive two-year research effort involving
three detailed case studies with 45 case study interviews, 125 stakeholder surveys collected from
ten aerospace software upgrades, feedback from numerous aerospace industry practitioners and
Massachusetts Institute of Technology (MIT) faculty.

Ten aerospace software upgrades were analyzed at both an enterprise level and an organizational
level to identify the presence of Lean practices. At the enterprise level, metrics typically used to
measure enterprise performance (Flow Time, Stakeholder Satisfaction, Quality Yield, and
Resource Utilization) were found to be appropriate for the software requirement process but not
adequately implemented. An organizational analysis observed five of the twelve Lean practices
as effectively implemented and identified opportunities to implement four more Lean practices.
Acknowledgments

Numerous individuals and organizations supported this research. I would like to thank Boeing, Lockheed Martin, Aerojet, Rockwell Collins, Allied Signal, Raytheon, Massachusetts Institute of Technology, the United States Air Force, and the consortium members of the Lean Aerospace Initiative for their time and support of the research effort. The individual contributions of Colonel Jeff Sogard, Lieutenant Colonel Mark Fish, Major Fred Gregory, Catherine Nobe, Andy C. Lay, Norton “Ki” Scofield, Kathleen Ward Yde, Dr. Eric Rebentisch, Professor Stanley Weiss, Professor Nancy Leveson, and Professor John Deyst Jr. are greatly appreciated and have helped make this research effort a success. Thank you to all of my colleagues in the System, Design and Management program and my friends for provided support over the two years, especially Jennifer Hamelin for spending hours supporting the thesis reviews.

Thank you to Darleen Druyun, Judy Stokley, Colonel Gary Smith (ret.) and Lieutenant Colonel Tom White (ret.) for providing me opportunities to successfully lead programs and learn the fundamentals of aerospace product development.

Special thanks to Karen Tobin, my mother and father, Alice and Anthony Ippolito and my sisters Jennifer and Nicole. All have been with me to enjoy the high and low points of the MIT experience and I am forever grateful for their unconditional support and love.

Finally, thank you to my advisor Professor Earll Murman. I could not imagine anyone having a better advisor and mentor. This research would not have been a success had it not been for his countless hours of support and encouragement.
Table of Contents

Chapter 1: Executive Summary ........................................................................................................ 11

Chapter 2: Introduction .................................................................................................................. 17

2.1 Importance of the Requirements Phase .................................................................................... 17

2.2 Sources of Information on Aerospace Software Requirement Derivation ......................... 21

2.2.1 Software Technology Support Center (Hill Air Force Base, Utah) ................................ 21

2.2.2 Carnegie Mellon University (CMU) Software Engineering Institute (SEI) ....................... 21

2.2.4 Military Standard 2167A (DOD-STD-2167A, 1988) ......................................................... 24

2.2.5 Institute of Electrical and Electronics Engineers, Inc. (IEEE) ......................................... 24

2.2.6 Requirements and Technology Concepts for Aviation (RTCA) Software Considerations in the Airborne Systems and Equipment Certification [DO-178B] ................. 24

2.3 Software Development Processes ............................................................................................ 25

2.3.1 Waterfall Software Development Model ............................................................................ 26

2.3.2 Incremental Software Development Model ........................................................................... 27

2.3.3 Spiral Software Development Model .................................................................................... 27

2.4 Requirement Derivation Process ............................................................................................. 28

2.5 Lean Aerospace Initiative (LAI) ............................................................................................... 30

2.6 Research Focus ......................................................................................................................... 34

2.7 Summary ................................................................................................................................ 34

Chapter 3: Research Method ........................................................................................................ 35

3.1 Research Design ....................................................................................................................... 35

3.1.1 Baseline End to End Cycle Time ....................................................................................... 36

3.1.2 Study Successful Programs .................................................................................................. 36

3.1.3 Capture Program/Process Leader’s Knowledge ................................................................. 37

3.1.4 Adopt a Value Stream Approach ....................................................................................... 38

3.1.5 Utilize the Lean Enterprise Model (LEM) ......................................................................... 38

3.2 Stakeholder Survey .................................................................................................................... 38

3.3 Case Study ................................................................................................................................ 41

3.4 Interim Testing of Results ........................................................................................................ 42

3.5 “Real-time” Implementation of Research Findings ................................................................. 42

3.6 Research Method Summary ..................................................................................................... 43

Chapter 4: Case Study Analysis .................................................................................................. 45

4.1 Commercial Auto-pilot Software Upgrade Case Study ............................................................. 45

4.1.1 Commercial Auto-pilot Software Upgrade Value Stream .................................................... 46

4.1.2 Commercial Auto-pilot Software Development Environment .............................................. 47
List of Figures

Figure 2.1: Cost Committed, Cash Flow and the Ability to Influence at Different Stages of the Product Life Cycle [Fredriksson, 1994] ................................................................. 18

Figure 2.2: Cumulative Impact of Errors [Davis, 1990] .................................................................................. 19

Figure 2.3: Cost (Effort) to Repair Software in Relationship to Life Cycle Stage [Davis, 1990] ............................................................................................................... 20

Figure 2.4: Classic Waterfall Model Depicted by Brooks [Brooks, 1995] .................................................... 26

Figure 2.5: Spiral Model Software Process [Boehm, 1988] ............................................................................. 28

Figure 2.6: Software Requirements Derivation Process [Mil-Std-2167A, 1988] ........................................... 29

Figure 2.7: Lean Enterprise Model Architecture and Overarching Practices .............................................. 31

Figure 3.1: Illustrated Relationship between Case Study Data and Survey Data ..................................... 39

Figure 4.1: Commercial Auto-pilot Software Upgrade Value Stream .......................................................... 46

Figure 4.2: Commercial Auto-pilot Case Study Sources of Requirements Input [Agate, 1998] ....................... 50

Figure 4.3: Commercial Auto-pilot Software Upgrade Major Software Development Cycle Time Activities ............................................................................................................. 50

Figure 4.4: Commercial Auto-pilot Software Upgrade Requirement Derivation Process ......................... 51

Figure 4.5: Military Avionics Software Upgrade Value Stream ......................................................................... 53

Figure 4.6: Military Avionics Case Study Sources of Requirement Input ....................................................... 57

Figure 4.7: Military Avionics Software Upgrade Major Software Development Cycle Time Activities ........ 58

Figure 4.8: Military Avionics Software Upgrade Requirement Derivation Process ..................................... 59

Figure 4.9: Military Space Ground Terminal Software Upgrade Value Stream ............................................. 62

Figure 4.10: Military Space Ground Terminal Case Study Sources of Requirement Input ........................................... 65

Figure 4.11: Military Space Ground Terminal Software Upgrade Major Software Development Cycle Time Activities ............................................................................................................. 66

Figure 4.12: Military Space Ground Terminal Software Upgrade Requirement Derivation Process ................ 66

Figure 5.1: Lean Enterprise Metrics and their Relationship to Software Requirements .................................. 71

Figure 5.2: Defining the Customer ................................................................................................................... 72

Figure 5.3: Defining the End User ................................................................................................................... 74

Figure 5.4: Estimated End User and Customer Satisfaction ........................................................................... 75

Figure 5.5: Software Development Cycle Time Responses ............................................................................. 77
Figure 5.6: Estimated Percentage of Total Software Development Cost Associated with Software Requirement Rework ................................................................. 79
Figure 5.7: Estimated Percentage of Unplanned Derived Software Requirement Changes 80
Figure 5.8: Involvement of Contractor Leadership in Concept Definition and Software Requirement Analysis Activities versus the Percentage of Unplanned Rework 81
Figure 5.9: Typical Cost Distribution Across the Value Stream for the Military Avionics Software Upgrade [Gregory, 1998] ................................................................. 82
Figure 5.10: Cost Breakdown for Case 2 Aircraft Developer [Gregory, 1998] ................... 83
Figure 5.11: Lean Enterprise Model (LEM) Overarching Practices: Effectively Implemented..................................................................................................... 85
Figure 5.12: Case Studies Level of Integration............................................................. 87
Figure 5.13: Estimated Value of Each Software Development Phase........................... 88
Figure 5.14: Application Domain Performance versus the Level of Value Added ............ 89
Figure 5.15: Average Level of Importance of Various Aspects of End User Involvement ... 91
Figure 5.16: Average Level of Importance of Various Aspects of Customer Involvement .. 92
Figure 5.17: Process Leadership's Perception of the Importance of End User Involvement in Building Trust .................................................................................. 95
Figure 5.18: Average Estimated Level of End User Involvement in the Requirement Derivation Process............................................................................................ 96
Figure 5.19: Frequency of Unplanned Changes................................................................ 97
Figure 5.20: Military Space Ground Terminal Requirement Prototyping Process .......... 99
Figure 5.21: LEM Overarching Practices: Opportunities to Improve ............................. 101
Figure 5.22: Percentage of Survey Respondents that Could Estimate Cost and Schedule 103
Figure 5.23: Estimated Level of Formal Training Effectiveness ...................................... 105
Figure 5.24: Percentage of NA Responses for Various Forms of Requirement Training ... 106
Figure 5.25: Primary Responsibility for Cost and Performance ...................................... 107
Figure 5.26: Person(s) Primarily Responsible for Making the Requirements Complete and Traceable................................................................................................. 108
Figure 5.27: LEM Overarching Practices – Future Improvements ................................. 109
Chapter 1: Executive Summary

The Lean Aerospace Initiative (LAI) is a government, industry, labor, and academic consortium focused on incorporating the principles of Lean in the aerospace industry. Lean originated in the automotive industry and is grounded in the manufacturing domain. To the author’s knowledge, this is the first research effort specifically designed to apply the Lean principles and the Lean Enterprise Model to the aerospace software requirement derivation process. The data presented is the result of a comprehensive two-year research effort involving three detailed case studies with 45 case study interviews, 125 stakeholder surveys collected from ten aerospace systems, feedback from numerous aerospace industry practitioners, and MIT faculty.

The basic principles of Lean are focused on employing value added activities to reduce product development cycle time, increase quality, and reduce cost. The Lean Enterprise Model (LEM) (http://lean.mit.edu/public/index.html) developed by LAI is a systematic framework incorporating data and lessons learned of aerospace industry, academia, and government in support of Lean initiatives. This research utilized the LEM framework to identify the presence of Lean practices in the process of deriving software requirements on real time mission critical aerospace systems. LEM enterprise level metrics (Flow Time, Stakeholder Satisfaction, Quality Yield, and Resource Utilization) were used to develop process outcome measures. An organizational level analysis was also completed using the twelve Overarching Practices. The Overarching Practices are found in the LEM and were used as a guide for analyzing the presence of effective Lean practices for deriving software requirements.

The goal of this research is to identify value added practices using the Lean framework to improve the software requirements process. LAI industry and government consortium members selected the systems involved in the research effort. Each system is believed to be one system that was successful in deriving software requirements. Four application domains (commercial aircraft, military aircraft, missile/munitions, and a military space ground terminal) are represented in the ten software upgrades used in this research.
The process used to derive software requirements was selected as the research focus area based on feedback from the LAI consortium members. Aerospace product development practitioners have repeatedly expressed frustration over the difficulties associated with software and requirements. Deriving the software requirements is an important step in developing aerospace products. Requirement derivation activities occur early in the product development process and can have significant impacts on the cost, schedule and performance of the system [Fredriksson, 94]. The cost to correct errors made in the beginning grows exponentially the longer they go undetected [Davis, 90]. Analyzing the process using a Lean framework allows practitioners the opportunity to improve and avoid such adverse system impacts.

A structured approach based on the LEM was designed to integrate data from 125 stakeholder surveys and three case studies. Surveys were targeted at the program and process leadership (program manager, chief systems engineer, software developers, etc.). Lean principles were incorporated in designing the survey, which was pre-tested by a group of practitioners for completeness and understanding.

Several major observations were found as a result of this research. At the enterprise level, the metrics typically associated with manufacturing performance (Flow Time, Stakeholder Satisfaction, Quality Yield, and Resource Utilization) should apply to software requirement derivation. Despite the fact that only successful programs were used, the case studies failed to identify effective enterprise metrics for requirements. None appear to be used for the software development cycle time, customer satisfaction, end user satisfaction. Quality yield metrics associated with the rework of requirements found significant room for improvement. On average, 23% of the software requirements required unplanned rework at an average cost of 16% of the total software development cost.

Existing resource utilization measures are focused on productivity associated with generation of lines of code. Case study data indicates that this may not be the most appropriate measure for aerospace software upgrades. Coding appears be a small fraction of the total program development cost (less than 6% of the total). This data were supported by conducting a high level program value stream analysis to identify the elements of the system that MUST be
delivered in order for the capability to be used by the person operating the system, the end user. For example, the military avionics software upgrade case study requires changes not only to the avionics software code but changes to technical orders, support equipment, aircraft sensors and weapons. In addition, weapons and tactics trainers and government certification all required updates prior to fielding the capability. The presence of such a large number of value stream elements and the relative small cost of changing the avionics software suggests software upgrade programs should be thought of as much more than just code. A metric which measures total program cost based upon lines of code is not very meaningful.

An analysis of the processes at an organizational level observed five Lean practices as being effectively implemented in the program studies. The military space ground terminal software upgrade provided two examples of effectively implemented Lean practices, Implement Integrated Product and Process Development and Assure Seamless Information Flow. First, the software development team had a highly integrated organization with subcontractors collocated with the software developers to support the software development effort. Second, the team developed a continuous requirement analysis process that stabilized technical and non-technical risks associated with software requirements. This process allowed the military space ground terminal team to rapidly develop software to meet the needs of the end users.

The military avionics software case study illustrated how to effectively Maximize Stability in a Changing Environment by using information technology tools to assure requirement information flows to system stakeholders. End user feedback on requirements was captured electronically and automated to link between requirement documents and program documentation.

The stakeholder survey highlighted a fourth Lean practice as effectively implemented, Develop Relationships Based on Mutual Trust and Commitment. Survey data indicated the requirement derivation process can be an effective way to build trust between the end user and the software developers. The survey respondents considered this a very important aspect of end user involvement in the requirement derivation process.
A fifth Lean practice, *Maintain Challenge of Existing Process*, was observed to be effectively implemented on all three case studies. Each organization appeared committed to challenging the requirement process and all had process improvement initiatives underway to incorporate Lean. The research data was utilized by each organization to initiate software requirement derivation process improvements.

In addition to observing Lean practices that were effectively implemented, the research identified four Lean practices process participants could work to improve. The first Lean practice to improve, *Continuously Focus on the Customer*, was the result of survey data that identified process participants as having multiple definitions for the term “customer”. Case study interviews suggested that disconnects could exist between the needs of multiple customers versus the needs of the end user.

Data supporting another Lean practice, *Identify and Optimize Flow*, revealed the inability of the survey respondents to identify the distribution of cost and schedule by process development phase. Survey data suggests the process leadership does not have a clear understanding of when activities start and finish and the relationship of these activities to total program cost and schedule. Optimizing the process is difficult without existing measures of cost and schedule.

A third Lean practice to improve, *Optimize Capability and Utilization of People*, identified formal training as an area to improve. Survey data identified formal training as ineffective in helping requirement process practitioners perform their job. Currently On-the-Job (OJT) training was considered the only effective method of requirement training.

Survey data associated with a fourth Lean practice, *Make decisions at the Lowest Possible Level*, revealed that the person or organization with the primary responsibility for making sure requirements are complete, traceable, and meet the end users needs is unclear to process participants. The program manager was identified as having the primary responsibility for making sure requirements are cost effective but other areas show no clear single responsibility.
Three Lean practices were not observed during this research (*Ensure Process Capability Maturation, Nurture a Learning Environment, and Promote Lean Leadership at all Levels*). More research is needed to characterize the impact of these Lean practices on the process. Each requires an established level of understanding of Lean that was not present at the time the research was conducted.

The feedback from process practitioners, industry, government, and the MIT faculty has been critical to this research effort. The support from the aerospace community has helped to interpret data, highlight effective Lean practices, and identify opportunities for future improvements. This research is intended to build on previous research and lay the foundation for incorporating Lean into software development.
Chapter 2: Introduction

Almost all products are built to meet performance specifications. The requirements embedded in the specification are used to guide product developers during the development process. Requirements generally define the needs of stakeholders (customers, operators, maintainers, engineers, etc.) and are traditionally captured during the early stages of the product development efforts and translated to engineering requirements. Product development practitioners have found that generating requirements can be very challenging. This thesis examines Lean practices associated with the process used to derive software requirements for aerospace software systems.

Lean originated in the automotive industry and has been traditionally grounded in the manufacturing domain [Womack, Jones, and Roos, 90]. The concepts of Lean are centered around eliminating waste and non-value added activities in order to shorten the product cycle time, increase quality, and increase stakeholder satisfaction. These concepts would appear to be applicable to any type of industry. Massachusetts Institute of Technology, aerospace companies, and the government formed a consortium, the Lean Aerospace Initiative (LAI), focused on incorporating Lean in the aerospace industry. This research supports this effort as it identifies Lean practices at both the enterprise and organizational level.

Research data for the thesis were collected from upgrades to real time mission critical embedded aerospace software systems. The research focused on the multiple elements of the requirement process and the incorporation of Lean principles into the process. All of the systems providing data were part of commercial aircraft, military fighter and bomber aircraft, missile and munitions and space ground terminal programs.

2.1 Importance of the Requirements Phase

Researchers have often highlighted poor requirement definition as a major contributor to development program cost and schedule overruns. Software developers and process practitioners repeatedly cite inadequate or changing requirements as a source of frustration. The software and requirement literature is filled with warnings against requirement growth and process failures [Robertson, 1999].
"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all of the interfaces to people, to machines, and to other software systems." Frederick Brooks [Brooks, 1995]

Deriving requirements from stakeholders can be difficult. The requirement derivation process is one way to facilitate the transfer of knowledge from the stakeholders to the software developers. The process often crosses software and systems engineering disciplines and includes the senior program leadership. A requirement communicates the desired stakeholder functions to the engineering community for implementation into the product. As the role of software grows in aerospace products, the process to derive software requirements becomes increasingly important.

The requirement phase of the product development cycle has an impact beyond just defining how the product is expected to perform. Early stages of the product development effort have the highest probability of influencing the performance, the cost, and the time necessary to develop the product. Figure 2.1 illustrates the impact the initial stages of the product development effort can have on the product by comparing committed costs, cash flow and the ability to influence the product. The chart is notional and is not comprised of data but indicates the early stages of the development effort have a strong influence over the product and high percentage of product cost is committed. Figure 2.1 also suggests this stage in the process is completed at a relatively low cost.

**Figure 2.1: Cost Committed, Cash Flow and the Ability to Influence at Different Stages of the Product Life Cycle [Fredriksson, 1994].**
The notional lines highlighted in Figure 2.1 illustrate that the conception phase, during which the requirement derivation activities are occurring, is not only a significant determinant in the future cost of the product, but it comes at a relatively small fraction of the total cost.

The outcome from the early stages of the product development process is a set of system requirements from which the software and other sub-system requirements will be derived. During this phase the developer identifies customer needs, establishes target requirements and selects concepts to be implemented [Ulrich and Eppinger, 1995].

Mistakes made during the requirement derivation process may not be easily recognized. As the product development process proceeds, errors in the requirement specification cascade through the entire process. Additional errors in the design and implementation of the design further compound the problems associated with erroneous specifications (Figure 2.2). These errors can be difficult to detect early in the development process and are often not noticed until the product is in the final stages of testing when they impact cost, schedule, and performance.

Figure 2.2: Cumulative Impact of Errors [Davis, 1990]
Unfortunately the cost of error correction changes over time. Problems found later in the development are of an order of magnitude more expensive to fix than problems found earlier. By arbitrarily assigning a unit cost to the effort required to fix a requirement error during coding, researchers have compared the costs of fixing the requirement problems at various stages in the software development process. Figure 2.3 illustrates the estimated cost (effort) to repair software based upon when the error is found.

**Figure 2.3: Cost (Effort) to Repair Software in Relationship to Life Cycle Stage**

[Davis, 1990]

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Cost to Repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>0.2</td>
</tr>
<tr>
<td>Design</td>
<td>0.5</td>
</tr>
<tr>
<td>Coding</td>
<td>1.0</td>
</tr>
<tr>
<td>Unit testing</td>
<td>2.0</td>
</tr>
<tr>
<td>Acceptance</td>
<td>5.0</td>
</tr>
<tr>
<td>Maintenance</td>
<td>20.0</td>
</tr>
</tbody>
</table>

The data used to establish the relationship was based on three independent requirement studies. It may be dated and biased by a particular application domain, but it illustrates the relative relationship between the requirement errors, the development process, and the cost to repair defects. Davis notes that fixing requirements late in the life-cycle process was found to be up to 100 times more expensive than getting it correct during the requirements phase [Davis, 1990]. The long-term impact of deriving incorrect requirements is one of the major drivers behind seeking process improvements such as understanding and implementing Lean practices for deriving software requirements.

The process used to derive the software requirements has been a problem for numerous aerospace programs. The Software Technology Support Center (STSC) at Hill Air Force Base has worked with defense aerospace software programs and has published a guide for software managers where they state:
"Misinterpretation of user requirements is a major, if not the greatest, contributor to software failure." [STSC, 1996]

The STSC also found communication is a major source of requirement errors. This research attempts to improve the communication process by highlighting effective processes and identifying key practices designed to improve product quality, reduce the software development cycle time, and lower cost.

2.2 Sources of Information on Aerospace Software Requirement Derivation

The Software Technology Support Center is one of several sources of information regarding aerospace software requirement derivation and software development processes. Several organizations and documents were used as referenced material for this research. They include; the Software Engineering Institute at Carnegie Mellon University, Military Standard 2167A, Institute of Electrical and Electronics Engineers, Inc. (IEEE), and the Requirements and Technology Concepts for Aviation (RTCA) Software Considerations in the Airborne Systems and Equipment Certification (DO-178B). The author recognizes there are other sources of information on software requirements that provide valuable information for process practitioners. Additional sources were considered but not included in this effort.

2.2.1 Software Technology Support Center (Hill Air Force Base, Utah)

The Software Technology Support Center (STSC) at Hill Air Force Base in Utah, is charted to be a center for weapon, command and control, intelligence, and mission critical Department of Defense software systems. STSC provides a variety of products and services associated with military aerospace software development. In addition, they also provide published guidelines governing the entire software development process which includes requirement generation [STSC, 1996]. STSC is one source of information about the challenges facing modern military aerospace software development programs.

2.2.2 Carnegie Mellon University (CMU) Software Engineering Institute (SEI)

Material from the Software Engineering Institute (SEI) was used to help formulate and understand the objectives of this present investigation of the software requirement process. A

---

1 The STSC URL (http://stsc.hill.af.mil/stscinfo.asp) provided information of the STSC mission.
1. **To achieve agreement on the requirements:** Requirements definition is a communication intensive process that involves the iterative elicitation of information from a variety of sources that have divergent perceptions of what is needed. Multiple reviews of evolving requirements by persons with a variety of backgrounds are essential to the convergence of this process. Requirement definition must facilitate communication with the end users and customers, as well as personnel who will test the software when developed [SEI-CM-19-1.2, 1990].

2. **To provide a basis for software design:** Requirement documents must provide precise input to software developers who are not experts in the application domain [SEI-CM-19-1.2, 1990].

3. **To provide a reference point for software validation:** Requirement documents are used to perform software validation, i.e., to determine if the developed software satisfies the requirements from which it was developed [SEI-CM-19-1.2, 1990].

These goals were originally developed for writing the software requirements and NOT necessarily the overall process for deriving software requirements. The three goals identified by SEI were incorporated into a stakeholder survey used in this research in an attempt to build on the research done by the Software Engineering Institute.

**2.2.3 Software Process Maturity Model [Technical Report CMU/SEI-96-TR-020, 1996]**

The SA-CMM was developed with the intention of describing the buyer role in the software acquisition process. SA-CMM defines activities they recommend be satisfied in order to
achieving a specified level of maturity. A basic description of the five maturity levels can be found in Table 2.1.

Table 2.1 Definitions of the SEI process maturity levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>The Initial Level – Few defined processes. To mature beyond this phase basic process management controls are required.</td>
</tr>
<tr>
<td>Level 2</td>
<td>The Repeatable Level – Basic software acquisition process are established.</td>
</tr>
<tr>
<td>Level 3</td>
<td>The Defined Level – Software acquisition process is defined and repeatable.</td>
</tr>
<tr>
<td>Level 4</td>
<td>The Quantitative Level – The software processes are quantitatively understood and controlled.</td>
</tr>
<tr>
<td>Level 5</td>
<td>The Optimization Level – Continuous process improvement activity.</td>
</tr>
</tbody>
</table>

The SA-CMM does provide some guidance into establishing and managing the software requirement derivation process.

“The purpose of Requirements Development and Management is to establish a common and unambiguous definition of software-related contractual requirements that is understood by the project team, end user, and the contractor” [CMU/SEI-96-TR-020, 1996].

The scope of the derivation process adopted for this research is similar to the scope defined in the SA-CMM.

“Requirements development and management begins with the translation of operational or end user requirements into solicitation documentation (e.g., specifications) and ends with the transfer of responsibility for the support of software products” [CMU/SEI-96-TR-020, 1996].

The role of the software developer is covered in a separate publication, The Capability Maturity Model: Guidelines for Improving the Software Process [SEI, 1995]. This document was observed to be widely used in the aerospace industry to help develop and mature the software practices and processes. To the author's knowledge, all but one of the systems studied (commercial aircraft auto-pilot software upgrade) had an active SEI program with established SEI maturity levels defined. SEI will be referenced on several occasions in this thesis.
2.2.4 Military Standard 2167A (DOD-STD-2167A, 1988)

Military standard 2167A has been the foundation of military aerospace software development since 1988. This standard has been replaced by a subsequent update but is included in the research because two of the three case studies (commercial auto-pilot software upgrade and the military avionics software upgrade) make reference to this standard in company process literature. This military standard describes the different phases of software development and requirement analysis activities. The document also defines deliverable products (reports), major program reviews, and audits associated with software development. To the author's knowledge, none of the ten systems studied in this project had a contractual requirement to adhere to this standard.

2.2.5 Institute of Electrical and Electronics Engineers, Inc. (IEEE)

IEEE is one of the leaders in software development literature. The IEEE Computer Society has published volumes of useful information on the software development process activities. One of the more recent publications from IEEE, *Software Requirements Engineering (Second Edition)* [Thayer and Dorfman, 1997], serves as an excellent source of papers on software process activities from some of the leading academic and industry experts. Included in this collection of papers is an *IEEE Recommended Practice for Software Requirement Specifications* and an *IEEE Guide for Developing System Requirements Specifications*. The guide highlights useful techniques for requirement practitioners of all application domains.

2.2.6 Requirements and Technology Concepts for Aviation (RTCA) Software Considerations in the Airborne Systems and Equipment Certification [DO-178B]

DO-1787B is intended to provide software developers a set of practical guides or rules for developing airborne systems [DO-178B, 1990] for airworthiness certification. A working group comprised of the United States government, university computer science departments, aviation industry and aeronautical organizations developed the document. Airborne systems are expected to perform to a high degree of confidence in safety and comply with commercial aircraft flight requirements. The derivation of software requirements is briefly covered in DO-178B which states the primary objectives of the software requirements process are to develop high-level
requirements according to the system safety process. DO-178B provides a good definition of recommended software requirement process activities. The activities associated with the derivation of the software requirements from the customer and end user needs do not appear to be specifically addressed in DO-178B. To the author’s knowledge, DO-178B was written for airborne aircraft systems and may not be as applicable in other application domains.

2.3 Software Development Processes

The general activities used to develop software are fairly standard across the aerospace industry, but the specific model adopted to implement the process can vary by company and by organizations within the same company. Effective implementation can vary across programs and/or organizations. The following steps characterize the general process phases by which software is developed in the aerospace field [DSMC, 1990].

1. **Concept Development**: Explore the role of software within the system. Define the system level requirements based on the stakeholder needs.

2. **System Requirements Allocation**: Allocate system requirements to software sub-systems. This step in the process may not be a separate phase in the process but the function is still required.

3. **Software Requirements Analysis**: Develop and evaluate software requirements for completeness, understandability, validity, consistency, and adequacy of content.

4. **Design, Code and Unit Test**: The top-level and detailed design of the software is complete. This step translates the design into computer instructions and tests the software at the lowest level (unit testing).

5. **Systems Integration**: The software components are tested and integrated into the overall system.

6. **Validation and Verification**: The system is tested in an operational environment to ensure end user needs are meet as well as prove the product meets the system requirements.
These steps are intended to represent the high level activities and not the process in its entirety. Implementation of each step can be done differently depending upon the process model the development organization adopts. An organization may modify the development process to best match the background of the organization and the application domain of the product.

This research focused on three basic software development models that are well represented in the literature. These models may not reflect all possible software development models but they do illustrate different processes for code generation.

2.3.1 Waterfall Software Development Model
In the waterfall model, the software is developed sequentially. The objective is to eliminate software requirement rework problems by defining requirements at the start of the program and then follow a serial and structured development process. A waterfall model relies heavily on the assumption that the software development process occurs only once [Brooks, 1998]. In general the software requirements are specified in their entirety early in the design of the system. The waterfall model represents more traditional software development practices and may be a challenge to implement in modern complex systems. This is largely because in general, system and end user testing is late in the development process when changes to the software can be expensive. Each phase in the process is completed sequentially upon completion of the previous phase (Figure 2.4).

Figure 2.4: Classic Waterfall Model Depicted by Brooks [Brooks, 1995]
It is important to note that the waterfall model has some variations that show feedback loops from validation and verification into earlier process steps [Boehm, 1988]. The common element between variations in the waterfall model appears to be the desire to develop software in successive stages.

2.3.2 Incremental Software Development Model

In the incremental software development model, the software is developed using a series of software releases. The risk of meeting requirements is distributed over multiple product baselines. Each software release is a deliverable product baseline that can be used to capture end user feedback. This software development model attempts to build on the lessons learned during system testing and end user operations to gradually mature the system through the incremental building of capability.

The key element of an incremental model is that at each stage a working system is always available for the end user. This allows the developers to provide feedback into the requirements before the functionality is developed in its entirety. The incremental model also allows the software to be developed according to the resources constraints such as budget, schedule, people, and test assets. The model is similar to a spiral model except the prototypes developed under the spiral model are typically not part of a final deliverable product baseline [Brooks, 1995].

2.3.3 Spiral Software Development Model

The spiral software development model uses risk reduction activities that may include a risk-based prototyping process to develop software requirements and is illustrated in Figure 2.5. Knowledge gained from the prototypes is used to mature the software requirements during the software development process. The design, code, and testing begin when a small percentage of the requirements are understood. Capability can be delivered and modified with each software release.
2.3.4 Software Development Model Summary

The process model that the software development organization adopts can have an impact on the software requirement derivation process. Waterfall models are designed to develop software in successive stages and specify most of the software requirements before design begins. Incremental models allow requirements to be undefined at the start of design and increase functionality through multiple software releases. Spiral models are designed to account for many of the requirements to be undefined and use the development process to mature the requirements at the same time as the design.

2.4 Requirement Derivation Process

A generic requirement process was adopted from MIL-STD 2167A to establish a baseline to compare the different systems. For the purpose of defining the scope of this research project, the requirement derivation process was defined as starting with defining the customer needs during
concept definition and system requirements generation phase and concludes with the completion of the software requirement analysis and the subsequent definition of the software requirements. This appears to be consistent with the process boundaries defined by the Software Engineering Institute (see section 2.2.3). The flow of these three phases is illustrated in Figure 2.6.

The basic objective of the software requirement derivation process is to transform customer needs into software requirements. The software developers use the software requirements to design the software algorithms. The software algorithms are translated into software code and delivered as a functional product to the customer. The requirement derivation process is one way the developer can get input and feedback from the end users and the customers on the product under development.

The Concept Definition and System Requirements Generation phase is initiated with a customer requirement specification or a definition of the high-level customer needs. The objective of this phase is to perform system trade analysis and define the system level requirements. A System Requirements Review may be held to conclude this phase of the requirement process. The system architecture is typically defined during this phase of the process.
The System Design Requirement Allocation phase is initiated upon completion of the definition of the system requirements. The objective of this phase is to perform sub-system trade analysis and define the sub-system level requirements. A formal Preliminary Design Review may be held to conclude this phase of the requirement process. The sub-system architecture is typically defined during this phase of the process. The allocation of the requirements is usually to subsystems and/or hardware and software. Hardware requirement analysis is not discussed as part of this research and was considered out of scope of this effort.

The Software Requirement Analysis phase is initiated upon completion of the definition of the sub-system requirements. The objective of this phase is to define and write the software requirements. A software requirement specification is usually completed at the end of this phase and the software architecture is defined. A formal critical design review may be held to conclude this phase of the requirement process.

The process used to derive software requirements will most likely vary across industry. The product, corporate culture, and development model will make each process slightly different. The purpose of utilizing the model in military standard 2167A was to establish a common baseline to compare programs. The process shown in Figure 2.6 is used in this research as a basis for comparing and understanding other processes in an attempt to extract Lean practices and understand effective processes.

2.5 Lean Aerospace Initiative (LAI)

The Lean Aerospace Initiative (LAI) started in 1993 when the United States Air Force, about two dozen major U.S. aerospace firms, and MIT formed a research consortium focused on improving quality, reducing cost, and increasing customer responsiveness for defense aerospace products. Research conducted by the MIT Lean Aerospace Initiative has focused on the fundamentals of the concept of Lean in the aerospace industry (http://web.mit.edu/lean). The basic philosophy of Lean is to eliminate all non-value added activities and be responsive to customer or market needs in order to reduce the product development cycle time, increase product quality, and reduce overall cost [Weiss, Murman, and Roos, 1996].
One of the primary products of LAI is the Lean Enterprise Model (LEM). The LEM is designed to capture the lessons learned from research and industry benchmark data in order to facilitate the transfer of knowledge across the research consortium. Data in the LEM is derived from case studies and industry surveys for all phases of the product development lifecycle. The central repository of information found in the LEM has been a successful tool to build the aerospace knowledge base of Lean. Figure 2.7 illustrates the organization of the LEM information. Meta-principles identified in the LEM focus on the high level axioms of the Lean principles applicable to the enterprise. Enterprise metrics focus on measuring the incorporation of the Lean principles into the organization. The overarching practices characterize the operation of a Lean organization or enterprise. Enabling and supporting practices are actionable guidelines for operating a Lean enterprise.

Figure 2.7: Lean Enterprise Model Architecture and Overarching Practices

<table>
<thead>
<tr>
<th>Meta-Principles/Enterprise Principles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Level Metrics</td>
</tr>
<tr>
<td>Overarching Practices</td>
</tr>
<tr>
<td>Identify &amp; Optimize Enterprise Flow</td>
</tr>
<tr>
<td>Implement Integrated Product &amp; Process Development</td>
</tr>
<tr>
<td>Maintain Challenge of Existing Processes</td>
</tr>
<tr>
<td>Assure Seamless Information Flow</td>
</tr>
<tr>
<td>Develop Relationships Based on Mutual Trust &amp; Commitment</td>
</tr>
<tr>
<td>Nurture a Learning Environment</td>
</tr>
<tr>
<td>Optimize Capability &amp; Utilization of People</td>
</tr>
<tr>
<td>Continuously Focus on the Customer</td>
</tr>
<tr>
<td>Make Decisions at Lowest Possible Level</td>
</tr>
<tr>
<td>Promote Lean Leadership at all Levels</td>
</tr>
<tr>
<td>Maximize Stability in a Changing Environment</td>
</tr>
</tbody>
</table>

This research incorporated the enterprise level metrics and the overarching practices into the design of case studies and a survey. The goal was to identify Lean practices applicable to

---

2 Based on LAI presentations on the Lean Enterprise Model
software requirement development and to benchmark the current implementation of them. Enterprise metrics fall into four major categories (http://lean.mit.edu/public/index.html):

1. **Flow Time**: Focuses on measuring order-to-delivery time and/or product development cycle time.

2. **Stakeholder Satisfaction**: Multiple measures exist and may include on-time deliveries and continuous cost/prices improvement.

3. **Resource Utilization**: Objective is to measure productivity and may include metrics on output per employee and/or inventory turns.

4. **Quality Yield**: Product and process quality measures and may include metrics on scrap and rework rate and design changes after initial release.

Lean at the organizational level is characterized by twelve Overarching Practices (http://lean.mit.edu/public/index.html):

1. **Identify and Optimize Enterprise Flow**: Optimize the flow of products and services, either affecting or within the process, from concept design through point of use.

2. **Assure Seamless Information Flow**: Provide process for seamless and timeless transfer of and access to pertinent information.

3. **Optimize Capability and Utilization of People**: Assure properly trained people are available when needed.

4. **Make Decisions at the Lowest Possible Level**: Design the organization structure and management systems to accelerate and enhance decision making at the point of knowledge application and need.
5. **Implement Integrated Product and Process Development**: Create products through an integrated team effort of people and organizations which are knowledgeable of and responsible for all phases of the product’s life cycle from concept definition through development, production, deployment, operations and support, and final disposal.

6. **Develop Relationships Based on Mutual Trust and Commitment**: Establish stable and on-going cooperative relationships within the extended enterprise, encompassing both customer and suppliers.

7. **Continuously Focus on the Customer**: Proactively understand and respond to the needs of the internal and external customers.

8. **Promote Lean Leadership at all Levels**: Align and involve all stakeholders to achieve the enterprise’s lean vision.

9. **Maintain Challenge of Existing Processes**: Ensure a culture and systems that use quantitative measurement and analysis to continuously improve processes.

10. **Nurture a Learning Environment**: Provide for the development and growth of both organization’s and individual’s support of attaining lean enterprise goals.

11. **Ensure Process Capability and Maturation**: Establish and maintain process capable of consistently designing and producing the key characteristics of the product or service.

The Lean Enterprise Model (LEM) was developed by jointly by consortium members to provide the LAI consortium a structure and framework for implementing the Lean principles [Nightingale, 1999]. It provides an intellectual structure for thinking about the relationship between the Lean philosophy and the organization. The applicability of the principles may vary in each company and organization.

2.6 Research Focus

In 1997 meetings between the LAI sponsors and MIT identified software requirements as one area where research was needed. Based on this feedback, LAI started this research effort in January of 1998. The focus of the research is to highlight effective practices for deriving software requirements and identify Lean practices for deriving software requirements.

2.7 Summary

Deriving software requirements from system requirements is an important step in the product development process. The requirements are developed early in the product development effort and have a significant impact on the product costs and performance. Yet, software developers and process practitioners cite the software requirement derivation process as an area plagued with problems. This thesis focuses on identifying Lean practices associated with software requirement derivation. This research builds on previous work incorporating the tenants and lessons learned of Lean in an attempt to highlight Lean practices associated with the requirement derivation effort.
Chapter 3: Research Method

Data used in this document was the result of a two-year study involving over 150 requirement practitioners from the aerospace industry, United States Air Force, and MIT. The research was designed to capitalize on the knowledge and experiences of previous Lean research and Lean practices while incorporating input from process practitioners of successful requirement derivation efforts. The requirement derivation process was characterized with respect to the entire product development process in order to understand the environment and context from which the requirements were derived.

Detailed case studies and multiple program surveys were combined as the primary sources of data. The stakeholder survey captured 125 responses from ten aerospace software upgrade programs and was developed using the four Lean enterprise level metrics and the twelve overarching practices found in the Lean Enterprise Model. The objective of the survey was to obtain quantitative data to characterize the software requirement derivation process. Questions focused on understanding the value added contribution of the different phases of the software development process, the requirement derivation process, and the involvement of the stakeholders.

Case studies were used to complement the survey findings. Data were collected from 45 interviews of software requirement derivation process participants. These interviews provided a detailed understanding of the software requirement derivation process. This feedback was an invaluable source of information on the challenges facing the individuals who define requirements for aerospace software programs. The case studies helped explain the trends in the survey data and offered insight into the causes of the variations in the data. Regular presentations and interactive research reviews were conducted with requirement process practitioners to help guide and shape the interpretation of the case study observations and the survey data.

3.1 Research Design

The research was designed to focus on one phase of the software development life cycle, the process used to derive the software requirements. A structured approach was adopted to
combine the experiences of successful practitioners with the lessons learned from the MIT Lean Aerospace Initiative. The approach involved five major steps and is defined below:

1. Baseline the end-to-end software development cycle time from concept development to delivery.
2. The study of successful programs that used an effective software requirement derivation process.
3. The capture of the knowledge of the program and process leadership.
4. Apply the Value Stream methodology and approach.
5. Utilization of the MIT LAI Lean Enterprise Model.

### 3.1.1 Baseline End to End Cycle Time
The first step in the data collection was establishing the end-to-end software product development cycle time. For each system, the cycle time was defined as concept definition to delivery of the product to the customer. For some programs, the end-to-end cycle time may transcend organizational responsibilities. The definition of cycle time is important because it establishes a baseline from which quantitative analysis between systems can be accomplished. Understanding the cycle time is also an indicator of how responsive the product development team is to the end user or the customer. In addition, the Lean Aerospace Initiative believes reducing cycle time is closely linked to reducing costs and increasing value of aerospace products. The ten systems surveyed in the research were asked to define the product development cycle time for a particular software release. This data provided insight into the organization knowledge of the software development cycle time.

### 3.1.2 Study Successful Programs
Failures in software development have been researched and utilized to help define lessons learned and establish principles for the development of software [Davis, 1995]. This research effort focused on gaining knowledge from successful process practitioners. Participating organizations volunteered programs they felt met three criteria:
1. The company believed the process they used to derive software requirements was effective (as the company defines effective). Most companies felt the process was effective or successful because they were able to meet or exceed the contractual requirements within a cost and schedule parameter that was acceptable to the customer and end users.

2. The program had to be delivered to the customer by the time the research data were collected but could not be more than five years old. Industry felt the recent completion of the software program was important for the data to be relevant. All ten of the systems used for this research had been completed within two years of when the data was collected. Most systems were completed less than one year from when the data were collected.

3. Collecting data on new software development efforts was very difficult. The number of new programs is limited so the research focused on software upgrades to existing systems.

3.1.3 Capture Program/Process Leader's Knowledge

The role the requirement derivation process on the overall software development program is important and typically involves the leadership of the product development team. The case study research indicated that software requirement generation typically has a core team composed of the program and/or requirement process leadership. All of the processes involved a large number of stakeholders. The background of the stakeholders varied across the programs. A core team of requirement participants was typically the most involved. For the systems studied in this effort, the core team was comprised largely of program management, the chief system engineer or technical director, senior software engineers, senior systems engineers, test engineers, customers and end users. These individuals and disciplines were the targets for survey data collection and case study interviews. In general, the process and program leadership averaged about twelve individuals per system.
3.1.4 Adopt a Value Stream Approach

The path to developing a Lean software requirement definition process starts with an understanding of the product value stream. The definition of value stream is adopted from by Womack and Jones, *Lean Thinking*, and is defined as “the collection of activities required to develop a product, from concept definition to customer delivery”. *Lean Thinking* provides good examples of how value stream mapping has proved to be a successful tool to highlight waste and non-value added process activities.

The Value Stream analysis used in this research focused on the identification of the products necessary to deliver the end user capability desired by the software. Three high level illustrations of software upgrade value streams are given in Chapter 4.

3.1.5 Utilize the Lean Enterprise Model (LEM)

The basic tenants of the twelve LEM Overarching Practices\(^3\) have been incorporated into the industry survey. The research was designed to identify the presence of the Lean practices whenever possible and characterize the relationship between the Lean practices and the process used to derive software requirements. LEM enterprise level metrics were used as a guide to format outcome measures for the systems being studied. To the author’s knowledge, no other research effort has attempted to relate Lean to the process used to derive aerospace software requirements. The knowledge gained from this research could provide an important stepping stone for industry process improvement efforts.

3.2 Stakeholder Survey

The stakeholder survey (Appendix 1) was developed with the help of process practitioners and MIT faculty. It was designed to incorporate the tenants of Lean and the knowledge of previous work done by the Software Engineering Institute [Bracket, 1990]. The questions were created to characterize the requirement derivation process with respect to the twelve LEM Overarching Practices and the SEI goals (see Section 2.2) of a software requirement derivation process. The survey was reviewed by MIT researchers and tested by a team of requirement practitioners from industry and government that were not associated with the research but representative of the

\(3\) Information on the Lean Enterprise Model can be found at the LAI URL (http://lean.mit.edu/public/index.html)

individuals to be surveyed. After multiple iterations the survey was distributed to research participants.

An industry and/or Air Force program representative handled the selection of the survey respondents. Each program representative was provided direction on the purpose and nature of the survey with suggestions on the type of disciplines representing the program and process leadership (Section 3.1.3). The survey respondents were given the opportunity to mail the survey directly to MIT or seal and return as a group. Confidentiality is an important element necessary for research of this nature. Consequently, all data was treated as confidential and non-attributable.

The survey was designed to obtain quantitative information to identify Lean practices in the requirement derivation process. Explanations of the data trends were enhanced by using the knowledge gained during the case study research. The case studies provide a deeper understanding of the issues surrounding the software requirements derivation process. The combination of case study and survey helps to establish and explain causality of the data trends. Three case studies were selected from among the ten systems surveyed. Figure 3.1 illustrates the relationship between the case studies and the surveys.

Figure 3.1: Illustrated Relationship between Case Study Data and Survey Data
The ten systems included in the research vary in the size of the software development effort, the software development cycle, and the application domain. Table 3.1 illustrates some of the discriminatory characteristics of the ten systems studied.

Table 3.1: Discriminatory Characteristics of the Ten Aerospace Software Upgrades Used in the Research

<table>
<thead>
<tr>
<th>System</th>
<th>Application Domain</th>
<th>Software Development Cycle Time (months)</th>
<th>Case Study</th>
<th>Survey Responses</th>
<th>Development Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Commercial Aircraft</td>
<td>32</td>
<td>Yes</td>
<td>13</td>
<td>Incremental</td>
</tr>
<tr>
<td>2</td>
<td>Military Fighter Aircraft</td>
<td>62</td>
<td>Yes</td>
<td>14</td>
<td>Waterfall/ Incremental</td>
</tr>
<tr>
<td>3</td>
<td>Military Fighter Aircraft</td>
<td>49</td>
<td>No</td>
<td>12</td>
<td>Waterfall/ Incremental</td>
</tr>
<tr>
<td>4</td>
<td>Military Bomber Aircraft</td>
<td>46</td>
<td>No</td>
<td>15</td>
<td>Waterfall/ Incremental</td>
</tr>
<tr>
<td>5</td>
<td>Military Missile/ Munition</td>
<td>Unknown</td>
<td>No</td>
<td>7</td>
<td>Waterfall/ Incremental</td>
</tr>
<tr>
<td>6</td>
<td>Military Missile/ Munition</td>
<td>84</td>
<td>No</td>
<td>11</td>
<td>Waterfall/ Incremental</td>
</tr>
<tr>
<td>7</td>
<td>Military Missile/ Munition</td>
<td>29</td>
<td>No</td>
<td>14</td>
<td>Waterfall</td>
</tr>
<tr>
<td>8</td>
<td>Military Missile/ Munition</td>
<td>Unknown</td>
<td>No</td>
<td>13</td>
<td>Incremental</td>
</tr>
<tr>
<td>9</td>
<td>Military Space Ground Terminal</td>
<td>7</td>
<td>Yes</td>
<td>18</td>
<td>Spiral</td>
</tr>
<tr>
<td>10</td>
<td>Military Fighter Aircraft</td>
<td>36</td>
<td>No</td>
<td>11</td>
<td>Waterfall</td>
</tr>
</tbody>
</table>

The application domain of the ten systems fell into four general areas; a commercial aircraft system, military aircraft systems, military missile/munitions systems, and a military space ground terminal system. The software development cycle time, defined as concept definition to delivery of the product to the customer, ranged from seven months to eighty-four months. Three of the ten systems provided cases study data in addition to survey data. The number of survey responses ranged from seven to eighteen with an average of twelve responses per system. Half of the systems adopted a combination of the waterfall and incremental software development model. One utilized a spiral software development model and two others a waterfall software development model. The remaining two systems adopted an incremental model for software development. Despite the diverse nature of the products, all of the systems were real-time, mission critical, embedded software systems. These common characteristics form the basis for comparison of the findings.
3.3 Case Study

The case study research was designed to provide a detailed understanding of the possible causes and effects of variations between the responsibilities of multiple stakeholders in the software requirement derivation process. These variations may be driven by the process used, the people involved and/or the nature of the product. The case studies explore the differences within the context of the system and within the bounds of the scope of the research project.

Three case studies were selected to represent a diverse sampling of real-time mission critical aerospace software upgrades. The first case study, System 1, was a software upgrade to a commercial aircraft auto-pilot system. The second case study, System 2, was an avionics upgrade to a military aircraft. The third case study, System 9, was a software upgrade to a military space satellite ground terminal. All three systems represent different application domains that offered insights into the challenges of defining software requirements for aerospace systems.

The case study field research proved to be a highly interactive experience. The research was conducted informally but in a methodical manner. Each case study was initiated with a meeting between LAI and the system industry and/or government leadership. The purpose of the first meeting was to understand how the software requirement process was accomplished and who was involved in the process. A site visit to the contractor’s facility for more detailed interviews with the program leadership and the process participants followed the first meeting.

The first task associated with each of the site visits was to establish the end-to-end process and the identification of the value stream elements. This activity proved to be beneficial to the research and to the company participants. Often the process mapping had elements outside of the software developers control and the research afforded the participants an opportunity to re-think how they manage those interfaces and relationships. The enterprise view of the software process and mapping of the value stream resulted in several internal process improvement initiatives being initiated during the course of the research. On-site visits provided the opportunity for requirement practitioners to provide candid feedback on the challenges and benefits of the process they followed.
Most of the case study interviewees were with the program and process leaders and key technical areas; Project Management, Chief Systems Engineer, End Users, Customers, System Test Engineers, External and Internal Interfaces, Software and Hardware Developers, and Systems Engineering.

3.4 Interim Testing of Results

The research presented a number of opportunities to include industry, government, and MIT feedback into the interpretation of the findings. The LAI consortium members were provided numerous opportunities to share their collective wisdom on the interpretation of the data. These opportunities included interim presentations at stakeholder facilities, LAI meetings and workshops, and MIT seminars. The interaction of the requirement community proved to be an invaluable part of the research process.

Requirement derivation can be one of the more challenging elements of the product development process to research. In the author’s opinion, two factors make practitioner feedback important to the success of the requirement research. First, the requirement derivation process involves senior individuals in the organization with significant application domain experience and appears to be heavily influenced by the individual and the program legacy experiences. These experiences are well recorded in the minds of the senior leadership and appear to influence the process the organization adopts. The interim feedback from these individuals on the trends and responses in the survey data provided insights into the data that would have been impossible to collect otherwise. Second, the lack of single contractor and/or organization responsible for activities associated with system upgrade made it extremely important to cross examine the multiple experiences of process practitioners who participated from the different stakeholder vantage points.

3.5 “Real-time” Implementation of Research Findings

The interactive industry element proved beneficial for industry and government participants as well. Several requirement process improvement activities were initiated in conjunction with this research in an attempt to capture the synergy of the research collaboration. The most notable activity is an effort by the United States Air Force (USAF) to reduce the cycle time for avionics
upgrades. The end user community representative, Air Combat Command, responsible for defining the end user requirements, led the USAF initiative. The continuous interactive involvement of MIT in the change initiative provided the USAF with opportunity to use data from other application domains. This allowed them to benchmark their process and initiative improvement activities based on quantitative data. In return, the USAF and military aircraft community provided one case study and four systems for survey data.

The USAF was not alone in using this requirement research to launch process improvement activities. The missile/munitions contractor, commercial aircraft, and space ground terminal contractors all used the data collection as an opportunity to gather benchmark data to improve their processes. The government and industry participants had a strong desire to use the research as an opportunity to improve their process despite having what they believed was an effective process for deriving software requirements. In the author’s opinion, the desire to use the data and capitalize on the synergy of this research is a clear indication of the significant opportunity that exists for researchers and practitioners to better understand how software requirements are derived.

3.6 Research Method Summary

The structured research design provided the context for the study by defining the system cycle time and the value stream elements for successful examples of software requirement development. Integrating the program and process leadership in the survey generation, data collection, and data analysis provided invaluable insights into the software requirement derivation process. The combination of data collection methods and three case studies with survey responses, provided a strong framework for understanding the requirement derivation process. The connection with the LAI LEM overarching practices provides the first look at how the Lean practices relate to the process of deriving aerospace software requirements. The integration of the Lean philosophy into the requirement process separates this research from other requirement research efforts.

Involving over 150 government and aerospace industry practitioners in the research data collection and analysis process has helped to focus the research effort and solidify the research
findings. Significant government and industry participation has allowed this effort to have an impact independent of the research findings. Data collection efforts have created new opportunities for government and industry practitioners to initiate requirement process improvement activities. These improvement activities represent "real-time" implementation of the research which will undoubtedly lead to significant changes in the software requirement process of the participating organizations. In turn, this will create a deeper understanding of the requirement derivation process and create new future research opportunities.
Chapter 4: Case Study Analysis

The case study analysis focused on highlighting the software development value stream, understanding the software development environment, and identifying the software requirement derivation process. The characterization of these aspects of the system provided a perspective to analyze observed process differences.

Defining the software development value stream is an important step in becoming Lean. The value stream analysis helps highlight the activities that add value to the software development process. A high level identification of the value stream elements for each of the three cases studies was done in the beginning of the case study data collection to help explain the role of the stakeholders in the requirement derivation process. The value stream elements helped determine which stakeholders would be involved in the process. The mapping activity helped identify the stakeholders and the importance of their involvement in the requirement process.

The focus of the value stream analysis was to isolate the products required for the software to provide capability to the end user. This idea is important because it implies software upgrades are more than just software code. During the software requirement derivation process, requirements for other supporting products are defined as well. Supporting products can include items like support equipment and training materials. Without these products the software has no 'value' to the end user because it can not be used.

The value stream analysis and the development environment offered insight into why the process was effective for each organization and allows the reader the opportunity to judge the applicability of the process to other situations.

4.1 Commercial Auto-pilot Software Upgrade Case Study

This case study was a software upgrade to an auto-pilot system for a commercial aircraft. The aircraft manufacturer had delivered a new aircraft configuration two years prior and the software upgrade studied for this effort was designed to modify the new aircraft to support a derivative configuration. The aircraft manufacturer subcontracted the software development to a leading industry supplier who specialized in auto-pilot systems.
4.1.1 Commercial Auto-pilot Software Upgrade Value Stream

The value stream elements for the commercial auto-pilot are more than just software code. The commercial aircraft software value stream includes government certification, changes to the technical orders, and support equipment changes (Figure 4.1). Government certification is required to ensure the aircraft meets all federal and international aviation regulations. Technical changes require modification to account for changes in the capability and configuration of the aircraft. Support equipment changes allow for the software to be utilized by the maintenance and support system. If these three elements are not available no capability can be delivered.

Figure 4.1: Commercial Auto-pilot Software Upgrade Value Stream

Without Technical orders, support equipment changes, and government certification, the software has no value to the customer/end users

Note: The interactions between value stream elements is not reflected in this illustration

The value stream illustration in Figure 4.1 is intended to represent software upgrades as more than just software code. Not shown is the complexity associated with managing the information and interaction between the value stream elements. Mapping the information flow between value stream elements is difficult to capture. The auto-pilot team used integrated schedules, system interface documents and requirement documents to facilitate the communication of tasks and system performance between the different value stream elements. A more detailed description of the value stream elements is shown in Table 4.1.
Table 4.1: Case 1 Software Development Value Stream Elements

<table>
<thead>
<tr>
<th>Case 1: Value Stream Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Code Value Stream</td>
<td>The software code value stream embodies the development activities associated with the generation of the software code necessary to deliver the capability to the customer and the end user.</td>
</tr>
<tr>
<td>Documentation (Technical Orders)</td>
<td>These documents provide directions on how the software upgrade should be incorporated into the aircraft as well as a documentation of the jet configuration change.</td>
</tr>
<tr>
<td>Government Certification</td>
<td>This value stream element represents the activities and products required to verify the aircraft meets mandatory flight safety requirements.</td>
</tr>
<tr>
<td>Support Equipment Changes</td>
<td>Changes to the mission planning software are necessary for the pilots to properly modify tactical aircraft operations planning based on the changes introduced in the aircraft software. The software to change the diagnostic tests is also included in this value stream element.</td>
</tr>
</tbody>
</table>

The case study interviews suggested the first order representation of the value stream shown in Figure 4.1 is a practical representation of how these different elements are viewed by the process practitioners. The separation of the value stream elements was observed to be a more accurate representation of the organizational structure than a reflection of a process flow mapping. Frequent coordination between the value stream elements was identified by case study interviewees as a key factor in the success of the program.

4.1.2 Commercial Auto-pilot Software Development Environment

The development environment was observed to be primarily shaped by the software development model, the working relationship between the aircraft manufacturer and the auto-pilot software supplier, process standards, and the level and flow of the requirement documentation.

The sub-contractor responsible for developing the auto-pilot was required to adhere to guidance in commercial practice DO-178B (see section 2.2.6). DO-178B offers guidelines on software requirement process activities but does not specifically represent a requirement derivation process or software model. The auto-pilot team indicated the incremental software model best represented the model they used to develop the software.
The relationship between the prime and the sub-contractor was based on open communication but grounded in a traditional hierarchical structure. The prime contractor controlled the access to end user information and the end user requirements were passed to the auto-pilot software developers in the form of a contractually binding auto-pilot specification. An end user requirement allocated to the auto-pilot software developer had to flow through seven different requirement documents.

1. Customer Needs (Marketing Requirements): Airline requirements are captured in this high level document prepared by the marketing division. The requirements associated with the different derivative configurations are captured in this document.

2. Aircraft Requirements: The requirements associated with the particular aircraft configuration is captured in this document. Over 300 engineers and managers during development reviewed this document.

3. Systems Requirements: Each major system within the aircraft has a requirements document. The Flight Controls and Avionics document is the major system governing the auto-pilot.

4. Auto-pilot Requirements: The major requirement document governing the development of the auto-pilot. This 600-page document was passed to the sub-contractor for implementation.

5. Interface Control Document (ICD): The technical interface between the auto-pilot and multiple systems are defined in this document.

7. **Auto-pilot Software Design Document**: This document is not a requirement document but rather a software design document. The software design is captured here.

During the case study interviews, the prime contractor and the software supplier offered insight into the philosophy behind the development requirement. Interviewees echoed a “Get it right the first time” philosophy on several occasions. This philosophy fueled a desire to specify all of the requirements accurately and completely as soon as possible. This was observed to have created two problems. First, the 600 page auto-pilot specification required continuous maintenance. Updates and revisions appear to have been costly. No automated traceability tool existed between the high-level customer requirements and the auto-pilot requirements passed to the supplier. A change to the specification required manual, labor intensive, engineering analysis of all requirement documents to ensure the change was fully captured. Second, the requirement derivation process continued almost to the delivery of the final code (see Figure 4.3). This was necessary to correct and update requirements that were specified earlier when less information was available about system performance and the operating environment.

### 4.1.3 Commercial Auto-pilot Software Requirement Derivation

Auto-pilot requirements can originate from a variety of different stakeholders and situations as represented in Figure 4.2. Program decisions can be made to incorporate certain requirements in the auto-pilot over other sub-systems. Company requirements can be generated in order to incorporate certain technologies into the auto-pilot system. Regulatory changes can cause requirement changes to the system. The end user may develop new needs requiring changes to the auto-pilot. The organization may learn lessons from legacy systems and want to incorporate the changes into the next system upgrade. The aircraft may have a new derivative configuration requiring changes to the system. Finally, the auto-pilot performance requirements may mature as the community increases the stakeholder community increases their knowledge of the aircraft performance.
The inputs from the stakeholders were collected and the aircraft manufacturer provided the necessary resources for the initiation of the product development effort. Figure 4.3 illustrates the major software development cycle time activities. Included is a breakout of the time to complete requirement derivation, the system design and software code development, and system testing.

Figure 4.3: Commercial Auto-pilot Software Upgrade Major Software Development Cycle Time Activities
Upon initiation of the program, the auto-pilot team followed a process to derive the software requirements from the various stakeholder needs (Figure 4.4).

Figure 4.4: Commercial Auto-pilot Software Upgrade Requirement Derivation Process

The software upgrade program was initiated with a milestone called “System Analysis and Design”. The objective of this milestone was to initiate the requirement derivation activities, define the auto-pilot system concepts, gather requirements from the various stakeholders and to develop a firm system configuration. A high-level requirement document was written in the form of an aircraft system specification. The aircraft system specification was created and passed to the auto-pilot developers to use as a starting point for the auto-pilot specification.

The second major activity in the requirement derivation process was the development of the auto-pilot specification. This specification was developed jointly with the supplier and the aircraft manufacturer auto-pilot engineers and system engineering division. Some of the requirements specified the flight control algorithms and as a result software development
activities began immediately. For example, aircraft software control laws were documented in the specification and passed to the supplier for immediate implementation. The subsystem architecture (e.g. hardware/software partitioning) had been fixed during the initial system development and the control laws were entirely implemented in the software.

The third major step in the process was the development of the supplier auto-pilot requirement documents. This occurred approximately halfway through the requirement process. Shortly following the start of the supplier software requirements documents, the software design documents were published. An initial release of the software was given to the aircraft manufacturer modeling and simulation team to begin testing the software to characterize the anticipated system performance. Approximately three months later, initial flight testing activities began. The software then went through a series of cycles where the software was released to flight test, anomalies were identified, the software was modified, and the changes were documented. These anomalies then drove the next release of the software and the subsequent documentation. For the software release examined in this case study, this cycle occurred on four occasions prior to the final software release.

The final step in the process was the delivery of the final software tape by the software supplier. After the final release, all of the requirement documents are updated and placed under configuration control to be used as a baseline for the next configuration release. This activity concluded both the software requirement derivation process and the software development activity.

4.2 Military Avionics Software Upgrade Case Study

The purpose of the military avionics software upgrade was to add new capability to the aircraft by incorporating new features and improving the performance of the system. The software development was done by an aircraft manufacturer that is a leading military avionics software developers.
4.2.1 Military Avionics Software Upgrade Case Study

The value stream analysis indicated some key similarities to the commercial auto-pilot case study. First, the military avionics value stream (Figure 4.5) has common elements with the commercial aircraft software development value stream. The software code, technical orders, support equipment and federal certification documentation are all similar in nature to the commercial value stream elements (Figure 4.1). The execution of the activities to generate those products is different but the value stream elements are similar. Second, the presence of changes to value stream elements other than code (technical orders, support equipment, federal certification documentation, multiple aircraft sensors, multiple weapons, and weapons and tactics trainers) further reinforces the need to analyze software upgrades at an enterprise level.

Figure 4.5: Military Avionics Software Upgrade Value Stream

The military avionics software upgrade value stream elements appear to be strongly influenced by the content of the software release and organizational divisions. The military aircraft software development value stream has elements that include changes to weapons systems, sensors, mission planning systems, and tactical trainers (Table 4.2). The value stream elements were
observed to increase the difficulty in managing the organizational challenges associated with coordination activities and software development across a large enterprise.

Table 4.2: Case 2 Software Development Value Stream Elements

<table>
<thead>
<tr>
<th>Case 2: Value Stream Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Code Value Stream</td>
<td>The software code value stream embodies the development activities associated with the generation of the software code necessary to deliver the capability to the customer and the end user.</td>
</tr>
<tr>
<td>Documentation (Technical Orders)</td>
<td>These documents provide directions on how the software upgrade should be incorporated into the aircraft as well as a documentation of the jet configuration change.</td>
</tr>
<tr>
<td>Government Certification</td>
<td>This value stream element represents the activities and products required to verify the aircraft meets mandatory flight safety requirements.</td>
</tr>
<tr>
<td>Support Equipment Changes</td>
<td>Changes to the mission planning software are necessary for the pilots to properly modify tactical aircraft operations planning based on the changes introduced in the aircraft software. The software to change the diagnostic tests is also included in this value stream element.</td>
</tr>
<tr>
<td>Multiple Aircraft Sensor Changes</td>
<td>Two sensors required changes to their software. One was an external aircraft sensor that requires software changes for the aircraft software to operate. The second sensor is an internal aircraft sensor. Both sensor changes allow the aircraft to better counter enemy threats.</td>
</tr>
<tr>
<td>Multiple Weapons Changes</td>
<td>Three weapons carried by the aircraft require a software modification in order to perform on the aircraft. Two of the weapons are being added for the first time and one is being modified to increase the weapons effectiveness. Without the weapons software changes the aircraft can not employ the weapon and utilize the capability.</td>
</tr>
<tr>
<td>Weapons &amp; Tactics Trainers</td>
<td>A training system is used to teach pilots the appropriate changes to the execution of the tactical aircraft operation and weapon system and teach new emergency training techniques resulting from the capability change.</td>
</tr>
</tbody>
</table>

The case study interviews indicated the coordination across the value stream elements was a major challenge for process participants. The development of the systems capability required coordination across five major contractors and six different government agencies. A structured and formal development environment was observed as a means to facilitate the communication between value stream elements and software developers. The information flow across the enterprise was observed to be difficult to track because of the large number of organizations...
involved. The software development team used integrated schedules and system interface documents to manage the task execution across the enterprise. Incorporating information technology tools to map information flow and track value stream dependencies would appear to be necessary for this type of development effort. At the time of the case study, the avionics development organization was beginning to explore the use of information management tools to support existing management methods.

4.2.2 Military Avionics Software Development Environment

The development environment was observed to be primarily shaped by the software development model, the working relationship between the end users, customers and the software developers, and the level of the requirement documentation.

The military avionics software was developed using a hybrid of a waterfall and incremental development model. The software is executed under a traditional waterfall model, but program management plans the project as incremental releases. The large number of organizations and contractors involved appears to make developing software without anomalies or defects a significant challenge. Changes to value stream elements outside of the software developer control required the aircraft manufacture to deliver developmental software baselines to test and verify value stream element performance. Multiple software baseline iterations were observed to be one way to reduce the risk associated with this type software development. A “clean-up tape” was planned at the very end of the program schedule to capture anomalies found during system integration flight test.

The relationship between the software developers, the customers, and the end users was less formal and more integrated than the hierarchical commercial auto-pilot development effort. A higher level of integration was observed to be a necessary element to survive in an environment where changes are coordinated across multiple organizations and multiple companies.

The military avionics software upgrade had six different requirement documents. This was similar to the commercial auto-pilot case study, however the level of detail at each level was significantly less.
1. **Weapon System Specification**: A high-level document outlining the major capabilities desired by the end user. The specification is not aircraft configuration specific.

2. **Individual Aircraft Configuration**: Outlines the major functionality associated with each aircraft configuration.

3. **Systems Requirements (Avionics)**: Most of the changes impacting the capability of the system are captured in an avionics specification. This document was almost one-sixth the size of the commercial auto-pilot equivalent.

4. **Software Specification**: The low level software requirements. The software only requirements were documented for the software developers.

5. **Interface Control Documents (ICD)**: Internal and external system interfaces are controlled with these documents. The interface documents are the primary requirement product transferred between the organizations responsible for the development of the various value stream elements.

6. **Software Design Document**: Software algorithm design requirements are captured in this document. This document is also used as a source of configuration control over the software release and documents the changes in each release.

The software developers used an electronic repository to capture the documents and trace the requirements between documents. The electronic capability allowed the developers and requirement process practitioners to track the history of a requirement and make changes to existing requirements with minimal effort.
4.2.3 Military Avionics Software Requirement Derivation

Avionics software requirements can originate from a variety of sources (Figure 4.6). In order to help control the process, a dedicated end user representative is expected to initiate the requirement development effort with a rough list of capabilities they feel the end user community would like incorporated into the next software baseline. The list is sorted by identifying firm candidates, those that must be in the next baseline, and discretionary candidates, those that are not yet determined and subject to program resource constraints, most notably time and budget.

In general, the origin of the candidates can come from the operational units, the government program office, the test community, the software developer, or the external interfaces. Improvement ideas from operational units are captured on an official request via a United States Air Force Form 37. The government program office and the operational and development test communities often initiate improvement candidates based on their flight test and system development experiences. The software development contractor often initiates candidates based on new technology or experiences from previous development efforts. Other candidates are introduced from external interfaces or other value stream elements that need aircraft changes to incorporate new capabilities or missions into the aircraft.

Figure 4.6: Military Avionics Case Study Sources of Requirement Input
The inputs into the requirement process represent different stakeholder communities whose needs must be captured and understood during the requirement derivation process. The collection of the inputs is the responsibility of the end user representative. The software developer, weapons programs, sensor programs, customers and the end users were all involved in the requirement derivation process. Major software development process cycle time activities are shown in Figure 4.7 and a breakdown of the requirement process is shown in Figure 4.8. The other value stream elements have been intentionally omitted for simplicity. Value stream elements not shown undergo their own requirement derivation process after the initiation of the software development program.

Figure 4.7: Military Avionics Software Upgrade Major Software Development Cycle Time Activities

- **System Design and Software Development**: ~31 months
- **Requirements Derivation**: ~26 months
- **System Testing**: ~16 months
- **Field**:

**Military Avionics Case Study**

Software Development Cycle Time

~62 Months
The software development program is initiated upon conclusion of the Avionics Systems Requirements Review Council (ASRRC). The ASRRC was led jointly by the customer (Systems Program Office) and end user (United States Air Force Air Combat Command) and supported by the software development organization (aircraft manufacturer). The purpose of the meeting was to review a firm list of capabilities to be incorporated in the next software release. A customer provided list of end user needs was given to the software developer in the form of broad statements: “Incorporation of Weapon system X on the aircraft.” No specific implementation was defined. The derivation of cockpit interfaces (mechanization) and performance expectations occurred after the first review. The customer list of software candidates was reviewed with the software developers and a firm list of critical requirements is generated. The completion of the ASRRC signals the beginning of the formal requirement derivation process.

The first review between the end users, customers, software developer, and value stream stakeholders was held within a year of the ASRRC. Two weeks prior to the review, end users were brought to the software developer location to evaluate early prototype mechanization of the “firm” software candidates. To provide the software developer with a prioritized list of software candidates the end users rank the candidates in order of importance. The prioritization allowed
the software developer to narrow down the end user needs and efficiently appropriate resources required for the execution of the software requirement derivation process.

The second review was approximately four months later and was expanded to include the representatives of the external value stream elements (weapons, sensors, etc.). The objective of the meeting was to identify concepts based on the end user needs. Simulations were run two weeks prior to the meeting to show the end users prototypes of the cockpit displays. Preliminary program cost and schedule estimates were prepared based on the end user needs. The candidates were then reprioritized based on this early analysis to establish a list whose estimated development costs equal 120% of the customer budget.

The external interfaces to the avionics were “frozen” approximately eighteen months into the requirement process. Stakeholders agreed upon changes to all of the value stream elements including the changes impacting the sub-systems or weapons outside of the avionics software developer control. This step was necessary for the software developers to complete accurate cost, performance, and schedule estimates for the third formal review.

The third major stakeholder review was held shortly after the external interface freeze. This review solidifies the final implementation concepts. The scope was determined based on detailed cost estimates and updated trade studies. High fidelity modeling and simulation of cockpit displays were shown two weeks prior to the meeting to support the decision making process. The final system requirements were defined and work began on the sub-system and software requirement definition.

A formal system design review was held approximately two years after the process was initiated. The internal interface requirements were agreed upon at this meeting and the preliminary software design was initiated. The software requirements were written, the software specification approved, and the software requirement derivation process completed.
4.3 Military Space Ground Terminal Software Upgrade Case Study

The military space ground terminal is responsible for the fusion of multiple data inputs from U.S. space based assets and the display of this data to the United States Space Command operators who support mission critical operations. The purpose of the software upgrade to the ground terminal was to add new capabilities and missions to the ground terminal system. The software development contractor was primarily responsible for the development of the system. Suppliers were used to develop specific system functions. Unlike the commercial auto-pilot case study, the suppliers were not responsible for delivering a software configuration. The suppliers were integrated in the software development team and expected to contribute as members of an integrated development team with the ground terminal contractor having the primary responsibility for the software.

4.3.1 Military Space Ground Terminal Software Upgrade Value Stream

The value stream had similar elements as the commercial auto-pilot and the military avionics case studies. The software code value stream, documentation and government certifications are similar between case studies. Code is required to deliver the software functions. Technical orders and government certification were required for the software to be released to the field. Government certification was required prior to delivery of the system to the end users.

The presence of additional value stream elements beyond the software code (documentation, government certifications, broadcast system changes, and trainers software) suggests the need to analyze software upgrades at an enterprise level. A more detailed explanation of the different elements is shown in Table 4.3. The military space ground terminal software upgrade value stream is depicted in Figure 4.9.
The value stream elements appeared to be shaped by the content of the release. With the exception of the broadcast system software changes, the military space ground terminal software developer had control over the value stream elements. Case study interviewees suggested coordination across the value stream elements and the close relationship between the government customer, end user, and software developer were key ingredients to the success of the team.

Table 4.3: Military Space Ground Terminal Software Development Value Stream Elements

<table>
<thead>
<tr>
<th>Case 3: Value Stream Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Code Value Stream</td>
<td>The software code value stream embodies the development activities associated with the generation of the software code necessary to deliver the capability to the customer and the end user. This is common on all three case studies.</td>
</tr>
<tr>
<td>Documentation (Technical Orders)</td>
<td>These documents provide directions on how the software upgrade should be incorporated into the ground terminal system as well as a documentation of the capability change associated with the configuration change.</td>
</tr>
<tr>
<td>Government Certification</td>
<td>This value stream element represents the activities and products required to verify the aircraft meets mandatory performance requirements which include safety in the form of reliability requirements.</td>
</tr>
</tbody>
</table>
Broadcast System Software Changes

The system used to broadcast the data from space based assets to the ground terminals required a software modification. This modification required changes by the ground terminal system to accommodate the broadcast system and accomplish their role of display the integrated sensor picture.

Trainer Software

A training system is used to teach terminal operators the appropriate changes to the execution of the operation of the system and new emergency training techniques resulting from the capability change.

Integrated schedules and interface control documents were used to manage the activities between the value stream elements. At the time of the research, the software developer was exploring the use of information technology tools to facilitate the communication of critical requirement information across value stream elements.

4.3.2 Military Space Ground Terminal Software Development Environment

The military space ground terminal development environment was observed to be shaped by the software development model, the working relationship between the software developers and the end users and customers, and the level of the requirement documentation. This is also similar to the military avionics case study.

The military space ground terminal software was developed using a spiral development model with an emphasis on the software prototyping. Software prototypes were used to demonstrate and explore the performance of software candidates with the customers and end users prior to implementation. Case study interviews indicated the use of a spiral software model would not have been possible had it not been for the close working relationship between all stakeholders, customers, and especially the end users. As a result, the military ground terminal software was developed under a highly integrated product team environment. For example, the software subcontractors were collocated at the space ground terminal contractor software development facility and shared software development assets.

The collocation of the suppliers also facilitated the rapid transfer of information between requirement process participants. The quantity of the requirement documents was observed to be another method the organization used to transfer knowledge quickly. Only three types of
requirement documents existed for the system, as compared to the seven on the commercial autopilot system and six on the military avionics system.

1. **System Specification**: The only requirement document covering the ground terminal system. Both hardware and software requirements are captured here. The contractor adopted a commercial hardware platform that simplified the requirement documentation and allowed the team to focus on system performance with respect to the software because the hardware requirements became fixed.

2. **Interface Control Documents (ICDs)**: Similar to the two other case studies. The external and internal system interfaces requirements are documented in ICDs and distributed to the respective value stream elements and stakeholder representatives.

3. **Software Design Document**: The effective capture of the software requirements in the system specification allowed for a clear distinction between design and performance requirements. The software design was captured in the software design document.

A software requirement specification was not used. Software development team members discovered many of the requirements located in the software specification to be either system capability requirements or software design. The software specification was deleted and the requirements were placed in either the software design document or the system specification. Case study interviews indicated the effective management of requirements and the adherence to integrity between the documents was a key contributor to the ability of the team to rapidly evaluate new requirements and understand the performance of these requirements with respect to the existing software baseline. This process was observed to be very effective and ideally suited for this team because of the experience level of the process participants, the maturity in their development process, technical competence of the organization, and product complexity.
4.3.3 Military Space Ground Terminal Software Requirement Derivation Process

The military space ground terminal software requirements originated from similar sources as the two previous case studies (Figure 4.10). Differences in the origination process appear to be centered on the selection and prioritization of the candidates for implementation. The commercial auto-pilot program aircraft manufacturer selected the candidates for implementation. The military avionics program government end user representative(s) selected the candidates for implementation not the aircraft manufacturer. The end users and aircraft manufacturer spent a good deal of time prioritizing and sorting the candidates and subsequently the requirements.

Figure 4.10: Military Space Ground Terminal Case Study Sources of Requirement Input

The input from the stakeholders was incorporated in the software development and requirement generation process. The process was initiated with a meeting between the end users, the government customer, and the software developers and signaled the beginning of the software development cycle. Figure 4.11 illustrates the major software development activities.
Figure 4.11: Military Space Ground Terminal Software Upgrade Major Software Development Cycle Time Activities

- System Design and Software Development ~ 5 months
- Requirements Derivation ~ 4 months
- System Testing ~ 3 months

Military Space Ground Terminal Case Study Software Development Cycle Time ~ 7 months

Figure 4.12 illustrates the major activities associated with the military space ground terminal software upgrade process.

Figure 4.12: Military Space Ground Terminal Software Upgrade Requirement Derivation Process

- Engineering Analysis Control Board (EACB)
  - Program Initiation
  - Concept Definition
  - Prototype Definition
- Technical Interchange Meeting 1
  - Technical Review of Project
  - Customer & End User feedback on Software Prototypes
- Technical Interchange Meeting 2
  - Technical Review of Project
  - Customer & End User feedback on Software Prototypes
- Engineering Analysis Control Board (EACB) 2
  - Concept Selection
  - 100% Cut
  - Prototype Selection
  - Ground Terminal Requirement Derivation Process Complete
The objective of the first meeting, the Engineering Analysis and Control Board (EACB), was to assure the stakeholders understood the software candidates and the associated concepts to be included in the software baseline. The system level requirements were outlined and agreed upon by the stakeholders. Software prototypes were defined and the requirement derivation process was formally initiated.

Following the EACB the end users, government customers, and software developers went through a series of technical interchange meetings designed to increase the flow of communication between developers and end users. Meetings were informal and designed to resolve questions associated with the expected software performance and the operation of the system. The second and third meetings were called Technical Interchange Meetings (TIM) and occurred within one month of each other.

The fourth and final meeting between the stakeholders concluded the software requirement derivation process. The final meeting was the second Engineering Analysis Control Board whose purpose was to approve the candidates for implementation into the software product baseline. Software candidates not ready for implementation were either studied further or cancelled. The end user and the government customers served as the chairperson and decision making authority for the meeting. Changes made to the software requirements were documented in the system specification. EACB meeting minutes provided a documented history for the origination of end user needs.

4.4 Case Study Analysis Summary

The case studies represent software upgrades in three different application domains: commercial aircraft, military aircraft, and military space ground terminals. The case study analysis focused on highlighting the value stream elements, understanding the software development environment, and identifying the software requirement derivation process.

The value stream analysis proved to be a very effective method for identifying the elements necessary to deliver capability to the end user. The presence of value stream elements other than
Software code highlighted the need to analyze software upgrades in a greater context than just coding. Software upgrade programs associated with the case studies required a comprehensive and complete enterprise level understanding of the system in order to analyze and understand the requirement derivation process.

The commercial auto-pilot upgrade, the military avionics upgrade, and the space ground terminal upgrade had common value stream elements (software code, government certification, and technical orders) and some unique elements as well. Common elements may be more similar in their purpose than their process. The nature of government certification is different between the three case studies. The commercial auto-pilot system requires a very formal approval process focused on flight safety evaluated by the Federal Aviation Administration and their world counterparts. The military avionics case study suggested the government was concerned with flight safety and system performance. The military space ground terminal case study suggested the government certification focus be centered on system performance with system reliability being an emphasized performance requirement. Unique value stream elements appeared to be driven by the application domain of the product. The value stream analysis helped to identify the unique application domain specific elements associated with the software development effort.

A common observation across all three case studies was the importance the case study interviewees placed on the coordination across the value stream elements. In all cases the tracking of activities across these elements was primarily done through integrated schedules and system interface documents. The focus on task execution is a common theme on all of the case studies.

The military space ground terminal case study used a more structured and formal development environment than the commercial auto-pilot case study. End user feedback into the selection of software candidates were captured and their roles in the process was documented. This environment provided the means to facilitate the communication between value stream elements and software developers. This is similar to the military avionics case study.
A difference between the requirement processes of the three case studies is the relationship between the software development cycle time and the requirement selection process. Understanding the end user defined priority of each candidate appeared to be more important to the longer cycle time military avionics program than the shorter cycle space ground terminal program. The military space ground terminal cycle time was seven months versus the avionics cycle time of 62 months. If a space ground terminal software candidate entered development and was not mature enough to enter the deliverable software baseline, the development contractor could move the candidate to the next release causing only a seven month delay to the end user. A similar situation on the military avionics development program may require the end user to wait several years before the candidate can be incorporated. The military avionics software developer wanted to make sure if this situation was presented and a candidate was moved to the next baseline, that it was not one of the most important end user candidates. The case study interviews suggested the end users and software developers jointly believe the failure to deliver a high priority software candidate within the desired baseline will most certainly impact the end user satisfaction. The space ground terminal program did not appear to be faced with this challenge because they had the opportunity to adopt a software development model and requirement generation process to rapidly develop software.

The software development environment was observed to be influenced by the software development model, the working relationship between the software developers and the stakeholders (primarily suppliers, customers, and end users) and the level of detail and quality of the requirement documents. All three case studies chose to adopt three different software models that appeared to be strongly influenced by the relationship between the software developers and the stakeholders. The commercial auto-pilot system was observed to have a hierarchical relationship between the stakeholders that was managed by the aircraft manufacturer. The auto-pilot team elected to use an incremental model to develop the software that allowed the aircraft manufacture to “flow down” the requirements to the supplier responsible for the software development. Auto-pilot requirement documents were extremely large, detailed, and difficult to maintain. Changes to the requirements occurred through the entire software development process.
The military avionics software upgrade chose to adopt a hybrid of an incremental and waterfall model. A long development cycle time and the large number of value stream elements created an environment where a structured and formal process was needed to accomplish the requirement derivation process. The software development and requirement generation teams made good use of electronic information systems and streamlined the requirement documents to lower the labor effort required to support requirement derivation and analysis activities.

The space ground terminal upgrade adopted a spiral model that appeared to support the short product development cycle time. The shortened software development cycle time provided flexibility in the process to reduce the amount of time required to sort and prioritizes requirements.
Chapter 5: Lean Practices

Lean originated in a manufacturing environment [Womach and Jones, 92] and its applicability to other enterprise functions, including the software development environment is emerging. This research looked at the software requirement development practices found in the framework of the Lean Enterprise Model (LEM) (see Chapter 2.5). The findings presented in this chapter provide an assessment of the state of “Leanness” in deriving software requirements and present a foundation for future Lean research in software development and product development. Although the data is specific to software development, the findings may be applicable to requirement generation in other product development areas.

5.1 Lean Enterprise Metrics

Enterprise metrics in the LEM are focused on four major areas; Flow Time, Stakeholder Satisfaction, Quality Yield, and Resource Utilization (Figure 5.1). The research was designed to look at each of their measures to investigate applicable metrics in the software development and requirement derivation environment.

Figure 5.1: Lean Enterprise Metrics and their Relationship to Software Requirements

<table>
<thead>
<tr>
<th>Lean Enterprise Metrics (<a href="http://web.mit.edu/lean">http://web.mit.edu/lean</a>)</th>
<th>Enterprise Metrics Related to Software Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Flow Time: order to delivery time in month, product development cycle time</td>
<td>• Software development cycle time: concept definition to delivery of the product</td>
</tr>
<tr>
<td>• Stakeholder satisfaction: on-time deliveries, continuous cost/price improvement</td>
<td>• Customer and End User satisfaction</td>
</tr>
<tr>
<td>• Quality Yield: scrap and rework rate, design changes/initial release/project plan</td>
<td>• Percentage of the total software development cost associated with rework of the derived software requirements</td>
</tr>
<tr>
<td>• Resource Utilization: output per employee</td>
<td>• Percentage of the unplanned (additions, modifications, deletions) derived software requirement changes</td>
</tr>
</tbody>
</table>

• Resource Utilization: lines of code per employee or per dollar ($)
5.1.1 Stakeholder Satisfaction

The value stream analysis (Chapter 4) for three aerospace software upgrades illustrated a large and diverse stakeholder community. For this research, the customer and end user were selected as the primary focal points for measuring stakeholder satisfaction and two sets of data were collected. First, the program leadership (including the customer and end user) were asked to define their customer and end user (Appendix 1: Question 1.0). Second, the survey respondents were asked to evaluate the level of satisfaction of the customer and the end user (Appendix 1: Question 3.4).

5.1.1.2 Defining the Customer

During the case studies, the interviewees repeatedly expressed that their software requirement process was focused on the customer. All of the interviewees from the ten programs claimed to be organized so they could achieve high levels of customer satisfaction. To better understand the relevance of a customer focused processes the survey respondents were asked to define the customer. The responses to the question shown in Figure 5.2 indicated the term “customer” does not have a uniform definition across programs or application domain. The lack of uniform responses on each system indicated that within a given program there are multiple definitions of the customer. This needs to be understood when considering subsequent data related to customers.

**Figure 5.2: Defining the Customer**

Assuming that the "Customer" is defined as the organization/person to whom you deliver your product and "End User" is defined as the person who operates the system. For your project which of the following categories best fits the terms "Customer" and "End User".
Case study interviewees offered two explanations for the lack of consistent responses across all systems. First, the respondents felt the term “customer” represents the next organization or person in the software development process. Software developers repeatedly mention the system test organization as their customer because the code was delivered to test and their involvement in the development effort subsided. As a result, some responses indicated “another unit within your organization” as their customer. A second explanation offered by the interviewees suggests the multiple organizations depicted in the value stream analysis (Chapter 4) create artificial barriers in the product flow. Once the product was delivered to the next company or the government, the development process was complete. In the case of the military avionics program, delivery of the software to the government system program office is not equivalent to delivering the product to the pilots. The government system program office notifies the United States Air Force Operational Test and Evaluation (OT&E) organization the software has met the contractual requirements and is ready for operational testing. The OT&E organization is responsible for the process to evaluate and examine the product prior to delivery of the software to the operational pilots.

5.1.1.3 Defining the End User

In contrast to the definition of the customer, the definition of the end user appeared to be much more clear to the survey respondents as illustrated in Figure 5.3. Data suggested that the definition of the end user varied by application domain. The commercial auto-pilot defined the end user as the airline pilot. The military avionics upgrades, the missile/munitions upgrades and the military space ground terminal upgrades defined the end user as the warfighter. The term warfighter is common among the military technology development community and is intended to represent the operational combat units. For the military aircraft and missile/munitions program this term is interpreted as the pilots. The military space ground terminal interpretation includes the operational ground terminal operators.
Figure 5.3: Defining the End User

Assuming that the "Customer" is defined as the organization/person to whom you deliver your product and "End User" is defined as the person who operates the system. For your project which of the following categories best fits the terms "Customer" and "End User".

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%-Govt SPO</td>
<td></td>
</tr>
<tr>
<td>90%-Space Ground Terminal</td>
<td></td>
</tr>
<tr>
<td>80%-Missile/Munitions</td>
<td></td>
</tr>
<tr>
<td>70%-Military Avionics</td>
<td></td>
</tr>
<tr>
<td>60%-Commercial Aircraft</td>
<td></td>
</tr>
<tr>
<td>50%-Another Unit</td>
<td></td>
</tr>
<tr>
<td>40%-Airline Pilot</td>
<td></td>
</tr>
<tr>
<td>30%-Prime Contractor</td>
<td></td>
</tr>
<tr>
<td>20%-Warfighter</td>
<td></td>
</tr>
<tr>
<td>10%-Other</td>
<td></td>
</tr>
</tbody>
</table>

Survey Respondents Definition of End User

The uniform response across all systems is consistent with the case study feedback. Case study interviewees expressed a strong desire to make sure the requirement derivation process developed products that would meet the end user needs. Having a consistent definition of the end user was observed as an important principle in developing a product that satisfies the end user needs.

5.1.1.4 Customer and End User Satisfaction

The case study research did not uncover clear quantitative customer satisfaction or end user satisfaction metrics for deriving software requirements. Customer and end user satisfaction appeared to be evaluated on a subjective scale. The commercial auto-pilot case study interviews suggested customer and end user satisfaction was more a function of how many aircraft were purchased and how many discrepancy reports were written about the system. Both the military avionics case study and the military space ground terminal case study interviews indicated customer satisfaction was reflected in the amount of profit given to the contractor or the score a government system program office gave the contractor on a formal Contractor Performance Assessment Report (CPAR). Like their commercial counterparts, the military systems also
tended to equate end user satisfaction with the number of defects or anomaly reports that were written by the operator.

Given the lack of a clear metric, the survey respondents were simply asked to estimate the level of satisfaction of both the end user and the customer. (Appendix 1: Question 3.4). The results varied across the systems and are shown in Figure 5.4.

**Figure 5.4: Estimated End User and Customer Satisfaction**

For your program, estimate the level of satisfaction of both the end user and the customer.

The standard deviation of the responses for most of the systems was less than one. For seven of the ten systems (Systems 1, 3, 5, 7, 8, 9, and 10) the level of customer satisfaction was approximately equal to the level of end user satisfaction. The end user satisfaction was notably higher than the customer satisfaction for System 4. Customer satisfaction was noticeably higher than end user satisfaction on System 6 and slightly higher on System 2.

The difference between the level of end user and customer satisfaction between systems is interesting. Despite surveying programs that were thought to be successful, four of the ten programs believed the customer and end user was “very satisfied”. Case study feedback from
the requirement practitioners offered two possible explanations. First, developing software requirements and subsequently the coded software for aerospace systems can be very challenging. Success is viewed by some as a relative experience. During the case study interviews, many practitioners shared experiences of failed software programs and ineffective requirement processes. For these individuals success may have simply been the delivery a software system within the budget available, the time required, and the performance that pleased the end user. A second explanation relates to the lack of firm requirement effectiveness measures with respect to the end user and the customer. The lack of any clear and quantitative measure makes evaluating requirement process effectiveness difficult.

5.1.2. Flow Time
Flow Time was the second enterprise metric examined. In a manufacturing enterprise, Flow Time can be thought of as the time required between placing an order to the delivery of the product. This research adopted the end-to-end software development cycle time, from concept definition to delivery of the product to the customers, as the enterprise level flow metric. The survey asked the program and process leadership to define the software development cycle time (Appendix 1: Question 2.3). The identification of the actual software product flow and the subsequent software development cycle was accomplished by interviewing the Program Manager, Technical Director, or Chief Systems Engineer for all 10 systems. For each program this was then used as a baseline to compare the survey responses for their respective programs.

The survey results for each system are shown in Figure 5.5 and varied widely for most systems. Only two systems, System 9 and System 3, had small variations in the responses. Six of the systems contributing data had variations in their responses from 35% to 55%. 
The wording of the question might have been a problem for some of the survey respondents. The words “delivery of your product to the customer” might be confused due to a lack of common understanding of the term “customer” (Figure 5.2). Also the word “product” might be interpreted to be the work of the individual rather than the final software product. Although these reasons could explain variations below the true value they could not explain variations above the true value.

Figure 5.5 clearly shows many of the individuals on the system did not know their product cycle time. Figure 5.5 only represents eight of the ten systems. Restructuring of the program scope and duration had occurred so frequently on the other two systems that the respondents did not know when the project originally started or what the content was when it started.

The case study interviews highlighted two reasons for varying responses. First, the respondents were confused about the term “cycle time”. Responses from three of the military aircraft programs confused the software update cycle (the time between software updates) with the
software development cycle time (the time to develop a software update). Second, the lack of an established definition of software development start and end activities appears to be another reason for the lack of consistency in the responses. Many of the software developers felt the software development effort ends when the code was delivered to system test. Several of the software developers felt the software development cycle time began when the software requirements analysis started, not when concept definition began. These observations appeared to be linked with the challenge of managing software development efforts where require information needed to be passed across large numbers of value stream elements.

5.1.3 Quality Yield

Quality Yield measures found in manufacturing enterprises were modified for the requirement derivation process. The Lean manufacture focus of the quality yield measure has been centered on concepts like scrap and rework. Requirement volatility metrics were used in all three of the case study programs to track changes, additions, or deletions to the software requirements. At the time of the research no metric emerged as a method for measuring the cost of rework. The second case study, the military avionics upgrade, had started tracking the cost of requirements rework at the time of the investigation, but no data was available at that time for the software release used in the study.

5.1.3.1 Requirements Rework

Two questions were asked with respect to requirement rework. First, the survey respondents were asked to estimate the percentage of the total software development cost associated with the rework of the software requirements (Appendix 1: Question 2.5). Second, the survey respondents were asked to estimate the percentage of the unplanned derived software requirement changes (additions, modifications, and deletions) with respect to the total numbers of software requirements (Appendix 1: Question 2.6).

The response rate to the questions indicated that questions with respect to requirement rework are difficult to answer. Approximately 38% did not respond to either question on requirement rework despite only being asked for an estimate. Of those that did respond the replies varied significantly for each system and across all ten systems.
The data in Figure 5.6 shows the variation in each system response as well as the average response for all systems for the first question associating rework with cost.

**Figure 5.6: Estimated Percentage of Total Software Development Cost Associated with Software Requirement Rework**

On your program, approximately what % of the total software development cost is associated with rework of the software requirements derived from system requirements?

The lowest estimated average for any system was approximately 6%. The highest estimated average across the ten systems was approximately 43%. The association of requirement rework and software development cost was observed to be a new concept for some of the practitioners involved in the research. The variability in the responses indicated that the program and process leadership had multiple perspectives on the impact of software requirement rework. This correlates, with the cost of rework. Programs with the top three range of responses (Systems 1, 6, and 9) also had approximately two and a half times the combined average cost of rework (28% versus 8%) versus the systems with the three lowest ranges of variability (Systems 3, 5, and 10).

The average percentage of unplanned derived software requirements had a much wider variation in responses and a higher total average of 23%. One system had estimates less than 5% and as
high as 200% rework. The estimated percentage of the requirement changes was difficult to estimate. The high, low, and average estimates for all systems are shown in Figure 5.7.

**Figure 5.7: Estimated Percentage of Unplanned Derived Software Requirement Changes**

On your program, relative to the total # of software requirements derived from system requirements, estimate the % of the unplanned derived software requirements changes (additions, modifications, deletions) during the development effort?

On average 23% of the requirements are subject to unplanned changes costing 16% of the total software development cost, despite having effective processes to derive software requirements. One practitioner suggested a possible cause of the large variations could be due to widely varying definitions of derived software requirements.

The percentage of unplanned rework shown in Figure 5.7 appears to show a strong correlation between the involvement of the software developer leadership (government customers and end users were excluded) in the requirement derivation process. Figure 5.8 shows the estimated percentage of unplanned changes for each system (shown in Figure 5.7) plotted against the percentage of the contractor software developers who claimed to have worked in both the concept definition and the software requirement analysis activities.
The data shown in Figure 5.8 indicates a relationship between the involvement of the contractor leadership and the estimated percentage of rework of the derived requirements. As the percentage of respondents who worked in both phases (Concept Definition and Software Requirements Analysis) increased, the percentage of estimated rework of the derived software requirements declined.

5.1.4 Resource Utilization

The fourth Lean enterprise metric is focused on measuring the output per employee. Case study interviews suggest software programs typically use a resource utilization measure associated with the source lines of code to some variable such as time or money. The value stream analysis in Chapter 4 suggests measures based on lines of code may not be an appropriate or effective enterprise measure. An example is the military avionics upgrade case study. The value stream analysis discussed in Chapter 4 highlighted a large number of external elements required to field a capability. These value stream elements are determined by the capability desired by the end
user and the customers. The percentage of effort required to code the avionics software is small compared the total cost of the program.

Figure 5.9: Typical Cost Distribution Across the Value Stream for the Military Avionics Software Upgrade [Gregory, 1998]

The data shown in Figure 5.9 was derived from historical data from an earlier software upgrade program similar to the Case 2 system and is intended to illustrate a typical software only change a military avionics system. The amount spent on the weapons programs is not included when calculating the cost of developing the value stream elements. Most of the weapon systems support multiple aircraft platforms and are funded separately. If the money for the weapons programs was added to the total budget, the percentage of the costs distributed to the aircraft manufacturer would be lower.

In this case, the avionics coding costs include a part of the aircraft developer section of the Figure 5.9. The relatively small percentage of cost attributed to the aircraft software developer, 37%, is perhaps the strongest case for the need to map the value stream elements. Figure 5.9 suggests that the greatest potential for reducing cost lies outside the aircraft manufacturer
control. Government Furnished Equipment (GFE) sensors and supporting subsystems are as significant a percentage of the total budget as the aircraft manufacturer.

The decomposition of the aircraft developer cost illustrates a representative distribution of costs for a military avionics software upgrade (Figure 5.10).

**Figure 5.10: Cost Breakdown for Case 2 Aircraft Developer [Gregory, 1998]**

For this example, the system test costs were 50% of the total software development effort. The case study interviews claimed that the cost of coding was included in the design portion, 30%, of the cost distribution in Figure 5.10. Interviewees claim the coding effort is less than half of the design effort. This data would indicate the coding effort is approximately 15% of aircraft manufacturers total cost, which in turn is only 37% of the total cost to develop the capability. Thus the cost of software code generation for an avionics upgrade is less than 6% of the total cost of the upgrade.

Case study interviews suggest this data is representative of military avionics software upgrades. As a result, the most significant opportunities to reduce total aircraft software cost exists outside
of the developing the software code! Therefore data measuring code output may not be the most accurate measure of enterprise resource utilization and productivity. For this reason, survey data was not obtained for a resource utilization enterprise metric.

5.2 Lean Overarching Practices: The Organizational Level

Identifying Lean practices at the organizational level was accomplished by analyzing the experiences of process practitioners. Case study interviews were used to identify process elements that worked well, process elements that needed improvement, and to speculate on future opportunities for change. Responses were then aligned with the twelve Lean Overarching Practices (Chapter 2) by the author. The result was an observation by the author suggesting that the Lean practices could be sorted into three categories:

1. **Effectively Implemented**: Lean practices that were observed as process elements that worked well.

2. **Opportunities to Improve**: Lean practices that were observed as process elements that could be improved.

3. **Future changes**: Lean practices that were not observed but are candidates for future implementation.

The representation of the requirement derivation process with respect to the Lean Enterprise Model and the Lean practices allows software requirement practitioners the opportunity to capitalize on the existing Lean knowledge in order to improve the process.

5.2.1 LEM Overarching Practices: Effectively Implemented

Interviewees from all ten systems used as part of this research thought they had an effective process for deriving software requirements. The responses from the interviews helped to relate the elements of the process that worked well in the context of the Lean Enterprise Model. Survey results and all three cases provided data supporting the presence of five Lean practices as
being “Effectively Implemented” in the software requirement derivation process (Figure 5.11). Data collected for each of these five practices will now be discussed in detail.

Figure 5.11: Lean Enterprise Model (LEM) Overarching Practices: Effectively Implemented

5.2.1.1 Implement Integrated Product and Process Development (IPPD)

“Create products through an integrated team effort of people and organizations which are knowledgeable of and responsible for all phases of the product’s life cycle from concept definition through development, production, deployment, operations and support, and final disposal” (LEM).

IPPD is comprised of several elements and enabling practices. Two were studied as part of this research: integrated product teams in the case studies and survey data on phases of the software development process that add value.

The importance of the first enabler studied in this research, implement integrated product teams, was illustrated with the value stream analysis in Chapter 4. Process practitioners believe that the coordination of activities across large numbers of value stream elements is more effective when
those effected stakeholders are actively involved and a contributing member of a requirement process team.

All ten of the systems included in the research believed to be using an integrated product team approach to product development. During the case study interviews, all teams claimed to have customer and end user involvement in their product team, all claimed to have suppliers integrated, and all claim to have of the stakeholders involved in the integrated product team from concept definition to delivery of the software to the customers. However, the differences between the three case studies requirement teams are significant with varying degrees of team integration. Some teams were highly integrated while others were organized under a less integrated and formal structure.

Previous LAI research in supplier relations suggested three types of integrated product teams (Figure 5.12) [Bozdogan and Deyst, 1996]. The “old” approach to an integrated product team is the one where a hierarchical relationship structure exists between the prime contractor and the suppliers. Rigid boundaries separate the prime contractor from the suppliers, customers, and end users. The “current” lean approach is the one adopted by many in the aerospace industry where the old formal boundaries are being replaced by more collaborative relationships. The relationship between the prime contractor and the suppliers, customers, and end users is not completely fluid because it is constrained by previous workshare arrangements and organizational relationships. A third type of IPT is a new or “emerging” Lean organization with no boundaries, and a highly collaborative and seamlessly integrated organization. The team is tightly integrated and can be collocated. The commercial auto-pilot software upgrade, military avionics software upgrade, and the military space ground terminal software upgrade provided clear examples of the old approach, the current approach and the emerging approach, respectively.
The case study interviews with the commercial auto-pilot indicated that the organizational relationship between the end users, customers, prime contractor, and suppliers was one based on company culture and legacy experiences. In contrast, the military space ground terminal indicated the relationship between the end users, customers, prime contractor, and suppliers was based on necessity. The military space ground terminal team felt that a highly integrated organizational structure was necessary to rapidly develop software. Case study interviews of the military avionics team indicated a strong desire by their team to move to a more integrated relationship, similar to the military space ground terminal team. The desire of the military avionics team to be more integrated was fueled by a belief that a more integrated relationship would allow the team to develop the software more rapidly and cost effectively. The less rigid the interfaces between the end users, customers, prime contractor, and suppliers, the more the team desired a more integrated organizational structure. The more integrated the structure, the more rapidly the team was able to develop the software.

The second enabler, implement integrated process development, was analyzed using stakeholder survey data and is represented in Figure 5.13. In general, a software development process has
multiple phases starting with concept definition and ending with validation and verification of the system. The software development model and content may impact the required activities and in some cases, the sequence of the activities associated with each phase. To optimize the product flow, the non-value added activities in each of the phases need to be identified in order to improve the integrated product development process. Survey recipients were asked to identify the value that each phase in the software development contributes to developing software in a timely, cost effective approach to meet the end users needs (Appendix 1: Question 2.0). The responses across the multiple systems did not vary significantly and were aggregated together in Figure 5.13.

**Figure 5.13: Estimated Value of Each Software Development Phase**

Estimate the value that each of the following contribute to developing software in a timely, cost effective approach to meet the users needs.

![Estimated Value of Each Software Development Phase](image)

The data indicates that all process phases add approximately the same level of value. The data also suggests the level of contribution of all phases to be almost "a critical contribution".

The survey respondents were also asked to evaluate their program performance in each software development phase. Unlike the previous question, the responses by system did vary by product
type. Responses are plotted in Figure 5.14 together with the data in Figure 5.13. All four military avionics upgrade program respondents felt that they had executed equally well in all phases. The commercial program and the four missile/munitions programs felt that they had executed the last three phases of the software development process (design, code, unit test, system integration, and validation and verification) better than the first three phases (concept development, system requirements allocation, and software requirements allocation). The data indicates there is room for improvement and opportunities for cross learning between the sectors studied.

**Figure 5.14: Application Domain Performance versus the Level of Value Added**

Two Questions are plotted here: "Estimate the value that each of the following contribute to developing software in a timely, cost effective approach to meet the users needs." versus "How well do you think your program executed the following phases of software development."

It is important to note that interviews with the military space ground terminal program identified an explanation for the low score associated with system and software requirement allocation. The process they use does not formally allocate system and software requirements because they do not use a software specification and their requirement changes are typically in software due to the adoption of commercial-off-the-shelf hardware.
5.2.1.2 Assure Seamless Information Flow

"Provide process for seamless and timeless transfer of and access to pertinent information" (LEM).

The importance of information in the requirement derivation process can not be overstated. Repeatedly process practitioners stressed the importance of accurate and accessible information. This effective process element maps directly to a LEM Overarching practice, Assure Seamless Information Flow.

Researchers have highlighted the importance of involving the customer and the end user in the concept development and requirement generation process [Ulrich and Eppinger, 95]. Involving the customer and the end user is one way to capture the latent and explicit needs of the stakeholders. The information the software developers require during the software requirement generation process is not limited to the information provided by the customer and the end user. However, the scope of the research effort was limited to understanding the information provided by the end users and customers.

The survey of the leadership focused on understanding the importance of six aspects of involvement, typically requiring end user or customer information (Appendix 1: Questions 3.6 and 3.7). The survey respondents were asked to evaluate the level of importance of these aspects with respect to the end user (Figure 5.15) and the customer (Figure 5.16). The most important aspect of the end user involvement was agreement between the end user and the developer on performance expectations. Survey respondents from all ten systems thought giving developers an understanding of the operational environment and system utilization were very important aspects of end user involvement. Survey responses from all but the commercial auto-pilot system thought building trust was a very important. Survey responses on three of the ten systems considered clarifying validation and verification somewhat important while the other seven considered this to be a very important aspect of end user involvement. Developing new business opportunities was considered somewhat important by all ten systems.
The same question asked with respect to the customer had significant variations in the responses across all ten systems as shown in Figure 5.16. One possible cause of the increase in the distribution of the responses could be the multiple definitions existing for the customer (Figure 5.2). The lack of a consistent definition of customer will most likely impact the responses to the survey question.

Agreement on the performance expectations, understanding the operational environment, and system utilization were all considered important aspects of customer involvement. The military aircraft programs, on average, valued the customer involvement in these areas more than the missile/munitions programs and the commercial auto-pilot. Commercial auto-pilot systems responses varied significantly from the other systems when it came to building trust and clarifying validation and verification criteria. The commercial system did not consider the customer involvement as important in these areas. Developing new business had the most diverse distribution of any category. The military space ground terminal upgrade, one missile/munitions program and one military avionics upgrade placed more importance on the aspect of involving the customer than the other systems. A complete representation of the responses can be found in Figure 5.16.
aspect of involving the customer than the other systems. A complete representation of the responses can be found in Figure 5.16.

**Figure 5.16: Average Level of Importance of Various Aspects of Customer Involvement**

How important are the following aspects of CUSTOMER involvement in the software requirement process?

- **Very Important**
- **Somewhat Important**
- **Not Important**

The three case studies illustrated need to utilize information provided by the customers and the end users. Information was obtained in meeting notes, specifications, e-mail, contract documents, and other documentation. In all three case studies, meetings with the end users and customers provided information and the product of these meetings was disseminated to the software developers. The method of dissemination and the apparent effectiveness varied across systems. The military avionics case study developed a very effective integrated database to capture and disseminate knowledge to software developers. The database was built using existing commercial tools (Framemaker) and was designed to be central repository of information so the software developers could access the key information when needed. Interviews with process participants revealed that the database was important because it provided a single location for the key information.
The database had four major sections: System Requirements Analysis, System Operation, Primary Flight Control Computer Changes, and Appendices.

1. **System Requirements**
   This section defined the individual software candidates and notes with the history of their origination. The notes included the meeting the candidate was introduced, who identified the candidate for implementation, and any relationship the candidate might have to existing field performance. Each software candidate has a description of the basic capability the candidate is intended to provide as well as references (web links) to the applicable documentation changes resulting from the implementation of the candidate.

2. **System Operation**
   The system operation section describes the major trade-offs and high-level functional flow diagram of each candidate, where applicable. The intent is to describe the high-level concept of operations of the each software candidate. This section also covers any special analysis required by law. Required changes to government furnished equipment and impacts to external interfaces are documented and links to lower level specifications are provided for each software candidate.

3. **Primary Flight Control Computer (PFC)**
   The primary flight control computer is the major component still under control of the aircraft developer. Required changes to the PFC resulting from the implementation of a software candidate by the aircraft contractor or external interface changes are captured here.

4. **Appendices**
   Any additional information required by the developer can be stored here.
The integrated database aided in the seamless transfer of information. During the case study interviews, the importance of the central repository of information became more clear not only to the researcher but to the contractor as well. By the time the case study was completed the developers had expanded access to the database to include other functions (logistics and test) and were looking to more efficiently capture the information resident in the database.

5.2.1.3 Develop Relationships Based on Mutual Trust and Commitment

"Establish stable and on-going cooperative relationships within the extended enterprise, encompassing both customer and suppliers" (LEM).

Trust was considered by many of the interviewees as the glue that kept the effective process together. The value stream elements (Chapter 4) illustrated how diverse the stakeholder community can be for an aerospace software upgrade. Often these stakeholders have conflicting requirements that need to be resolved through the course of the derivation effort. Trust was considered a critical factor that contributes to the ability of the process team to resolve disputes. The LEM practice, Develop Relationships Based on Mutual Trust and Commitment, maps directly to this belief.

Trust can be developed many different ways and is an important part of a healthy working relationship. The research objective for this Lean practice was to understand the role the requirement derivation process played in building trust between the end users and the software developers. Survey respondents were asked to evaluate the importance of various aspects of end user involvement (Figure 5.15) with building trust being one of the response categories. The responses for this category were isolated and illustrated in Figure 5.17.
Data suggests that the software requirement derivation process can be an important contributor to building trust in an organization. One observation from the case study interviews suggests that the level of end user involvement may also impact the program and process leadership’s perception of the relationship between end user involvement and building trust. The lowest average response shown in Figure 5.17 was on System 1, the commercial auto-pilot case study. This organization had a hierarchical relationship between the software developers and the end user (illustrated in Figure 5.12). The hierarchical relationship is in contrast to the more integrated military avionics case study (System 2) and the military space ground terminal case study (System 9). Case study interviews from both military programs observed a strong belief that the integrated team structure developed and fostered the mutual respect and trust element of an effective process team. The commercial auto-pilot team was not as integrated and the survey respondents did not view trust as an important aspect of end user involvement. Interestingly, the commercial process had the lowest level of end user involvement of the three case studies as shown in Figure 5.18.
Feedback from practitioners suggest the perception of the importance of building trust will increase as the end user gets more involved.

5.2.1.4 Maximize Stability in a Changing Environment

"Establish strategies to maintain program stability in a changing customer-driven environment" (LEM).

Over the course of the research project no single word was more closely associated with the requirement derivation process than "change". Virtually every process practitioner expressed a desire for the requirement process to be responsive to changes. This is one of the basic Lean meta-principles and the foundation for an overarching practice, Maximize Stability in a Changing Environment.

Perhaps no other phase of the software product development process is more susceptible to instability than the requirement derivation phase. During the formation of the research plan, practitioners communicated the need to better understand the software requirement derivation
process to help reduce the amount of changes to the requirement baseline. The survey respondents were asked to estimate the frequency that changes in end user and customer needs contributed to unplanned changes (Appendix 1: Question 2.7) in the software requirements.

The responses varied across each system and are illustrated in Figure 5.19. No category occurs "very often" indicating the perception of frequent requirement changes by the customer and the end user may be based more on perception than reality. The category with the lowest frequency of occurrence and the smallest variation was "software unable to meet requirements". Case study observations validated this finding. The software can meet the specified requirement but this does not automatically make the specified requirement correct. Requirements are sometimes omitted and added later. Although the question as asked dealt with frequency of change, perhaps a question dealing with impact of unplanned changes would have been helpful. Unplanned changes might not occur frequently, but might have a high impact.

Figure 5.19: Frequency of Unplanned Changes

The case studies offered additional insight into the variations between systems. The military space ground terminal (System 9) viewed the frequent changes in end user and customer needs
as a source of competitive advantage. The commercial auto-pilot (System 1) and the military avionics system (System 2) viewed the requirement changes unfavorably. The long software development cycle times and the relationships with the end user and the customer were observed as the root cause of differences in the ability of the developers to effectively manage the customer and end user changes.

The military space ground terminal prime software developer believed the frequent customer and end user changes were not a significant problem as long as the process supported the changes to the customer and end user needs. This opinion was shared on the military avionics program as well. Since the time of the case study, a process improvement initiative has been launched to develop such a process.

The military space ground terminal team developed a process to support changes and minimize the program risk associated with the changes. The process separated technical risk and non-technical risk. The technical risks associated with the frequent changes led the software developer to adopt a spiral model and introduce a continuous rapid prototype process. The prototype process allowed the customer and the end users to make changes to low cost software prototypes versus developing the software, incorporating the software into the product baseline and then make changes. Software prototypes are included into the software baseline after gaining customer and end user approval. By providing a stand-alone prototype, the contractor was able to preserve the integrity of the software baseline.

The military space ground terminal contractor shortened the software development cycle time to incorporate the software prototype into a seven-month spiral model (Figure 5.20). The shortened cycle time minimized the opportunity for changes to the software development baseline. The non-technical risks were minimized as well. A fixed amount of time limited the amount of money that could be spent and thus limited the possibility of large cost overruns. The team contractually decoupled the prototype effort from the seven-month development program. The separation of the two activities allowed the contractor and customer to effectively manage the risk associated with budget instability. Prototypes were deleted and added based on the level of
funding. The software baseline developed under the seven-month cycle time became protected from funding irregularities.

Figure 5.20: Military Space Ground Terminal Requirement Prototyping Process

The military ground terminal program manager viewed the combination of the spiral model, a shortened cycle time, and a continuous software requirement analysis activity as a source of competitive advantage. Software developers were encouraged to be creative and challenge the requirements in order to develop new value added features and missions to delight the end users. The program manager viewed this as the primary reason the program was able to achieve the highest level of end user satisfaction (Figure 5.4) among the ten systems studied.

The high level of end user satisfaction had positive impacts on the financial statement. The team received the highest percentage of profit of any government program within the company. In addition to increased profit, the process allowed the contractor to take advantage of new business opportunities. During the case study fact finding, the contractor received an unplanned $200K to explore the possibility of incorporating new requirements into the ground terminal. The military space ground terminal has been able to create a “win-win” environment for all stakeholders.
5.2.1.5 Maintain Challenge of Existing Processes

"Ensure a culture and systems that use quantitative measurement and analysis to continuously improve processes" (LEM).

At the time of the research all three case studies were engaged in requirement process improvement activities. The programs included in this research effort all considered process improvement an important contributor to their success. Although these efforts are too new to measure quantitatively, a short summary of each is included to indicate the presence of this LEM overarching practice.

The commercial auto-pilot team was not directly engaged in the process improvement activities but supported the corporate effort to improve processes. The aircraft manufacturer was heavily engaged in enterprise level improvements focused beyond the auto-pilot. Their activities focused on incorporating the tenants of Lean on the entire aircraft development effort.

The military avionics case study was actively involved with the government customer and end user to incorporate the tenants of Lean in avionics software development. Lean practices were being incorporated at the organizational level and all process steps were being challenged. Process activities focused on increasing the information flow between functions and developing new measure for requirement process effectiveness occurred as a result of synthesizing the research data collection with existing process improvement initiatives.

The military space ground terminal system was actively engaged in process improvement activities and was focused on increasing the ability to pass critical information on requirements “real-time” to software developers and system testers during the prototype process. The improvement activities are focused on increasing the quality of information to reduce cost and increase performance.
5.2.2 LEM Overarching Practices: Opportunities to Improve

Despite having a successful program, process practitioners on all three case study programs thought the process could be improved and matured to achieve greater levels of efficiency. Opportunities to improve were observed to align with four LEM overarching practices and are depicted in Figure 5.21.

Figure 5.21: LEM Overarching Practices: Opportunities to Improve

5.2.2.1 Continuously Focus on the Customer

"Proactively understand and respond to the needs of the internal and external customers" (LEM).

As mentioned earlier in Section 5.1, every system involved in the research expressed a strong belief that they are focused on the customer. The multiple interpretations of the term "customer" raises questions on the interpretation of a "customer focused" approach. More data is required to better understand the impact of having multiple definitions of customer on the overall requirement process effectiveness.
5.2.2.2 Identify and Optimize Flow

"Optimize the flow of products and services, either affecting or within the process, from concept design through point of use" (LEM).

One of the basic assumptions of process optimization is that the person or organization can measure the existing process. Program and process leadership were asked to estimate the cost and schedule (as a percentage of total cost and time) of each phase in the development effort (Appendix 1: Question 2.4). The research intended to collect this data in an attempt to look for areas where process phases could be optimized.

The responses from all ten systems indicated many of the survey respondents were either not comfortable estimating the percentage of cost and schedule associated with each software development phase or they did not know the information. Figure 5.22 shows the percentage of survey respondents who did not respond to this question for each system. When estimating cost, only three of the ten systems had more than 50% of the leadership who could respond to the question. Estimating schedule appears to be easier than cost. Of the ten surveyed systems, six had more than 50% of the leadership who could respond.

The inability to separate the schedule by phases is consistent with earlier data on program cycle time shown in Figure 5.5 which illustrated significant variability in the survey respondents estimates of the software development cycle time.
The case study interviews revealed several possible causes for the inability of the survey respondents to estimate the percentage of cost and schedule associated with each phase. First, the phases of software development listed in the survey question may not be consistent with the “real world”. Most of the systems utilize a concurrent software development process where multiple software baselines are being developed simultaneously. The concurrency makes separating the phases difficult.

Second, the cost data may not be well understood by all process participants. Cost data did not exist on some of the case study programs because of a lack of activity based cost accounting at lower levels. The first case study provided a clear example. Cost information for the commercial auto-pilot was collected by the finance department and was unknown to most of the auto-pilot developers (prime and subcontractor).

Finally, the start and finish dates of the software development efforts are not clear. The start of concept development is clouded because some early requirement evaluation work is often done
to evaluate the feasibility of incorporation requirements prior to the requirement phase. The end date of the development effort also appears confusing. In the military programs, the case studies identified the government as one cause of the confusion. In the military avionics upgrade, the software is delivered to the government operational test organization. Several of the industry respondents thought this was the end of the software development effort. The operational test organization is separate from the development organization and is chartered to make sure the software is suitable for release to the operation flying units. Some for the respondents felt this was the end of the development effort. The lack of quality data did not allow for a comparison of time and cost spent per phase and suggests that the task of optimizing flow can be very challenging.

5.2.2.3 Optimize Capability and Utilization of People

"Assure properly trained people are available when needed" (LEM).

Requirement practitioners repeatedly voiced frustration over the difficulties in completing requirement derivation process activities. One observation present in all three cases studies was the lack of formal training of the process participants. The process participants had a strong desire to perform better or more efficiently but felt they may not be adequately prepared for the job. Requirements work requires the knowledge of the development process, the application domain, and the end user. This knowledge will certainly increase over time. Another way to increase the knowledge is through the use of formal training.

Training is an important part of building an effective organization. Unfortunately, software personnel are poorly trained to handle the demands of the many diverse tasks associated with their job [Humphrey, 89]. During the formation of the research topic, practitioners expressed a desire to understand the adequacy of software requirement training. The survey respondents were asked to evaluate the effectiveness of multiple training methods (Appendix 1: Question 1.3).

The survey responses shown in Figure 5.23 showed little variation across systems or disciplines. The data indicated formal training does not appear to be effective. On average, survey
respondents thought university training, professional training, in-house, and university training was somewhat ineffective.

Figure 5.23: Estimated Level of Formal Training Effectiveness

As a whole, how effective have the following training programs been in helping you perform your duties in the software requirement derivation process?

More than 40% of the respondents found professional training programs, professional conferences, and university training programs "Not Applicable" (NA). Most only had experience with on-the-job training and in-house training (defined as government training for government personnel and contractor training for contractor personnel). A breakdown of the "NA" responses is shown in Figure 5.24.
On-the-Job training was the highest response for all systems and the only response considered as effective. The case study interviews suggested the formal training programs might not be effective due to the lack of consistent complexity associated with software upgrades and the application domain specific nature of software requirements. Program complexity also appears to vary with each system and with each release. The changing nature of the content and the interface with external systems requires the requirement derivation process to be flexible and adaptive in order to develop the technology to meet the end user needs. The case study interviewees thought teaching requirement derivation techniques might be difficult under these conditions. One interviewee suggested that university, professional and conference programs might be more effective if they included real applications in their curriculum.

5.2.2.4 Make Decisions at the Lowest Level
Past research has indicated a good software requirement should be cost effective, meet the end user needs, complete, and traceable. The program and process leadership was asked to identify the discipline or individual with primary responsibility for each of these four software requirement elements. The focus was on understanding if the person or function with the primary responsibility was at the lowest possible level in the organization.
The United States Air Force has found system test to be almost 50% of the total cost of the software upgrade for a typical avionics software upgrade [Gregory, 1998]. Data in Figure 5.25 indicates that despite the large percentage of cost associated with test, not one of the survey respondents identified systems test as having primary responsibility for cost. The survey responses did not vary by system and were aggregated to reflect the opinions of the respondents in general.

Figure 5.25: Primary Responsibility for Cost and Performance

Project management was considered to be the discipline or individual with the primary responsibility for assuring the software requirements are cost effective. A distant second response was the chief systems engineer. Case study interviews and survey data indicated the responsibility for cost is a program management responsibility. The requirement analysis team identifies the cost, schedule, and performance impacts associated with implementing a particular requirement and project management decides whether these parameters are acceptable. The
largest number of responses fell into six areas: chief engineer, program management, end user, customer, system test, and systems engineering.

The distribution of the responses for assuring the requirements are complete and traceable to system requirements was found to be similar and is shown in Figure 5.26.

**Figure 5.26: Person(s) Primarily Responsible for Making the Requirements Complete and Traceable**

The case study interviews indicated the responsibility for making sure the requirements are complete and traceable was often shared between the chief systems engineer, systems engineering, and software development.

**5.2.3 LEM Overarching Practices: Future Improvements**

The final part of the organizational analysis focused on the future improvements to the requirement process. Three LEM overarching practices associated with future improvements are shown in Figure 5.27.
The LEM practices associated with future improvements were not characterized with data. Each of these Lean practices requires an established level of understanding of the Lean principles and concepts. At the time of the research, this level of understanding was being matured and developed and as a result future research is needed to quantify and characterize how to implement these practices.
Chapter 6: Summary

The Lean Aerospace Initiative industry and government members have expressed a strong desire to investigate processes for deriving software requirements. To the authors' knowledge this is the first research effort attempting to apply the Lean principles and the Lean Enterprise Model with the software requirement derivation process. Data used to support this research are backed by a comprehensive two-year research effort involving three detailed case studies with 45 case study interviews, 125 stakeholder surveys, and feedback from countless industry practitioners and MIT faculty. The systems involved in the research effort have been based on successful programs and represent four application domains: commercial aircraft, military aircraft, missile/munitions, and a military space ground terminal.

Lean Aerospace Initiative consortium involvement has been invaluable in interpreting survey responses and clarifying case study observations. The support from MIT, government, and industry is a recognition of the importance of the requirement derivation process to the product development process. Errors made early during the requirement derivation process cascade through the entire product development process and become an order of magnitude or more expensive to fix over time. Incorporating the principles of Lean and the Lean Enterprise Model offers organizations the opportunity to become more effective and efficient in deriving software requirements.

6.1 Case Studies

All three case studies used a software requirement process they believed to be effective. Each offered benefits and challenges associated with the process they chose to adopt. No single process emerged as a panacea, but they all have tenants that could be applied if they are properly understood.

6.1.1 Commercial Auto-pilot Software Upgrade Case Study

The most unique aspect of the commercial auto-pilot case study was the relationship between the software developers, customers, and end users. Unlike the other case studies, the aircraft contractor has complete control over the development environment. The control over the development environment allowed the developers to have the opportunity to have direct access
to end user representatives and test aircraft. The process appeared to draw strength from on-site surrogate end user representatives who could be made available to provide rapid feedback. A second observed benefit from the process appears to be the focus on a “hallmark” requirement, flight safety. None of the three case studies had as strong of a focus on one functional or performance based requirement as the commercial auto-pilot case study. A hierarchical relationship allowed the organization to have a clear single requirement focus but limited the ability to optimize the management of the requirement process. The large number of documents and the size of the document made the requirements process very labor intensive and increased rework of requirements.

The process also created some challenges for the developers. Easy accessibility of surrogate end users can make the organization complacent about including the “real” end users in the process. The commercial auto-pilot system had the lowest level of end user involvement in the concept definition phase, shown in Figure 5.18.

6.1.2 Military Avionics Software Upgrade Case Study

The military avionics requirement process was the most formal and well documented of the three case studies. The structured and repeatable process appeared to be a highly effective approach to synchronizing the large number of value stream elements and external interfaces involved in the development effort. The military avionics case study had the largest number of value stream elements of the three case studies. A less formal structure may make coordination between the value stream elements very difficult. The military avionics process allowed the requirement derivation team, which includes the external value stream elements, to focus on delivering capability versus code.

The second observed benefit from the military avionics process is the ability of the organization to improve the process. The formal nature of the process and the well document requirement approach allows the developer to have access to a significant amount of data necessary to make informed process improvement decisions.
The underlying challenges of the military avionics software upgrade requirement process is centered around the relationship between the software developers and the government customer and end users. Software developers are dependent upon the government for end user input to the requirement process and validation and verification. This dependency can make executing the requirement derivation process a challenge. The end user input is critical to the development of the software requirements and the inability of the software developer to have direct access to the end user increases the amount of effort required to facilitate the relationship.

6.1.3 Military Space Ground Terminal Case Study
The military space ground terminal had a high level of stakeholder involvement that contributed to the software developers ability to shorten the software development cycle time. The seven-month cycle time provided the customer and end users the flexibility to change requirements as the “real world” changed. A shortened cycle time and the ability to add requirements quickly provided new business opportunities for the contractor. The relationship between adding new requirements and new business is one of the observed drivers behind the focus of the organization on new capability and new missions for the requirement process.

The space ground terminal requirement process is tightly coupled to the tacit knowledge of the organization. Process participants rely on the experience of the organization to be successful. The lack of documentation makes the organization sensitive to changes in personnel and staff and was identified in the case study interviews as a future concern.

6.2 Lean Methodology and the Lean Enterprise Model
The integration of Lean practices in the research approach proved to be beneficial. Value stream mapping provided valuable insights into the software systems. The military avionics value stream mapping (Figure 4.4) illustrated the need to think of aerospace software upgrades as much more than software code. The value stream representation correlates with data shown in Figure 5.8 and Figure 5.9 that estimates the software code as less than six percent of the total cost of a typical software upgrade. The low percentage of cost associated with the code generation suggests a more “systems” orientated approach to software upgrades may be
appropriate. Current methods for managing the interaction between value stream elements, integrated schedules and interface control documents, may be incomplete. Information technology tools to track information flow and identify dependencies between value stream elements may be needed for software upgrade programs with large numbers of value stream elements.

The combination of case studies, stakeholder surveys, and the Lean Enterprise Model (LEM) provided an effective framework for understanding the relationship between Lean practices and the software requirement derivation process at the enterprise and organizational levels. An enterprise analysis of the three case studies highlighted the lack of effective enterprise process measures. Despite researching only successful programs, quantitative metrics on software development cycle time, customer satisfaction, and end user satisfaction were not observed. The program leadership was unable to estimate the software development cycle time on two programs while others had significant variations in responses. Existing measures of requirement volatility and lines of code metrics appear to inappropriately represent the total effort required to upgrade aerospace software systems. Requirements rework and the more subjective customer and end user satisfaction, may give better insight in how to measure aerospace software upgrades that have multiple value stream elements.

The organizational analysis utilized the twelve Overarching Practices identified in the LEM. Case study interviews identified aspects of the requirement derivation process that worked well, elements that could be improved, and candidates of future improvements. Comments by practitioners were aligned with LEM practices and observed to fall into three categories: Effectively Implemented, Opportunities to Improve, and Future Changes.

Five Lean practices were observed as Effectively Implemented. The data from these LEM practices identified practices that worked well and have been effectively implemented by some of the process practitioners.

1. **Implement Integrated Product & Process Development (IPPD):** Two aspects of IPPD were analyzed. First, the military space ground terminal software upgrade program
represented a clear example of a highly integrated "Emerging" Lean organization. Suppliers were collocated with developers and a strong relationship between the government customers and end users helped facilitate cycle time reductions. Second, all phases in the integrated process add value to developing software. Practitioners believe the phases towards the end of the product development process are being accomplished well (coding, system integration, and validation and verification). Data shows that the front-end requirement phases (concept definition and requirement allocation) could be improved. Practitioners could focus process improvement activities in these areas to increase efficiency.

2. **Assure Seamless Information Flow:** The military avionics software upgrade provided an excellent example of how to utilize integrated databases to support the requirement derivation process. Requirement information was centrally located and by the end of the case study research, accessible to all stakeholders. One of the primary benefits of the information captured in the database is the ability to document the end user input into the requirement derivation process. Data in Figure 5.15 suggests agreement between the end user and the developer on performance expectations was the most important aspect of the end user involvement during the requirement derivation process. All ten systems thought giving developers an understanding of the operational environment and providing a description of how the system will be used were important information elements provided by the end user during the requirement process.

3. **Develop Relationships Based on Mutual Trust and Commitment:** Building trust between end users and developers was identified by process practitioners as an important enabler to developing an effective process. Data shown in Figure 5.17 suggests the requirement derivation process was an effective method for building trust. Most of the systems surveyed believed the end user involvement in the requirement process was "very important" in building trust between software developers and end users.

4. **Maximize Stability in a Changing Environment:** The military space ground terminal was very successful in utilizing a rapid prototype and continuous requirement derivation
process to increases responsiveness to the government customer and end users while reducing the software development cycle time. The feedback from the case study interviews suggests this process is a good example of Lean meta-principles effectively implemented in a software requirement process.

5. **Maintain Challenge of Existing Process:** At the time of the case study research all three case studies were involved in requirement process improvement activities. Feedback from practitioners suggested that more effort is needed to change the way the organization accomplishes the requirement derivation process. The desire to change the process was most visible on the military avionics system where the contractor and end user were actively involved in process improvement activities to streamline the requirement process and reduce the software development cycle time.

Four Lean practices were observed as current process elements that could be improved and were placed into a category called *Opportunities to Improve*. Data from case study interviews and the stakeholder survey were used to provide examples for each of the LEM practices considered by practitioners as process elements needing improvement.

1. **Continuously Focus on the Customer:** The multiple definitions of the term customer raises questions regarding the impact of not focusing the organization on one customer. More research is required to understand the impact of multiple customer definitions on the overall requirement process effectiveness.

2. **Identify & Optimize Enterprise Flow:** The data in Figure 5.22 suggest program and process leadership on many of the aerospace software upgrades are not able to easily estimate the cost and schedule distribution of the software development program. The inability to make these estimates appears to be linked to two causes. First, “academic” software models outlining software development phases do not represent the “real world” where many process activities are done concurrently. Second, the complete cost and schedule information is not well known to many of the process participants. In either case, none of the three case studies were observed as having a cost and schedule
information understood by all process participants. The large number of value stream elements was observed to further complicate this task.

3. **Optimize Capability & Utilization of People:** The data in Figure 5.23 suggests formal training methods are not effective. The review of university, professional, in-house, on-the-job training, and professional conferences found on-the-job training was currently the only effective training program. Feedback from industry suggested formal training programs might increase their effectiveness by incorporating aspects of on-the-job training in their curriculum, but more research will be required.

4. **Make Decisions at the Lowest Level:** The data shown in Figure 5.25 and Figure 5.26 suggest the primary responsibility for making sure requirements are complete, traceable and meet the end user needs does not fall under any one discipline or individual. The primary responsibility for making sure requirements are cost effective was associated with Program Management. The lack of primary responsibility for cost and performance associate with the software and systems engineers suggest these elements are not yet fully empowered down to the working level. Previous research has suggested empowering designers with cost data can be a powerful enabler in reducing product cost [Hoult, Meador, Deyst, and Berry-Dennis, 1998]. Further research is required to understand the impact on software development.

The last grouping of Lean practices, *Future Changes*, represented practices that were not observed but available for future implementation. The three overarching practices require a level of maturity and understanding of Lean principles that we have yet to see on many programs. This offers practitioners hope that we can still push further in our attempts to improve the requirement process and develop Lean products that have higher quality, shorter cycle times, and lower cost.
Chapter 7: Future Research

Future efforts can be categorized into three areas. First, the author intends to extract data from this research effort and generate recommendations for the incorporation of Lean practices and principles for the process used to derive aerospace software requirements. Recommendations will be made through the Lean Aerospace Initiative at MIT in the spring of 2000.

Second, the research method (discussed in Chapter 3) proved to be very successful and future research should consider adopting this approach to look at other phases of the product development process. The method used to collect data proved to be one approach to understanding Lean principles and practices outside of the traditional manufacturing domain.

Third, the enterprise and organizational analysis highlighted several areas where existing methods may not be appropriate or could be improved. This thesis provides the opportunity for future researchers to continue this effort in an attempt to improve the enterprise metrics and Lean practices for deriving aerospace software requirements.
Appendix 1: Survey of Process and Program Leadership
Survey to evaluate Effective Practices for Establishing Software Requirements from System Requirements

Confidential Questionnaire

Massachusetts Institute of Technology
Research Sponsored By Lean Aerospace Initiative
This survey is part of an on-going research project by a consortium involving the U.S. Air Force, a number of firms in the defense aerospace sector, and Massachusetts Institute of Technology. The research projects focus on the investigation of the application of "Lean" practices in the defense aerospace industry. This survey is designed specifically to characterize the process by which software requirements are derived for real-time, mission critical, embedded software systems.

Your cooperation is vital to the success of this study! Please answer the following questions as they apply to you. We have tested this survey and expect it to take approximately 30-45 minutes to complete.

Please be candid and honest in your responses. We understand that you may have concerns about confidentiality. Several measures will be taken to ensure that your responses will remain confidential. Only the researcher named at the end of the survey will have access to the information requested in this survey. All analysis of the survey data will be presented in the form of aggregated statistics. No individuals or individual programs will be identified in the analysis or reporting of the responses. We understand that the success of any research depends upon the quality of the information on which it is based, and we take seriously our responsibility to ensure that any information you entrust to us will be protected.

We would prefer that you complete the survey and return it to MIT as soon as possible, but to ensure timely and complete reporting, please return the survey no later than three weeks after the date the survey was received. Surveys can be sent to:

Mr. Brian Ippolito
Lean Aerospace Initiative
Massachusetts Institute of Technology
Room 33-407
77 Massachusetts Avenue
Cambridge, Ma 02139-4307
Email: ippolito@mit.edu
Fax: 1.617.258.7845
Phone: 1.617.441.0486

<table>
<thead>
<tr>
<th>Respondent Name</th>
<th>Program Name</th>
<th>Company/Organization</th>
<th>Job Title</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
</table>

123
1.0 Assuming that the “Customer” is defined as the organization/person to whom you deliver your product and “End User” is defined as the person who operates the system. For your project, which of the following categories best fits the terms "Customer" and "End User". 

<table>
<thead>
<tr>
<th>Definition</th>
<th>Customer</th>
<th>End User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime Contractor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Government System Program Office</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Another unit within your organization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Airline Pilot</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warfighter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Pick only one for each category)

1.1 Which of the following generic software development processes best describes the process that your program currently uses?

- **Waterfall**: Software developed sequentially. Objective is to eliminate rework problems by defining requirements at Program start and then follow a sequential development process. Design starts after requirements analysis complete and coding begins after design is complete.

- **Incremental**: Develop and code the system in a sequence of incremental builds. Design begins when a large % of the requirements are understood but before software requirements analysis is complete. Requirements are added through incremental releases that culminate with a final product release.

- **Spiral**: Risk-based prototyping used to mature requirements. Feedback from prototypes used to mature the software requirements during the development process. Design, Code and Testing begins when a small % of the requirements are understood.

1.2 Which step in the software requirements process do you work? (check all that apply)

- **Concept Definition**: Define system level requirements, determine software architecture and evaluate system trade-offs

- **Sub-system Allocation**: Allocate system requirements to sub-systems

- **Hardware and Software Requirements allocation**: Allocate sub-system requirements to hardware and software

- **Software Requirement Analysis**: Define software requirements and software capabilities

- **System to Software traceability**: Tracing software requirements to top level system requirements
1.3 As a whole, how effective have the following training programs been in helping you perform your duties in the software derivation process?

<table>
<thead>
<tr>
<th>Training Programs</th>
<th>N/A</th>
<th>Not Effective</th>
<th>Somewhat Effective</th>
<th>Highly Effective</th>
</tr>
</thead>
<tbody>
<tr>
<td>University Training programs</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Professional Training Programs</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Government Training Programs</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Company Training Programs</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>On-the-job Training</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Professional Conferences</td>
<td>N/A</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

2.0 Estimate the value that each of the following contribute to developing software in a timely, cost effective approach to meet the users needs.

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>Low Contribution</th>
<th>Some</th>
<th>Critical Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Development: Explore the role of computer resources within the system. Define system level requirements.</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System Requirements Allocation: Allocate system requirements to software sub-systems</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirements Analysis: Evaluate software requirements for completeness, understandability, validity, consistency and adequacy of content</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Design, Code and Unit Test: Top-level and detailed design. Translation of design into computer instructions and complete unit testing.</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Systems Integration: Test and Integrate software components into overall system</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Validation/ Verification: Test the system in an operational environment to ensure end user needs are meet as well as prove the system meets the system requirements</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
2.1 How well do you think your program executed the following phases of software development.

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>Not Very Well</th>
<th>Average</th>
<th>Very Well</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Development</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System Requirements Allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirements Analysis</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Design, Code and Unit Test</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Systems Integration</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Validation/Verification</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

2.2 Estimate the value each of the following phases of the requirements process contribute to the successful derivation of software requirements from system level requirements?

<table>
<thead>
<tr>
<th>Phase</th>
<th>Low Value</th>
<th>Some Value</th>
<th>High Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Definition: Define system level requirements and evaluate system trade-offs</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sub-system Allocation: System requirements allocated to sub-systems</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hardware and Software Requirements allocation: allocation of sub-system requirements to hardware and software</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirement Analysis: Define software requirements and software capabilities</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System to Software traceability: Tracing software requirements to top level system requirements</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
2.3 On your program, how many months is your software development cycle? (Defined as concept definition to delivery of the product to the customer) _________ Months

2.4 On your program, approximately what % of the total software development cost and program time (in calendar months) is in each of the following phases? (your responses should total 100%)

<table>
<thead>
<tr>
<th>Phase</th>
<th>Cost</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Requirements Allocation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Requirements Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design, Code and Unit Test</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems Integration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validation/Verification</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Unaware of this information

2.5 On your program, approximately what % of the total software development cost is associated with rework of the software requirements derived from system requirements? _________ %

2.6 On your program, relative to the total # of software requirements derived from system requirements, estimate the % of the unplanned derived software requirements changes (additions, modifications, deletions) during the software development effort? _________ %

2.7 For your program, estimate the frequency in which the following factors contribute to unplanned changes (additions, modifications, deletions) to the derived software requirements.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Not Often</th>
<th>Sometimes</th>
<th>Very Often</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes in End Users needs</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Changes in End User expectations</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Changes in Customer Needs</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Changes in Customer Expectations</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software unable to meet requirements</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
3.0 For your program, estimate the level of involvement of the following disciplines/individuals in deriving software requirements.

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Not Involved</th>
<th>Somewhat</th>
<th>Very Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chief Systems Engineer</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Project Mgt</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>End User</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Customer</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System Test</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>External Interfaces</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Internal Interfaces</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Developers</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hardware Developers</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Systems Engineering</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Data Management</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

3.1 In your organization, who generally initiates process improvements for the software requirements derivation process? (check all that apply)

- Functional Mgt
- Chief Systems Engineer
- Project Mgt
- End User
- Customer
- Systems Engineering
- Other
- We do not do software requirements process improvements

3.2 For your program, estimate the level of involvement of the END USER in the following software requirements phases.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Not Involved</th>
<th>Somewhat</th>
<th>Very Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Definition</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sub-system Allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hardware and Software Requirements allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirement Analysis</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System to Software traceability</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
3.3 For your program, estimate the level of involvement of the CUSTOMER in the following software requirements phases.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Not Involved</th>
<th>Somewhat Involved</th>
<th>Very Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Definition</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sub-system Allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hardware and Software Requirements allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirement Analysis</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System to Software traceability</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

3.4 For your program, estimate the level of satisfaction of both the end user and the customer.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Not Satisfied</th>
<th>Somewhat Satisfied</th>
<th>Very Satisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>End User</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

3.5 For your program, evaluate the extent of which modeling and simulation tools are used to help with the following elements of the software requirements derivation process.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Not Used</th>
<th>Somewhat Used</th>
<th>Heavily Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept Definition</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Sub-system Allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Hardware and Software Requirements allocation</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Software Requirement Analysis</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>System to Software traceability</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
3.6 How important are the following aspects of **END USER** involvement in the software requirements process?

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Not Important</th>
<th>Somewhat Important</th>
<th>Very Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement between end user and developer on performance expectations</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Give developers an understanding of the operational environment of the system</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Give developers an understanding of how the system will be used</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build trust between developers &amp; end users</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clarify Validation/Verification Expectations</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop New Business Opportunities</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.7 How important are the following aspects of the **CUSTOMER** involvement in the software requirements process?

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Not Important</th>
<th>Somewhat Important</th>
<th>Very Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agreement between customer and developer on performance expectations</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Give developers an understanding of the operational environment of the system</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Give developers an understanding of how the system will be used</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build trust between developers &amp; customers</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clarify Validation/Verification Expectations</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Develop New Business Opportunities</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>1 2 3 4 5 6 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.8 For your project, which disciplines/individuals have the primary responsibility for assuring the software requirements are cost effective, traceable to system requirements, complete and meet the end users needs?  (Check Only 1 Per Column)

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Cost Effective</th>
<th>Traceable to System Requirements</th>
<th>Complete</th>
<th>Meet End user needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chief Systems Engineer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Mgt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>End User</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>External interfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal Interfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Developers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Systems Engineering</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE END

If you have any questions about this survey or the objectives of this research, feel free to contact:

Mr. Brian Ippolito

Lean Aerospace Initiative

Massachusetts Institute of Technology

Room 33-407

77 Massachusetts Avenue

Cambridge, Ma 02139-4307

Email: ippolito@mit.edu

Fax: 1.617.258.7845

Phone: 1.617.441.0486

For more information about the Lean Aerospace Initiative you can visit our web page at:

http://web.mit.edu/lean

Thank You Very Much!
References


Bozdogan, Kirk, and Deyst, John Jr., “Supplier Integration into Design and Development”, *LAI Executive Board Presentation*, 13 Nov 96

Brooks, Frederick P., Jr., *The Mythical Man Month*, Addison Wesley, 1995


Robertson, Suzanne and James, *Mastering the Requirements Process*, Addison Wesley, 1999


