Improving the Software Upgrade Value Stream

Prepared by:
Mr. Brian Ippolito
Professor Earll Murman
Massachusetts Institute of Technology

September 2001
RP01-01

IMPROVING THE SOFTWARE UPGRADE VALUE STREAM

Mr. Brian Ippolito[1]

Professor Earll Murman,

Massachusetts Institute of Technology, Cambridge, MA 02139

Overview

This paper reports findings from a two-year study [1] to identify Lean practices for deriving software requirements from aerospace system level requirements, with a goal towards improving the software upgrade value stream. The study was undertaken as part of the MIT Lean Aerospace Initiative. Three detailed case studies and 128 surveys collected from ten "successful" mission critical aerospace software upgrade programs support seven major findings:

1. Aerospace software upgrades are dependent upon multiple, interacting value streams.
2. Lean Enterprise Metrics for the end-to-end software development process are deficient.
3. Although all phases of the software development process are deemed to add value, they are not accomplished with the same level of effectiveness.
4. Information provided by the end user is considered "very important" by the requirement practitioners and needs to be captured through end user involvement early in the development process.
5. The responsibilities for assuring that the software requirements "meet the end user needs" and are "cost effective" are divided among different process owners.
6. Data suggests positive correlation between reduction in unplanned requirement changes and leadership continuity in both concept definition and requirement analysis phases.
7. Formal training for aerospace software requirement practitioners does not appear to be highly effective or abundant.

Background

The Lean Aerospace Initiative (LAI) is a government, industry, labor, and academic consortium focused on identifying and implementing the principles of Lean in the aerospace industry. Lean originated in the automotive industry and is grounded in the manufacturing domain. To the authors' knowledge, this is the first research specifically designed to apply the Lean principles and the Lean Enterprise Model to the process of deriving aerospace software requirements. The basic principles of Lean are focused on employing value added activities to reduce product cycle time, increase quality, reduce cost, and increase stakeholder satisfaction.

Deriving software requirements are an important step in developing aerospace software 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 [2]. The cost to correct errors made early in the phases of product development grows exponentially the longer they go undetected [3]. Analyzing the requirement process using a Lean framework allows practitioners the opportunity to improve the process to reduce the possibility of adverse system impacts.

The Lean Enterprise Model (LEM) [4] is a framework that incorporates enterprise level lean principles and practices, together with supporting data. 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

[1] Present address, Northrop Grumman, Electronics Systems and Sensors Sector, Baltimore, MD

measures. The twelve Overarching Practices identified in the LEM [4] were used as a guide for analyzing the presence of effective Lean practices for deriving software requirements.

The findings identified in this paper are the product of a comprehensive two year research effort involving three detailed case studies consisting of 45 interviews, in support of 128 stakeholder surveys collected from 10 aerospace systems, feedback from numerous aerospace industry practitioners, and Massachusetts Institute of Technology faculty and students. All 10 aerospace systems were real-time mission critical software upgrades representing four application domains (military aircraft, military space ground terminal, commercial aircraft, and missile/munitions). LAI industry and government consortium members selected the systems involved in the research effort. Each system is believed to be one that had a successful derivation of software requirements. Table 1 summarizes some of the discriminatory characteristics of the systems used in the research.

	Application Domain	Software Development Cycle Time (months)	Case Study	Stakeholder Survey Responses	Software Development Model Utilized
System 1	Commercial Aircraft	32	✓	13	Incremental
System 2	Military Fighter Aircraft	62	✓	14	Waterfall/ Incremental
System 3	Military Fighter Aircraft	49		12	Waterfall/ Incremental
System 4	Military Bomber Aircraft	46		15	Waterfall/ Incremental
System 5	Military Missile/ Munition	Unknown		7	Waterfall/ Incremental
System 6	Military Missile/ Munition	84		11	Waterfall/ Incremental
System 7	Military Missile/ Munition	29		14	Waterfall
System 8	Military Missile/ Munition	Unknown		13	Incremental
System 9	Military Space Ground Terminal	7	✓	18	Spiral
System 10	Military Fighter Aircraft	36		11	Waterfall

Table 1: Discriminatory Characteristics of the Ten Aerospace Software Upgrades Used in the Research

Research Findings

Finding 1: Aerospace software upgrades are dependent upon multiple, interacting value streams.

A value stream analysis and a detailed description of the requirement derivation processes completed on three aerospace software upgrade case studies indicated aerospace software upgrades are a multi-discipline and multi-organizational development effort. Figure 1 represents a high-level representation of a military avionics upgrade value stream. It identifies 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. The military avionics software upgrade requires changes not only to the avionics software code but changes to technical orders, support equipment, aircraft sensors, weapons, tactics trainers, and government certification. All of these elements represent multiple independent value streams that require updates prior to fielding the capability. The effective and efficient capture of the stakeholder needs is a challenge for the requirement derivation process owners.

Case study data suggest software code generation is less than 10% of the total software development program cost (Figure 2). To characterize the software upgrade process as simply a "code generation" exercise would be an oversimplification and an inaccurate representation of the complete set of activities associated with the

development effort of the military avionics upgrade program.

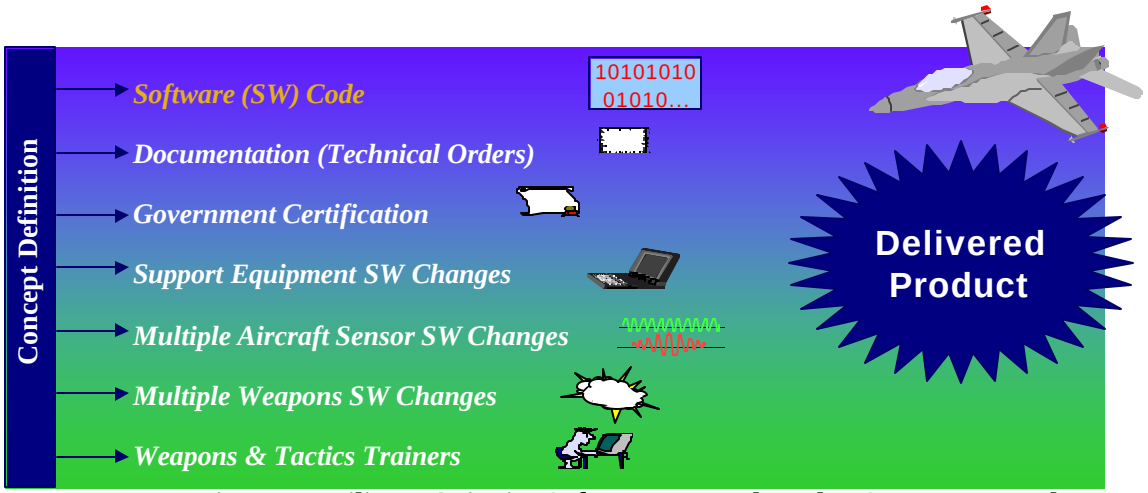

Figure 1: Military Avionics Software Upgrade Value Stream Example

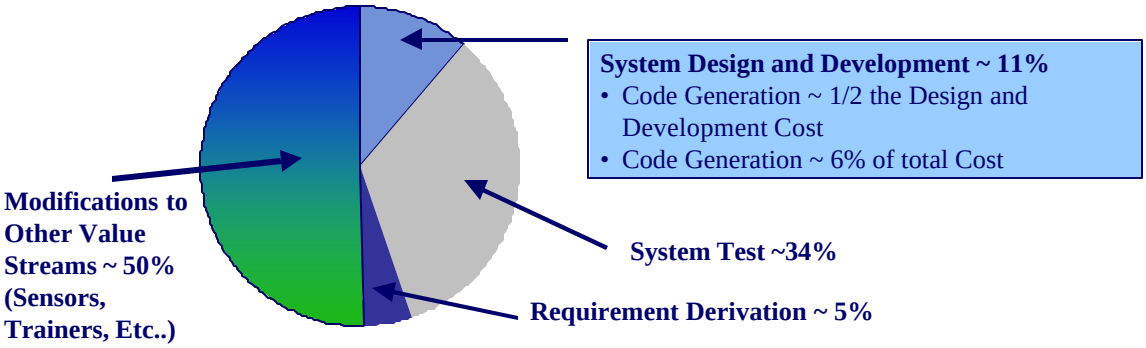

Figure 2: Estimated Cost distribution of the major end-to-end process activities for a military avionics software upgrade (based on historical data).

Finding 2: Lean Enterprise Metrics for the end-to-end software development process are deficient

At the enterprise level, LEM 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 investigated, the case studies failed to identify effective enterprise metrics for either the end-to-end software development or the requirement processes. Metrics used to measure the software development cycle time, customer satisfaction, end user satisfaction, and quality yield were deficient. Metrics measuring aspects of quality yield were found, but failed to capture the relationship between the requirement rework and the impact on the enterprise. Resource utilization measures were observed. but were focused on productivity associated with the generation of code. Quantitative measures for stakeholder satisfaction and the product development cycle time were not observed. A more detailed look at each of these enterprise metrics provided a deeper understanding of the importance of establishing enterprise measures.

Flow Time: For the software upgrade effort, the authors interpreted flow time to be the total program development cycle time (defined as concept definition to delivery of the product to the customer). Having a product development cycle time measure is important for organizations focused on delivering products faster, at a lower cost, and

with increased performance. Product cost and performance design trades, with respect to total time, are difficult to make if the product development cycle time is unclear or unknown. The program and process leadership were asked to define the software development cycle time for their program [1]. The results, shown in Figure 3, indicated that the end-to-end cycle time is not well understood by the process leadership. Furthermore, the end-to-end cycle times for most software upgrades are longer than three years. Respondents from two programs were unable to identify their product development cycle time (Figure 1).

Several factors contributed to the leadership's inaccurate estimates of cycle time: in some cases start and end dates of the product development process appear to be clouded by multiple program restructuring events; poor visibility into the value stream elements; and limited developer control of the major process activities (e.g. operational testing) at the beginning and end of the development process.

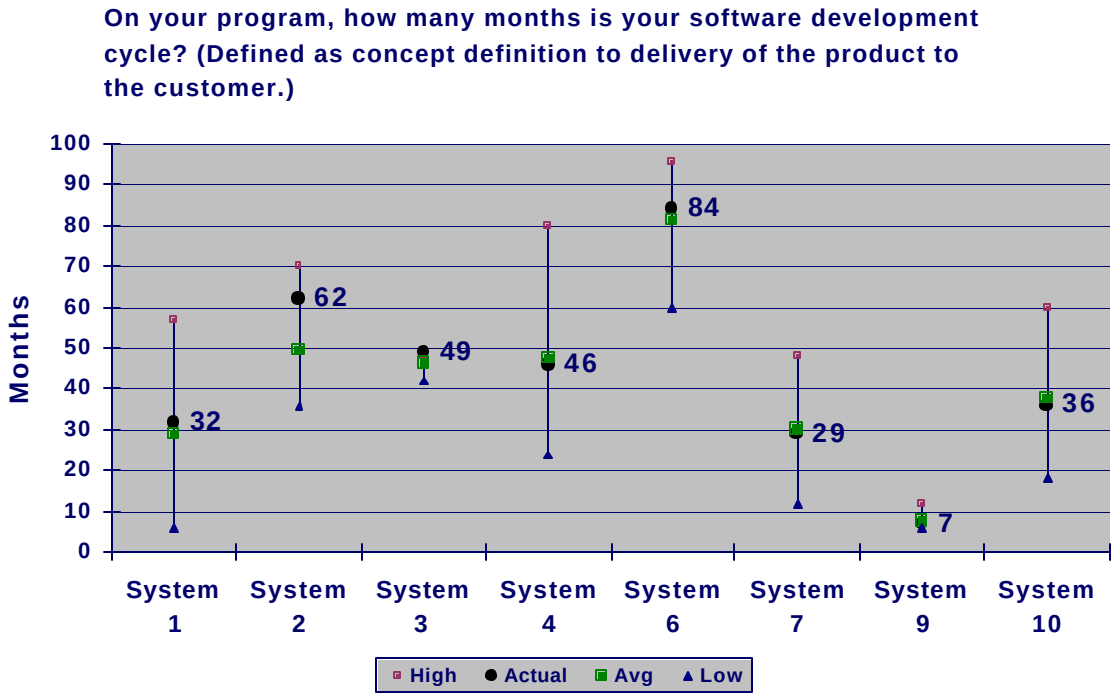

Figure 3: Software Development Cycle Time Responses

Stakeholder Satisfaction: Quantitative measures of stakeholder satisfaction were not observed. Overall, the subjective estimation of customer and end user satisfaction were very high. This is not a surprise given that the research focused on successful programs. The lack of customer and end user satisfaction measures suggests that more research is needed in this area to define effective enterprise metrics for software programs. Qualitative measures are subject to organizational issues and individual interpretation and may not be appropriate for multi-million dollar software upgrades.

Survey data indicated the program and process leadership had a uniform definition of the end user, shown in Figure 4. Interestingly this is NOT true for the definition of the customer. Multiple definitions of the customer existed and appear to vary based on the application domain as shown in Figure 5.

Caution should be taken when the term "Customer" is used to describe a person or organization since it is clear this definition has different interpretations.

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”.

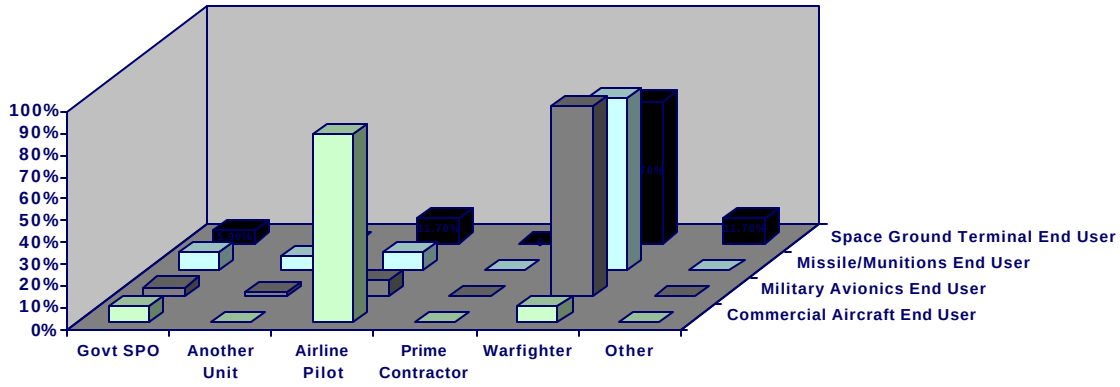

Figure 4: Defining the End User

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”.

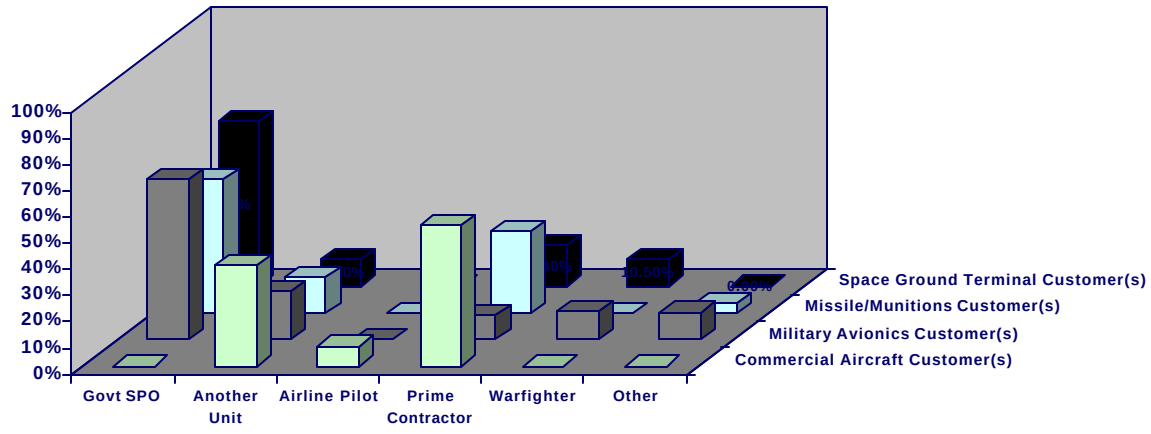

Figure 5: Defining the Customer

Resource Utilization: Case study interviews identified that the only measures of resource utilization employed in practice were associated with the code generation. The presence of such a large number of value stream elements and the relative small cost of changing the avionics software code (Figure 2) suggests software upgrade programs should be thought of as much more than just a coding effort. Metrics associated with lines of code should only be used for the effort associated with code generation NOT the entire development effort. Appropriate measures for the entire development were lacking and need to be developed.

Quality Yield: Metrics associated with rework of the requirements were observed on all of the case studies. All three programs tracked the additions, deletions, or modification of the software requirements. None of the programs were using a measure to understand how much of the rework was unplanned or the impact of the unplanned rework to the enterprise. Two new measures were developed by the authors to identify the amount of rework and the impact of this rework on the total software development cost.

First, the program and process leadership were asked to estimate the percentage of the total software development cost associated with the rework of the software. On average, unplanned rework was estimated to cost 16% of the total software development cost. A breakdown of the responses for all systems is shown in Figure 6.

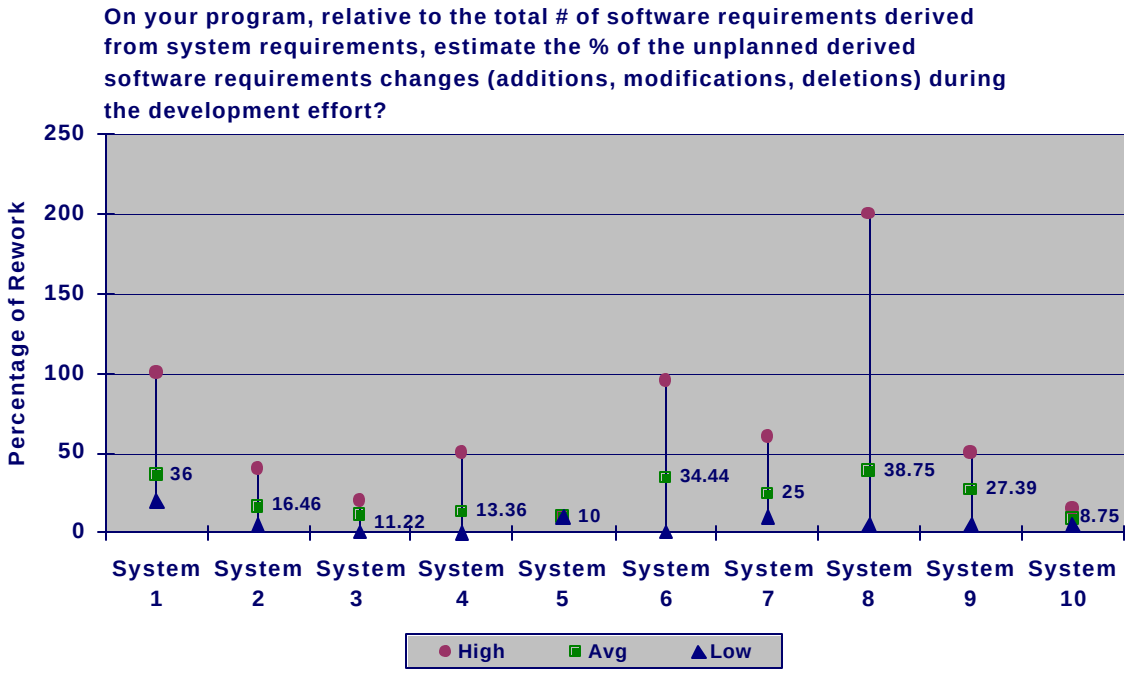

Figure 6: Estimated Percentage of Unplanned Derived Software Requirement Changes

Second, the program and process leadership were asked to estimate the percentage of the unplanned derived software requirement changes (additions, modifications, and deletions) with respect to the total number of software requirements. Unplanned requirements changes are important to track since many of the software development models and software development plans do account for planned rework. Unplanned rework was estimated to be approximately 23% of the total software requirements. A breakdown of the responses for all systems is shown in Figure 7.

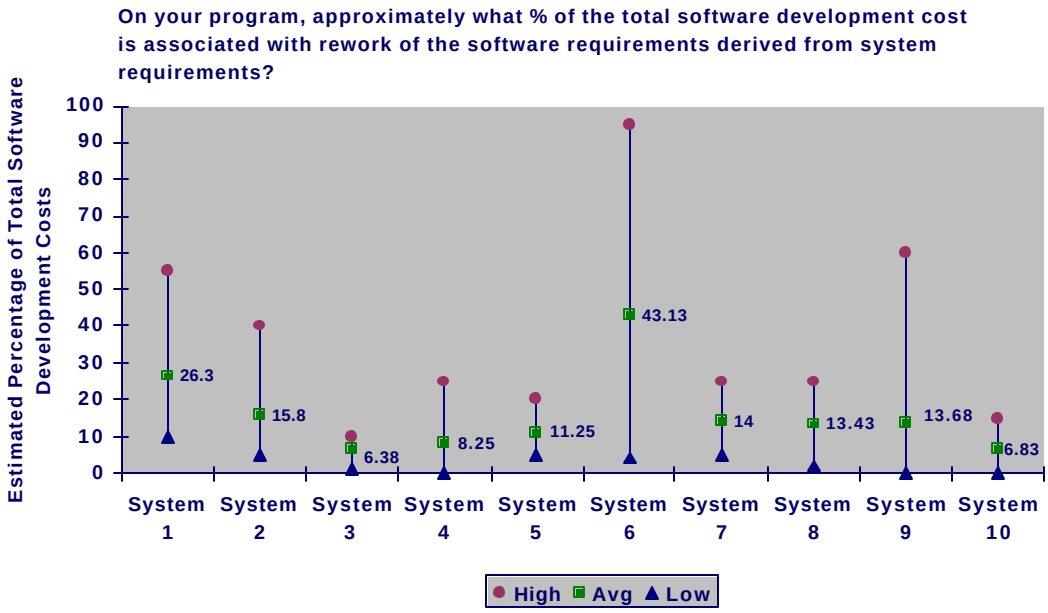

Figure 7: Estimated Percentage of Unplanned Derived Software Requirement Changes

Despite only analyzing successful programs, unplanned rework of derived software requirements is a significant percentage of the total software development costs.

In their entirety, Enterprise Level Metrics for the end-to-end software development process are deficient and could be significantly improved.

Finding #3: Although all phases of the software development process are deemed to add value, they are not accomplished with the same level of effectiveness.

To understand the importance of the requirement phase, the survey was used to establish the relative value of all phases of the software development process. The program and process leadership were asked to identify the value that each phase in the software development process contributes to developing software in a timely, cost effective approach to meet the end users needs. The objective was to understand the relationship between the early stages of the development effort when requirements are derived, and the later stages of the development effort. Survey responses indicated all phases of the development process add value. The results shown in Figure 8 illustrate the estimated value of all phases is about 6 on a 7 point scale. There was no significant variation by application domain.

The survey also asked the leadership to estimate their level of effectiveness in accomplishing these phases. Survey responses showed a variation by application domain. Figure 8 compares the estimated level of value of each phase versus the estimated level of effectiveness in accomplishing these phases.

Figure 8 suggests the phases of the process prior to Design, Code, and Unit Test are not accomplished as well as the later development phases (System Integration, Validation, and Verification). Design, Code, and Unit Test has the most uniform responses across the application domains. A greater level of effectiveness in the latter stages of the development effort and the higher level of effectiveness estimated by the space ground terminal and the commercial aircraft systems suggest an unequal distribution of effectiveness across systems and process phases. The difference between the perceived importance and the level of effectiveness in accomplishing the phases of the development process is solid indicators of potential areas of process improvement.

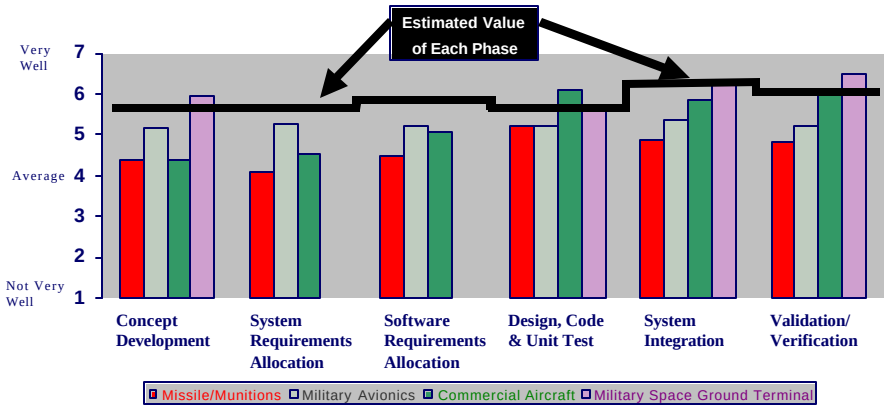

Figure 8: Application Domain Performance versus the Level of Value Added

Finding #4: Information provided by the end user is considered very important by the requirement practitioners and needs to be captured through end user involvement early in the development process.

During the case study interviews, requirement process practitioners repeatedly cited the need to involve the end user in the requirement derivation process. Unfortunately, end user involvement is not enough to guarantee success. Information must be identified and *effectively* captured from the end user during the requirement derivation processes.

Program and process leadership were asked to identify the level of importance of various aspects of end user involvement. The responses for all ten systems are shown in Figure 9. As a whole, the most important aspects of end user involvement were: the agreement between the end user and developers on performance expectations; giving developers an understanding of the operational environment the system; and providing an understanding of the how the system will be used. Clarifying validation and verification criteria also considered very important by seven of the ten systems.

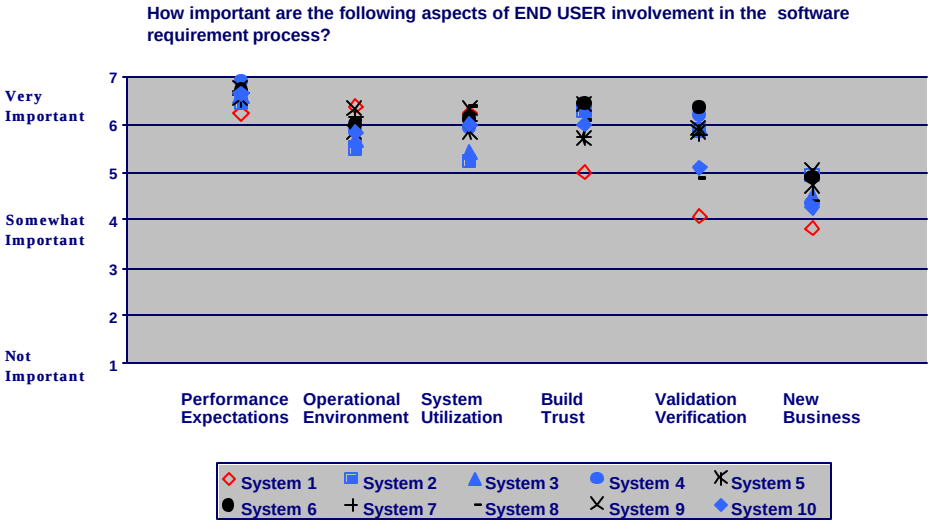

Figure 9: Average Level of Importance of Various Aspects of End User Involvement

Data collected on the three case studies, shown in Figure 10, indicates the end user involvement is greatest during the early stages of the process. Case study interviews also found process participants recognized the need to capture the critical information early in the development process. Practitioners indicated the effective and efficient capture of the critical information early in the process can be a challenge for the software developer but is critical to the successful derivation of software requirements.

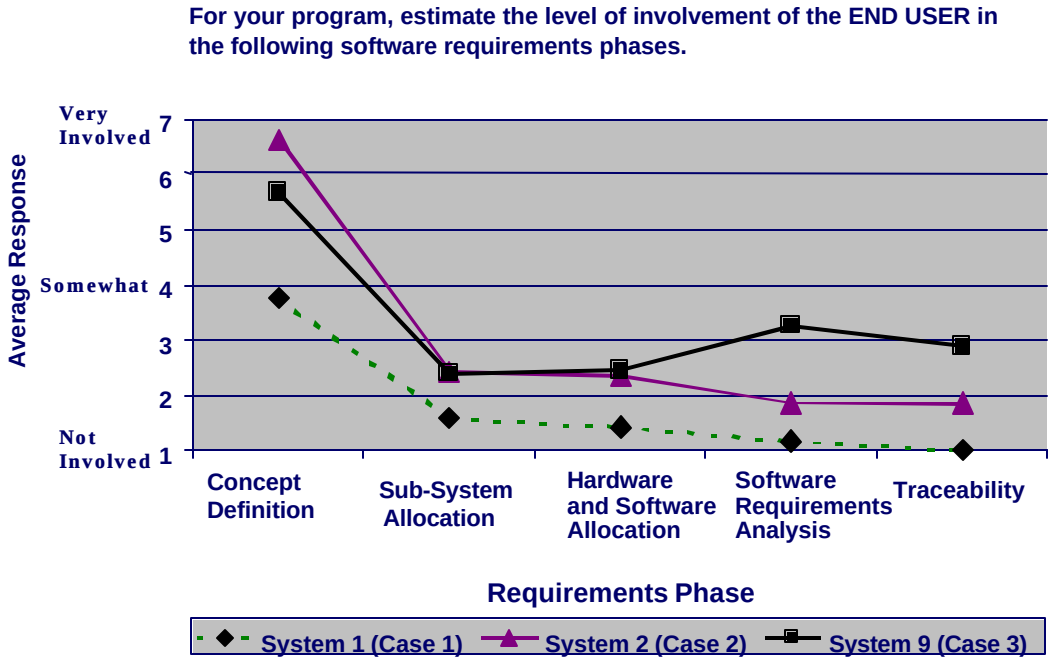

Figure 10: Average Estimated Level of End User Involvement in the Requirement Derivation Process[2]

The combination of the early involvement and the ability to extract critical information are key ingredients to an effective requirement derivation process.

Finding #5: The responsibilities for assuring that software requirements "meet the end user needs" and are "cost effective" are divided among different process owners.

In general, a software requirement should be cost effective, meet the end users needs, be complete, and be traceable [6]. Satisfaction of all four criteria contributes to the successful development of a good software requirement. Of particular importance are the aspects of "cost effectiveness" and "meet the end users needs". Case study interviews indicated a heightened awareness of the software developers to make design trades relating to cost and performance. The program and process leadership were asked to identify the discipline or individual with primary responsibility for making sure the requirements were cost effective and meet the end user needs. The results, shown in Figure 11, indicate the primary responsibility for cost is clearly the role of

[2] The phases in Figure 10 are different that the ones listed in Figure 8 since Figure 10 represents the phases of the requirement phases versus the phases of the software development process shown in Figure 8.

program management while the responsibility for meeting the end user needs is scattered among multiple owners (Customer, End User, System Engineering, Chief Systems Engineer, System Test and Program Management).

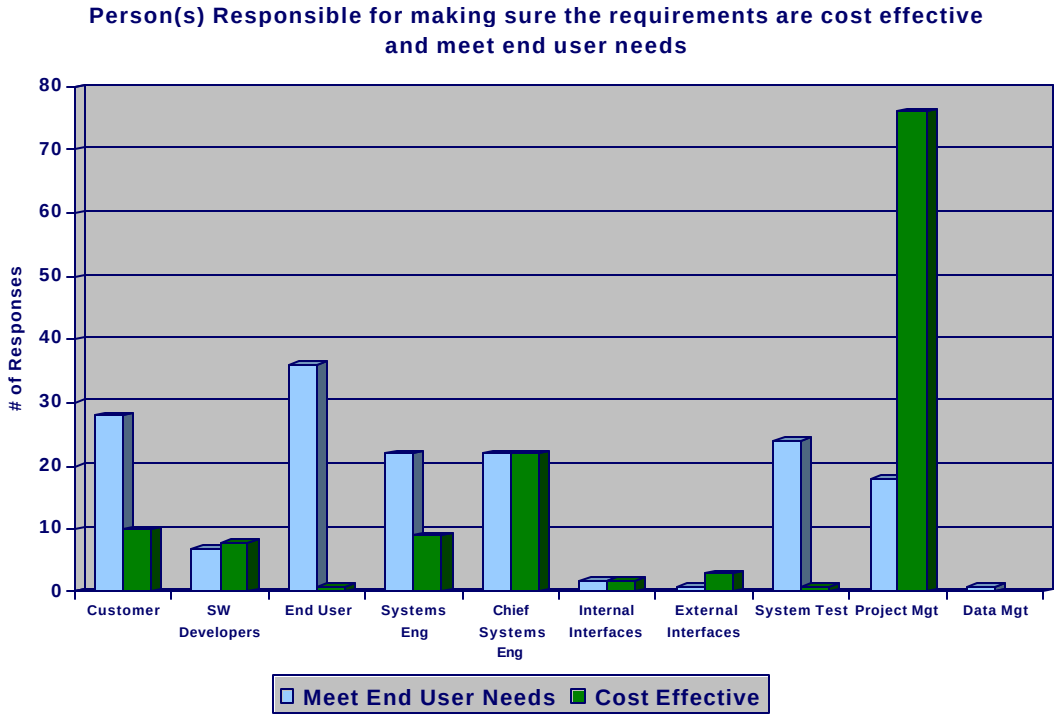

Figure 11: Primary Responsibility for Cost and Performance

The lack of unified responsibility for cost and performance appears to increase the difficulty in accomplishing effective design trades. In a different domain, earlier LAI findings showed significant cost and schedule benefits when designers had simultaneous access to cost and performance databases [6]. While this may have been in a different application domain, the research highlights the benefit of access to the cost impacts during design. Programs with interoperable design and cost databases had less cost and schedule slips. Improvements in the software upgrade programs might be possible if the person and/or discipline with the primary responsibility for cost and performance were unified.

Finding #6: Data suggests positive correlation between reduction in unplanned requirement changes and leadership continuity in both concept definition and requirement analysis phases.

Survey data indicated a correlation between the level of continuity of the contractor process leadership in the concept definition and requirement analysis phases and the amount of unplanned requirement rework. Higher levels of continuity reduced the amount of unplanned rework. Figure 12 illustrates the estimated percentage of unplanned changes for each system (shown in Figure 7) plotted against the percentage of the contractor software developers leadership who claimed to have worked in both the concept definition and the software requirement analysis activities.

The positive correlation between the two suggests the program leadership can reduce the amount of unplanned rework by increasing their continuity in concept definition and requirement analysis. More research is needed to better define the empirical relationship between rework and continuity but the positive correlation offers practitioner an area to focus process improvement initiatives.

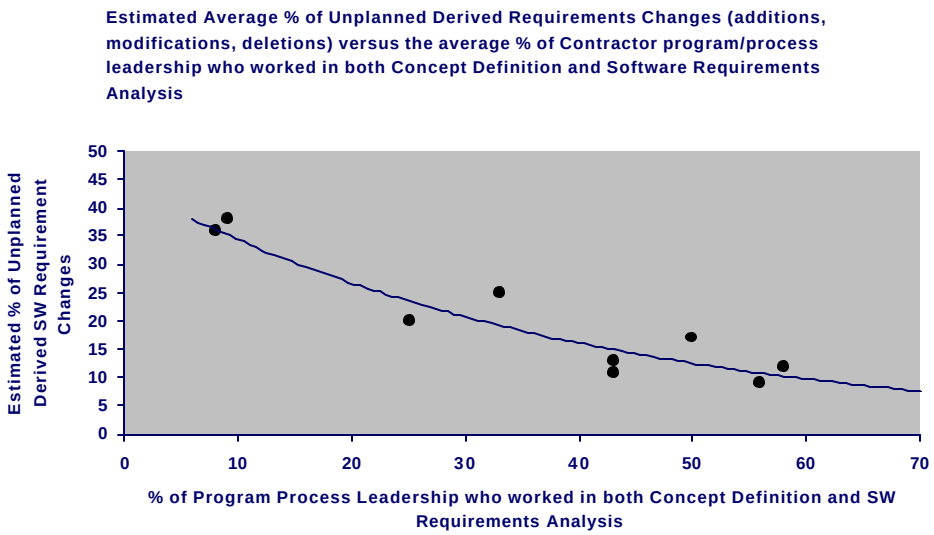

Figure 12: Continuity of Contractor Leadership in Concept Definition and Software Requirement Analysis Activities versus the Percentage of Unplanned Rework

Finding #7: Formal Training for aerospace software requirement practitioners does not appear to be highly effective or abundant.

The program and process leadership were asked to evaluate the effectiveness of multiple training methods in helping them perform their roles and responsibility in the requirement derivation process. Figure 13 illustrates the responses of all survey respondents. Overwhelmingly, survey respondents believed formal training is not effective.

On-the-job training was found to be the most widespread method and the only effective training method. Only approximately 50% of the survey respondents report they had formal university or professional training. Most respondents had in-house training but on average found it to be only "somewhat effective".

Feedback from requirement practitioners suggests formal training would be more effective if it included some of the OJT elements and students had "hands-on" experience deriving requirements. As programs become more complicated the need for better formal training will only increase.

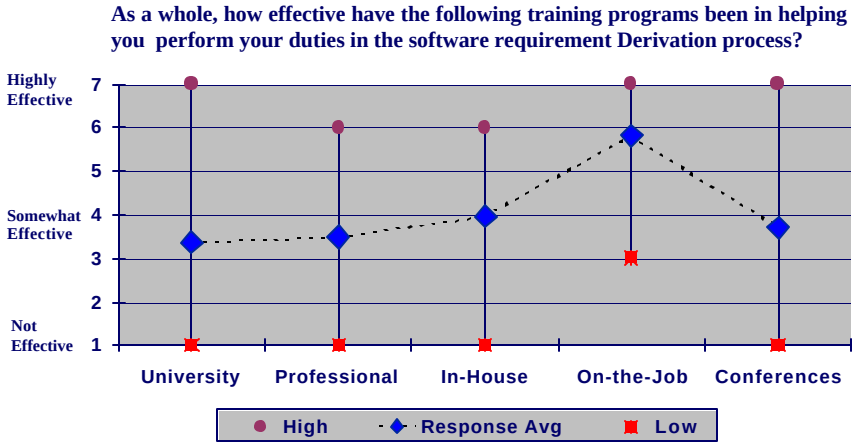

Figure 13: Estimated Level of Formal Training Effectiveness

Summary

The findings, recommendations, and identification of Lean practices were the result of a comprehensive two-year research effort that involved over a hundred and fifty individuals from the MIT faculty, LAI consortium, government, and industry. This research has illustrated that Lean principles and methodology transcend the manufacturing domain and offer real opportunities for improving the product development process.

The findings presented in this papers offer both quantitative and qualitative insight into the software requirement process. An analysis of the data have identified seven major findings that build a basis for developing a framework for increasing value-added contribution of the software requirement process.

Reference:

[1] Ippolito, Brian, Feb 2000, "Identifying Lean Practices for Deriving Software Requirements", MIT SM Thesis, MIT.

[2] Fredriksson, B., Nov 1994, pp 23-31, "Holistic system engineering in product development", The SAAB-SCANIA GRIFFIN.

[3] Davis, Alan, 1990, Software Requirements: Analysis & Specification, Prentice Hall.

[4] http://lean.mit.edu/public/index.html

[5] Bracket, Jan 1990, "Software Requirements; SEI-CM-19-1.2" CMU

[6] Hoult, David P., and C. Lawrence Meador, John Deyst Jr., Maresi Berry-Dennis, Nov 1995, "Cost Awareness in Design: The Role of Data Commonality.", LAI White Paper #LEAN 995-08, MIT.