Carnegie Mellon University
Software Engineering Institute

ROI Analysis of the System Architecture Virtual Integration Initiative

Jörgen Hansson, University of Skövde
Steve Helton, The Boeing Company
Peter Feiler, Software Engineering Institute

April 2018

TECHNICAL REPORT
CMU/SEI-2018-TR-002

Software Solutions Division

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM18-0482

Table of Contents

Acknowledgments **iv**

Executive Summary **v**

Abstract **viii**

1 Introduction **1**

2 ROI Based on Rework Cost-Avoidance **4**

3 Exponential Growth in Avionics Software Systems **7**
- 3.1 Growth Curve for Avionics Software 7
- 3.2 Limits of Affordability 8

4 “As-Is” System-Development Cost Estimates Using COCOMO II **9**
- 4.1 Setup of COCOMO II 9
- 4.2 Cost Computations 12

5 Rework Cost-Avoidance Estimates **15**
- 5.1 Phase-Based Rework Cost Percentages 15
- 5.2 Defect Removal Efficiency 17
- 5.3 Estimate of Cost Savings Due to Rework Avoidance 19

6 ROI Estimates **23**

7 Discussion on the Use of COCOMO II **27**

8 Conclusion **30**

References **32**

List of Figures

Figure 1: SLOC Growth of Avionics Software 8

Figure 2: Total System-Development Cost as a Function of Reuse 14

Figure 3: Rework Cost-Avoidance as a Function of Reuse for Three Project Sizes with 30% and 50% Rework 22

Figure 4: Projected Arithmetic ROI as a Function of Reuse for Three Project Sizes with 30% and 50% Rework 26

Figure 5: Projected Logarithmic ROI as a Function of Reuse for Three Project Sizes with 30% and 50% Rework 26

Figure 6: Computed NPV as a Function of Reuse for Three Project Sizes with 30% and 50% Rework 26

List of Tables

Table 1: SLOC Growth of Avionics Software 7

Table 2: Summary of Switch Settings in COCOMO II for the SAVI Cost Model 11

Table 3: Size of Subsystems with Respect to Criticality and Size of Reused Code Base in MSLOC 12

Table 4: Estimated Software-Development Cost in Millions of US$, Given MSLOC and Amount of Reuse Using the “As-Is” Process 13

Table 5: Estimated Total System-Development Cost, Including Hardware, in Millions of US$ 14

Table 6: Phase-Based Percentages of Introduced and Detected Defects 16

Table 7: Defect-Removal Cost, Given the Phase of Origin 16

Table 8: Nominal Phase-Based Rework Costs and Percentages 17

Table 9: Hayes’s Fault Taxonomy [Hayes 2003, p. 52] 18

Table 10: Expected Removal Efficiency of Faults and Defects When Deploying SAVI 19

Table 11: Avoided Cost as a Function of Rework and Software Reuse 20

Table 12: Computed NPV and PV for 33% Defect-Removal Efficiency 24

Table 13: Computed NPV and PV for 66% Defect-Removal Efficiency 25

Acknowledgments

This work was conducted in 2008-2009 with funding from the Aerospace Vehicle Systems Institute (AVSI) and performed by the authors along with members of AVSI, including David Redman (AVSI), Don Ward (AVSI), John Chilenski (Boeing), Keith Appleby (BAE Systems), Leon Corley (Lockheed Martin), Bruce Lewis (U.S. Army Aviation and Missile Research Development and Engineering Center Software Engineering Directorate, Department of Defense), Jean-Jacques Toumazet (Airbus), John Glenski (Rockwell Collins), Joe Shultz (GE Aviation), Bob Manners (FAA), and Manni Papadopoulos (FAA). The authors would also like to thank the members of the AVSI group for their comments and feedback.

The ROI study report was originally published as an AVSI System Architecture Virtual Integration (SAVI) report. To reach a wider audience, AVSI and the Software Engineering Institute (SEI) made the agreement to republish the study as an SEI technical report.

At the time of the study, Jörgen Hansson was a member of the technical staff at the SEI. Steven Helton retired from Boeing in 2015.

Executive Summary

The size of aerospace software, as measured in source lines of code (SLOC), has grown rapidly. Airbus and Boeing data show that software SLOC have doubled every four years. The current generation of aircraft software exceeds 25 million SLOC (MSLOC). These systems must satisfy safety-critical, embedded, real-time, and security requirements. Consequently, they cost significantly more than general-purpose systems. Their design is more complex due to quality attribute requirements, high connectivity among subsystems, and sensor dependencies—each of which affects all system-development phases but especially design, integration, and verification and validation.

Several analyses of software-development projects show that detecting and removing defects are the most expensive and time-consuming parts of the work. Finding and fixing defects alone often causes projects to overrun budget and schedule because developers must perform significant amounts of rework. The basis of their problem is that most defects are introduced in the pre-coding phases, specifically during requirements and design, but only a fraction are detected and addressed in the same phase. More than half are not found until hardware/software integration occurs. For aerospace and safety-critical systems, the cost of removing a defect introduced in pre-coding phases but detected in post-coding phases is two orders of magnitude greater than the cost of removing it before code development.

The System Architecture Virtual Integration (SAVI) initiative is a multiyear, multimillion dollar program for developing the capability to virtually integrate systems. The capability promises to allow developers to recognize system-level problems early and reduce leakage of errors into the post-coding phases. The program is sponsored by the Aerospace Vehicle Systems Institute (AVSI), a research center of the Texas Engineering Experiment Station, which is a member of the Texas A&M University System. Members of AVSI include Airbus, BAE Systems, Boeing, the Department of Defense, Embraer, the Federal Aviation Administration, General Electric, Goodrich, Aerospace, Hamilton Sundstrand, Honeywell International, Lockheed Martin, NASA, and Rockwell Collins.

This report presents an analysis of the economic effects of the SAVI approach on the development of software-reliant systems for aircraft compared to existing development paradigms. It describes the results of a return-on-investment (ROI) analysis to determine the net present value (NPV) of the investment in the SAVI approach. The investment into the approach over the multi-year SAVI initiative by the different member companies was estimated to be $86M. Their investment covers the maturation, adaptation, and piloting of SAVI practices and technologies, and the transition of the approach into member companies. The analysis uses conservative estimates of costs and benefits to establish a lower bound on the ROI; less conservative figures yield higher economic gains.

This study was performed in 2008-2009 in the context of a proof-of-concept virtual-integration study in the use of the SAE International standard named the Architecture Analysis and Design Language (AADL). The standard is a key element of the virtual system integration approach of SAVI and its effectiveness in early discovery of defects through analysis of AADL models of the embedded software systems.

Summary of Methods

The approach taken in this study was to determine the rework cost-avoidance based on SAVI practice by applying an efficiency rate for removing defects to the rework cost of a system compared to current practice. The approach included the following conservative assumptions:

- We adopted COnstructive COst MOdel (COCOMO) II, the leading tool for estimating software development costs using the current development process. Using typical development processes, we derived the total cost for developing three software systems of different sizes as follows:
  - Each system consists of three types of subsystems—safety critical, highly critical, and less critical with code bases of 30%, 30%, and 40%, respectively, of the total code base—a typical mix in aircraft industry. This let us differentiate the cost of subsystems with respect to their requirements.
  - Each subsystem is developed with both new code and the reuse of existing code. We considered three cases of new code development and varied the proportions of code reuse from 30% to 70%.
  - We used three system sizes: two based on the current generation of aircraft software systems (27 and 30 MSLOC) and one reflecting a future system of 60 MSLOC. The synthetic system clearly illustrates the economic impact of system growth, although building a system of this size is unaffordable.
  - The nominal labor rate is $28,200 per month for 2014, based on 2006 data of $22,800 = $150/hr. * 152 hr./mo. * 1.02694*(2014 – 2006), adjusted for annual inflation at 2.694%.
- On the basis of SAVI members' experiences, we estimated the total system-development cost from an estimate of the software-development cost using a multiplier of 1.55, which reflects software development making up about 66% of system-development cost.
- On the basis of documented and experiential evidence for aerospace systems, we used two conservative estimates for total rework cost: 30% and 50% of the total system-development cost.
- We determined ROI and NPV based on rework cost reduction attributed to earlier discovery of defects and did not include reductions in maintenance cost and deployment delays. We limited rework cost savings to discovery of requirements errors, which make up 35% of all errors and 79% of the rework cost.
- We used experts' estimates of the efficiency rate for removing defects of 66% as well as a reduced rate of 33% for more conservative estimates.
- We assumed that SAVI practices of model creation and analysis would replace existing document-based practices of system requirement and design specification at a similar cost.
- We used $86M as estimated investment by the SAVI member companies over multiple years to mature SAVI and transition current practice to SAVI.

Summary of Findings

Our analysis produced the following outcomes and observations:

- Approximately 70% of defects are requirements and design defects, but less than 10% are detected in these early phases. The rework cost of correcting defects introduced in requirements and design but detected late in the development lifecycle is one to two orders of magnitude higher than the cost of correcting them before coding. Requirements-related rework cost amounts to 79% and design-related rework costs 16% of the total rework cost.
- In the most conservative scenario, for a 27-MSLOC system, the smallest cost avoidance is $717 million (out of an estimated $9.176 billion cost of development, a 7.8% cost savings). This is with 70% reuse, rework cost as 30% of total system-development cost, and a removal efficiency of 33%. The arithmetic and logarithmic (continuously compounded) ROIs are 7.3 and 2.12, with an NPV of $263 million.
- The nominal cost reduction for a 27-MSLOC system is $2.391 billion (out of an estimated $9.176 billion, a 26.1% cost savings), occurring at 70% reuse, rework cost as 50% of total system-development cost, and 66% removal efficiency. The arithmetic and logarithmic ROIs are 26.8 and 3.33, with an NPV of $1.076 billion.
- The cost reduction is linear to the rework cost and removal efficiency. With other factors held constant in a scenario, each unit's increase in removal efficiency for requirements errors resulted in a cost reduction. For example, for the 27-MSLOC system with the highest degree of reuse, each 1% increase in removal efficiency resulted in a cost reduction of $22 million.
- Cost reduction for a given system size is also linear to the amount of reuse. However, comparing the cost reduction for systems of different size, we observed that the cost reduction for 60 MSLOC was more than twice that for 30 MSLOC. This is to be expected given the more-than-linear increase in interaction complexity in a larger system.

The predicted returns were considered to be higher than anticipated, which led to several follow-on activities. First, one of the SAVI system integrator members obtained an independent assessment by its organization's cost estimating group that agreed with the findings of this report. Second, this initial ROI study was followed by a second SAVI ROI study. In the second study, a Monte Carlo algorithm was used to drive the COCOMO II cost estimation, resulting in a reduced variation of results. In addition, the commercial tool SEER was used to build a SEER-SEM and SEER-H model of a Boeing 777-200 to explicitly estimate the cost of the non-software portion of the system and compare it to both publicly available data and estimates of the original SAVI ROI study presented here. The SEER analysis confirmed that the cost multiplier of 1.55 was acceptable for 2010. Unfortunately, the software count increases while the physical parts count remains stable, resulting in a software increase from 66% in 2010 to 88% of the total system-development cost by 2024.

Abstract

The System Architecture Virtual Integration (SAVI) initiative is a multiyear, multimillion dollar program that is developing the capability to virtually integrate systems before designs are implemented and tested on hardware. The purpose of SAVI is to develop a means of countering the costs of exponentially increasing complexity in modern aerospace software systems. The program is sponsored by the Aerospace Vehicle Systems Institute, a research center of the Texas Engineering Experiment Station, which is a member of the Texas A&M University System. This report presents an analysis of the economic effects of the SAVI approach on the development of software-reliant systems for aircraft compared to existing development paradigms. The report describes the detailed inputs and results of a return-on-investment (ROI) analysis to determine the net present value of the investment in the SAVI approach. The ROI is based on rework cost-avoidance attributed to earlier discovery of requirements errors through analysis of virtually integrated models of the embedded software system expressed in the SAE International Architecture Analysis and Design Language (AADL) standard architecture modeling language. The ROI analysis uses conservative estimates of costs and benefits, especially for those parameters that have a proven, strong correlation to overall system-development cost. The results of the analysis, in part, show that the nominal cost reduction for a system that contains 27 million source lines of code would be $2.391 billion (out of an estimated $9.176 billion), a 26.1% cost savings. The original study, reported here, had a follow-on study to validate and further refine the estimated cost savings.

1 Introduction

Analysis of software-development projects shows that detecting, locating, and removing defects are the most expensive and time-consuming parts of the work. Finding and fixing defects often cause projects to run over budget and schedule as developers perform significant amounts of rework in later phases of product development [RTI 2002, Dabney 2003]. For information technology (IT) applications, the defect-removal efficiency before delivery is generally about 80% to 85%; the cost of correcting these defects averages about 35% of the total system-development cost [Jones 2007]. Correspondingly, the time needed to rework defects averages approximately 35% of the total project-development schedule.

Experts have observed that the rework fraction of total development work increases with the size of the project and can be as high as 60% to 80% for very large projects [Basili 1994, 2001; Jones 1996; Cross 2002]. Aerospace software systems, in particular, have grown at a rapid pace. Airbus and Boeing data presented in this report show that growth in millions of source lines of code (MSLOC) for aircraft software will double every four years, and current-generation software exceeds 25 MSLOC. Safety-critical system design is intrinsically more complex than general-purpose system design because of the quality attribute requirements, high connectivity among subsystems, and sensor dependencies that affect all system-development phases but are most critical to design, integration, and verification and validation activities.

The main problem is clear: most defects are introduced in the early pre-coding phases of development, such as requirements and design, but the majority of defects are detected and removed in post-coding phases, such as integration and testing. The nominal cost of removing a defect introduced in pre-coding phases and detected in post-coding phases is generally one order of magnitude higher than the cost of removing it prior to code development. For safety-critical systems, the difference can be as much as two orders of magnitude higher. In this report, we present such data from multiple sources.

Software systems are growing in size, not only in the number of subsystems but also the degree of interactions between them. This condition likely will further raise the defect-removal cost, as each defect affects a larger number of subsystems. This condition will require innovative solutions in the following areas:

- Understanding the dynamics of defect introduction and removal—the phases in which defects are introduced, the phases in which they are detected, and the cost of removing defects relative to the phase lag between introduction and detection—is paramount to accurately estimating the rework cost in terms of total software cost.
- The dominance of rework cost resulting from requirements and architectural design defects clearly suggests a strong need for improved techniques to prevent and detect such defects.

The System Architecture Virtual Integration (SAVI) initiative is a multiyear, multimillion dollar program focused on developing the capability to virtually integrate systems before designs are committed to hardware, as a means of managing the exponentially increasing complexity of modern aerospace systems. Its objective is to discover system-level errors—typically requirements and design errors—through virtual integration that occurs earlier in the development lifecycle.

The program is led by the Aerospace Vehicle Systems Institute (AVSI), a research center of the Texas Engineering Experiment Station, which is a member of the Texas A&M University System. Membership in AVSI includes Airbus, BAE Systems, Boeing, the Department of Defense, the Federal Aviation Administration, General Electric, Goodrich Aerospace, Hamilton Sundstrand, Honeywell International, Lockheed Martin, NASA, and Rockwell Collins.

Using a quantitative architecture-modeling approach, SAVI aims to significantly increase early detection and removal of defects as well as to prevent defects from entering into the design in the first place. One project that SAVI members have undertaken is a proof-of-concept demonstration of SAVI technology to discover defects early in development through analysis of multiple functional and non-functional properties on a realistic multi-tier model of an aircraft avionics system [Feiler 2009, Feiler 2010, Redman 2010]. The technology consists of analytical technologies for virtual integration of embedded software systems such as the SAE International Architecture Analysis and Design Language (AADL) standard and tool suite. Another SAVI project is a return-on-investment (ROI) analysis, which is the subject of this report. This report presents our analysis of the economic effects on the development of software-reliant systems for aircraft when deploying SAVI relative to existing development paradigms. A challenge for this ROI study was that it had to be based on publicly available data rather than on competition-sensitive data of the members.

To evaluate the economic effects of the SAVI engineering practice, we compared the relative cost advantage of two development paradigms, one following the development practices in place today ("as is") and another deploying SAVI technology ("to be"). If all else remains the same, the difference between the two approaches is the efficiency in managing complexity and correcting defects. From our ROI (or rate-of-return) analysis, we computed the net present value (NPV, also called net present worth).

ROI is a measure of the monetary value generated by an investment or of the monetary loss caused by an investment. It measures the cash flow or income stream from the investment to the investor and denotes the ratio of money gained or lost (realized or unrealized) on an investment relative to the amount of money invested. In our case, SAVI members expected to invest $86M over multiple years into the maturation and transition of SAVI technology into practice, and the ROI is an indication of rate of return due to cost reduction.

NPV is commonly used for appraising long-term projects by deriving the time value of money and considering cash flows over time. Outgoing cash flows include start-up costs, initial investments, and operational costs; incoming cash flow implies positive cash flow from the investment, which, in the case of SAVI, is based largely on cost avoidance. Computing NPV indicates how much value an investment or project adds to the organization. A positive NPV implies that the investment would add value to the organization; a negative NPV implies that the investment would subtract value. In our case, NPV represents cost savings minus expenses in U.S. dollars for a project running from 2010-2018 using SAVI technology.

In Section 2, we present the ROI analysis in terms of a rework cost-avoidance formula. The sections that follow that analysis elaborate on the contributing elements of the ROI formula:

- In Section 3, we discuss the exponential growth of avionics software systems in terms of SLOC by analyzing the historical data to correlate major cost drivers to system size.

- In Section 4, we elaborate on how we obtained software and system-development cost estimates under the “as-is” process using COnstructive COst MOdel (COCOMO) II with the SLOC estimates as input.
- In Section 5, we present our method for estimating rework cost-avoidance due to early discovery of requirements-related defects as a percentage of total system-development cost.
- In Section 6, we present the ROI and NPV estimates based on the rework cost-avoidance formula.
- In Section 7, we discuss the limitations of using COCOMO II.
- Finally, in Section 8, we interpret the results of the ROI analysis and reflect on the assumptions we used as well as potential improvements.

2 ROI Based on Rework Cost-Avoidance

We based our analysis to determine ROI and NPV for a typical SAVI deployment on the investment (an estimated $86M) by SAVI member companies in the multi-year SAVI initiative to mature and transition the new technology and development paradigm to member company product groups. We also considered the cost of implementing SAVI in a project to be the same as the cost for current methods, once a team has been trained in the SAVI practice—a cost covered in the investment figure.[1]

The ROI analysis is based on avoiding rework cost by detecting defects that are currently detected post-unit test with a high-rework cost earlier in the development process through use of SAVI. The SAVI approach aims to reduce requirements and design defects through up-front modeling and validation, preventing these defects from flowing down to later phases where they would cause significant rework efforts and thus cost more to fix.

Our estimates of the possible savings from rework avoidance include several observations about factors that increase costs:

- Rework cost is primarily driven by failures in integration [Lutz 1993].[2]
- More than 70% of all defects can be traced back to defects introduced in pre-code development phases (requirements and design) with nominal rework cost of two orders of magnitude greater than the cost of removing defects before coding [Dabney 2003].
- Rework constitutes a significant portion of the total system-development cost with requirements-related rework making up 79% of the total rework cost [Dabney 2003]. Thus, increasing defect detection and removal efficiency lowers the rework cost.

We then computed the rework cost avoided as follows:

*rework cost-avoidance = estimated total "as-is" system-development cost * % rework cost * % requirements-error rework cost * % requirements-error removal efficiency*

---

1 Implementation costs in a "to-be" SAVI project consist of creating, evolving, and analyzing practice models of the system. Those activities replace current document-based methods for specifying system requirements, requests for bids, and design documentation. In the context of this study, we assume the cost for applying SAVI to be the same as the cost for current methods.

2 In this study, 387 software defects discovered during the integration and test phase of the Voyager and Galileo spacecraft were analyzed (the same software was used to control both spacecraft). Lutz found that 98% of the faults were attributed to "functional faults" (operating faults and conditional faults resulting from incorrect condition or limit values), behavioral faults (i.e., a system is not conforming to requirements), and "interface faults" (related to interaction with other systems' components). Only 2% of the faults were coding faults internal to a module. The functional and interface faults were direct consequences either of errors in understanding and implementing requirements or of inadequate communication among development teams. Safety-related errors accounted for 48% of the errors discovered in Voyager and 56% discovered in Galileo; 36% of the errors in Voyager and 19% of the errors in Galileo were related to interface faults. Inter-team communication errors (as opposed to intra-team) were the leading cause of interface faults (93% for Voyager and 73% for Galileo). One primary cause of safety-related interface faults was misunderstood hardware-interface specifications (67% for Voyager and 48% for Galileo). Errors in recognizing and understanding the requirements were a significant cause of functional faults (62% for Voyager and 79% for Galileo).

We estimated the total system-development cost using the "as-is" process as follows:

- We estimated the size in SLOC and complexity of software-reliant avionics systems by extrapolating data from previously built aircraft. Developers in AVSI checked these estimates for validity.
- We used COCOMO II [Boehm 2000, COCOMO II], which is an industry-accepted cost-estimating tool, to estimate software-development costs based on the software system size in SLOC and other parameters that reflect system and project complexity.
- We considered reuse percentages ranging from 30% to 70% in the COCOMO II-based estimation.
- We then derived system-development costs in proportion to the software-development cost by using a multiplication factor of 1.55. This figure was acceptable to the SAVI member companies.

The rework cost percentage represents the total rework cost as a percentage of the total system-development cost. The rework cost percentage for requirements errors represents the percentage of rework cost attributable to requirements errors in terms of total rework cost. The removal-efficiency percentage for requirements errors represents the percentage of requirements errors that are discovered and removed during the requirements phase due to SAVI early detection instead of in a later phase.

- We investigated the effects of changing the percentage of rework cost. Assuming all else remains the same, we considered rework to be a conservative 30% of total cost and a nominal 50% of total cost for 2010, in agreement with SAVI members.
- We computed the rework cost for requirements errors as a percentage of the total rework cost. We used an estimated number of defects that would typically be introduced and discovered in different development phases for a system of a defined size and complexity. Empirical data derived from case studies [RTI 2002, Dabney 2003] and recent experiences of the companies participating in the SAVI initiative corroborated our estimates.
- We determined the effectiveness of defect-removal efficiency in SAVI-based development compared to current system-development practices, based on the ratios of defects being introduced in the different phases of the system-development lifecycle and the likelihood of detecting a defect in a certain phase. On the basis of Miller's fault taxonomy [Miller 1995] and fault distributions derived from current system-development practices (i.e., as is) [Hayes 2003], we applied conservative assumptions about the effectiveness of SAVI deployment in reducing certain classes of faults. Specifically, for each fault class, experts from SAVI-member companies assigned a 0, 50, or 100% probability value for the impact of the SAVI approach on fault-removal efficiency and improved early fault detection.[3] This evaluation resulted in a defect-removal efficiency of 66% for requirement defects. We also added a skeptical scenario in which we reduced the defect-removal efficiency by a factor of 0.5 to 33%.

---

3 These estimates were made based on the proof-of-concept demonstration experience and member company experiences with analytical model-based technologies.

We used the resulting "to-be" rework cost-avoidance figures to compute arithmetic and logarithmic ROI, present value (PV), and NPV values following Dabney's approach [Dabney 2003].

Fairley and Willshire classified rework into three categories [Fairley 2005]:

1. Evolutionary rework, which is caused by external factors, including changing requirements, design constraints, and environmental factors. This rework is unavoidable given the unforeseeable nature of external factors.
2. Retrospective rework, which is conducted to improve structure, functionality, behavior, or quality attributes of previous versions to accommodate the needs of the current version.
3. Corrective rework, which is aimed at fixing defects discovered in current and previous versions.

In our rework cost-avoidance and ROI calculations, we only take into account corrective rework before initial delivery. SAVI practices will have cost-savings effects beyond directly reducing the rework cost. An example of these effects is avoiding the programmatic costs of delays in system delivery and reduction in continuing sustainment costs. Furthermore, retrospective and evolutionary work may experience cost reduction due to SAVI. The ROI, PV, and NPV calculations do not reflect such additional cost savings.

3 Exponential Growth in Avionics Software Systems

Software was first used in commercial aircraft in 1968 when Litton LTN-52 Inertial Navigation Systems entered service on the Boeing 707 [Potocki de Montalk 1991]. Software has since grown to become more important for various services in an aircraft; correspondingly, software has increased in size and complexity.

3.1 Growth Curve for Avionics Software

Table 1 and the corresponding plot in Figure 1 illustrate the growth of software content between 1974 and 1993, based on Airbus data [Potocki de Montalk 1991]. The Words and SLOC columns denote the size of the software in 32-bit words and in SLOC, respectively. The conversion from Words to SLOC uses a factor of 0.2, which is based on the assumption that one word equals four bytes and that one SLOC is approximately 20 bytes (i.e., 4/20 = 0.2) [Hatton 2005]. Since the historic growth appears to be exponential, we derived the log of SLOC, or ln(SLOC), and extrapolated the data for more recent years, as shown in Figure 1. The projected values for SLOC growth over the interval 2000–2010 align well with internal and proprietary data that member organizations of AVSI have gathered from recent development of avionics software.

Table 1: SLOC Growth of Avionics Software

Model	Year	Words	SLOC	ln(SLOC)	SLOC LineFit	ln(LineFit)
A300B	1974	23,000	4,600	8.434	46,302	10.74
A300FF	1981	200,000	40,000	10.597	186,911	12.14
A310	1985	2,000,000	400,000	12.899	414,901	12.94
A320	1988	4,000,000	800,000	13.592	754,528	13.53
A330/340	1993	10,000,000	2,000,000	14.509	2,044,375	14.53
	2000				8,252,771	15.93
	2006				27,293,677	17.12
	2010				60,585,711	17.92
	2014				134,486,402	18.72
	2018				298,529,010	19.51

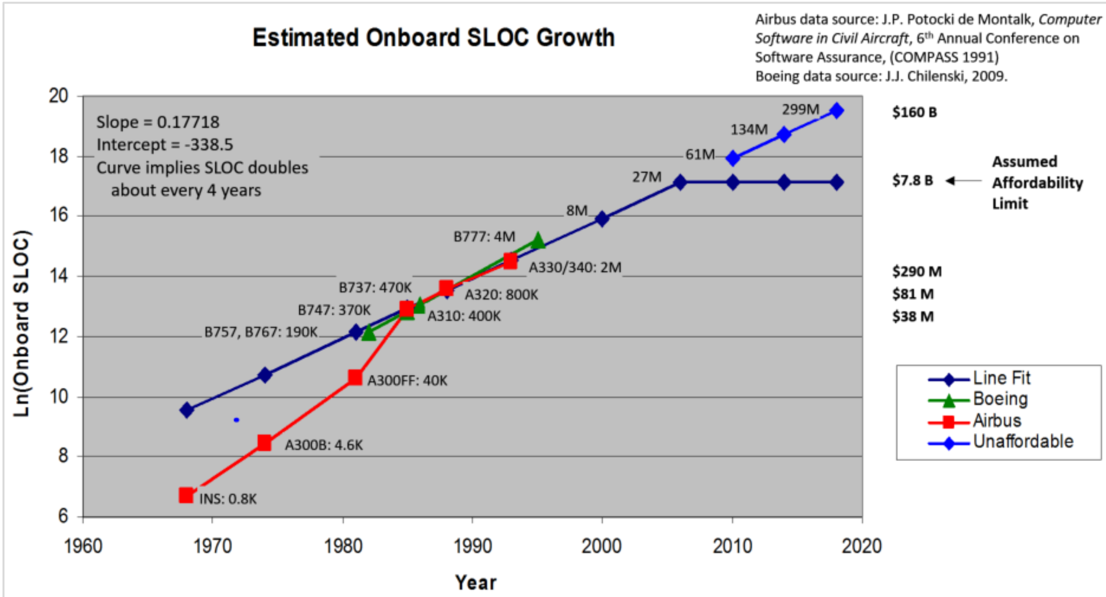

Figure 1: SLOC Growth of Avionics Software

To calculate the SLOC LineFit points that are plotted in Figure 1, we computed the interception point at which a line intersects the *y*-axis by using existing *x*-values and *y*-values. The interception point is based on a best-fit regression line plotted through the known *x*-values and known *y*-values. It uses the slope of the linear regression line through data points for known values of SLOCs using ln(SLOC) (denoted *y* in formula) and known years (denoted *x* in formula). The slope is the vertical distance divided by the horizontal distance between any two points on the line, which is the rate of change along the regression line.

Formally, we have $a = \bar{y} - b\bar{x}$, $b = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sum(x-\bar{x})^2}$, where *x* and *y* represent the averages of known data. The ln(LineFit) provides the exponential growth, which we used to project the number of SLOCs, based on trends in existing aerospace software. The SLOC LineFit is computed as $SLOC\ LineFit = e^{(+(b*Year)+a)}$. When we compared the values of SLOC and SLOC LineFit, we saw that small code sizes grew faster, but the projections held well for 1985–1993. Therefore, when computing the slope (*b*), we used the interval 1985–1993, for which we had public data.

3.2 Limits of Affordability

Unconstrained growth will rapidly lead to unsustainable costs. Thus, a limiting factor related to affordability will produce an inflection point, or S-shape, in the growth curve. SAVI members consider the cost of $10 billion as such an inflection point, shown in Figure 1 as equivalent to 27 MSLOC. The purpose of the proof-of-concept activity for the SAVI virtual-integration approach is to show that a change in practice will result in considerable cost reductions, thus allowing much larger software systems to be developed and certified at sustainable cost.

4 "As-Is" System-Development Cost Estimates Using COCOMO II

We estimated the system-development cost under the "as-is" process by first using COCOMO II to estimate the software-development cost and then extrapolating the total system-development cost based on a realistic multiplication factor that the SAVI members agreed with. This factor of 1.55—corresponding to software representing 66% of total system-development cost—was considered nominal for the 2010 time frame. This figure was further confirmed in a follow-on SAVI ROI study by explicitly estimating the system cost through the use of SEER [Ward 2011, SAVI 2015a, SAVI 2015b].

COCOMO II has been designed with the following capabilities [Boehm 2000, p. 3]:

1. Provide accurate cost and schedule estimates for both current and likely future software projects.
2. Enable organizations to easily recalibrate, tailor, or extend COCOMO II to better fit their unique situations.
3. Provide easy-to-understand definitions of the model's inputs, outputs, and assumptions.
4. Provide a constructive, normative, and evolving model.

COCOMO II represents the evolution of COCOMO 81 [Boehm 1981], which it replaces. The COCOMO family of tools has enjoyed wide adoption in the software industry and has been successfully tailored to specific domains within that industry.

COCOMO II uses a measure of SLOC that represents the size of the system to be developed. The measure of SLOC in COCOMO II follows the guidelines developed by the Software Metrics Definition Group [Park 1992].

4.1 Setup of COCOMO II

We set the following parameters in COCOMO II:

- The size of the code base is in MSLOC of C code (three cases): 27, 30, and 60.
- The code base is divided into three subsystems based on criticality: 30% safety critical, 30% highly critical, and 40% less critical—a typical mix in aircraft industry.
- The percentage of software reuse is varied from 30% to 70%.
- The nominal labor rate is $28,200 per month for 2014 (based on 2006 data of $22.80 = $150/hr. * 152 hr./mo. * 1.02694*(2014 – 2006) and has been adjusted for inflation at 2.694% annually.[4]
- We used the post-architecture model because it is the most detailed model offered in COCOMO II.

---

4 Computed Consumer Price Index average from 1991 to 2008 [U.S. Department of Labor 2011]

COCOMO II features parameters to characterize the software-development project to ensure that an estimate is accurate and true to the preconditions of the project. Each of the parameters reflects a particular attribute of the project and is rated on a six-point scale. Each rating is assigned an effort multiplier. The product of all effort multipliers results in an *effort adjustment factor* (EAF). The EAF is applied to the system size to estimate project effort taking into account the project-specific attribute ratings. We tailored the COCOMO II model to an aircraft-industry-specific project scenario by assigning appropriate ratings for these project attributes.

Following COCOMO II guidance, we set attribute ratings conservatively toward the nominal level. Thus, if a level fell between high and very high, we set it at high. The following attributes were assigned ratings with the effort factor shown in parentheses, and the rating assignments are summarized in Table 2. The attributes are as follows:

- Aerospace development is conducted by internationally distributed teams that engage in multisite development (SITE).
- Aerospace software is safety critical; as a result, it requires a high degree of reliability (RELY).
- Aerospace software is embedded, operates under stringent processor and memory-resource constraints, and requires efficient utilization of processing hardware (TIME) and memory (STOR).
- Aerospace software requires more documentation than conventional software (DOCU).
- Aerospace software is more complex than conventional software (CPLX).

The multisite-development (SITE) attribute indicates the degree of site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia). We chose the level

- very low (1.22): international

The required-software-reliability (RELY) attribute denotes the extent to which the software must perform its intended function over a period of time. We chose three criticality levels to reflect less critical, highly critical, and safety-critical subsystems:

- nominal (1.00): moderate, easily recoverable losses
- high (1.10): high financial loss
- very high (1.26): risk to human life

The execution-time-constraint (TIME) attribute indicates the expected use of processing capacity. We chose the level

- high (1.11): 70% use of available execution time

The main-storage-constraint (STOR) attribute represents the degree of main storage constraint imposed on a software system or subsystem. We chose the degree

- high (1.05): 70% use of available storage

Regarding the attribute that matches documentation needs to lifecycle needs (DOCU), developing for reusability imposes constraints on a project's RELY and DOCU ratings. The RELY rating should be at most one level below the required-reusability (RUSE) cost-driver rating.[5] The DOCU rating should be at least nominal for nominal and high RUSE ratings and at least high for very high and extra-high RUSE ratings. We chose the level

- high (1.11): excessive for lifecycle needs

The product-complexity (CPLX) rating is the subjective weighted average of the complexity rating with respect to control operations, computational operations, device-dependent operations, data-management operations, and user-interface management operations. Aerospace software ranks extra high for control operations and device-dependent operations due to the hard-real-time-control way resources and devices are managed. For computational operations, database operations, and user interfaces (less critical and highly critical subsystems), we considered the complexity to be at least high. Few software systems exhibit the complexity of aerospace software; therefore, we chose the ranking of very high for safety-critical subsystems. We chose levels

- high (1.17)
- very high (1.34)

Because large aerospace software consists of many subsystems with different requirements and criticalities, we decomposed the system into a number of modules in COCOMO II and set different switch levels for each subsystem, resulting in EAF changes.

Table 2 summarizes the switch settings in COCOMO II that focus on system-specific aspects.

Table 2: Summary of Switch Settings in COCOMO II for the SAVI Cost Model

	Safety Critical	**Highly Critical**	**Less Critical**
Fraction of Code Base	30%	30%	40%
CPLX	Very high	High	High
RELY	Very high	High	Nominal
STOR	High		
TIME	High		
DOCU	High		
SITE	Very low		
EAF	2.66	2.03	1.85

COCOMO II has an additional set of parameters that characterizes qualities more specific to organizations, as they are related to staff expertise, organization maturity, development environment, and the like. These parameters, which we set to the nominal value, are

- database size (DATA)
- platform volatility (PVOL), referring to hardware and operating systems
- use of software tools (TOOL)

---

[5] RUSE captures the effort needed to make software components intended for reuse.

We also used personnel attributes to capture the level of skills that personnel possess. These include

- analyst capabilities (ACAP)
- applications experience (APEX)
- programmer capabilities (PCAP)
- platform experience (PLEX)
- programming-language experience (LTEX)
- personnel continuity (PCON)

4.2 Cost Computations

We considered 30%, 40%, 50%, 60%, and 70% of reuse for the three code bases that contain 27, 30, and 60 MSLOC. The reuse percentage was applied to all subsystems, as shown in Table 3. The Code Base column shows the total SLOC divided into the three criticality categories, with safety-critical software accounting for 30%, highly critical software for 30%, and less-critical software for 40%—a typical mix in the aircraft industry. The next set of columns shows the amount of reused code for each criticality category based on a reuse percentage ranging from 30% to 70%.

Table 3: Size of Subsystems with Respect to Criticality and Size of Reused Code Base in MSLOC

MSLOC	Criticality	Code Base	Reuse				
			30%	40%	50%	60%	70%
27	Safety critical (30%)	8.1	2.43	3.24	4.05	4.86	5.67
	Highly critical (30%)	8.1	2.43	3.24	4.05	4.86	5.67
	Less critical (40%)	10.8	3.24	4.32	5.4	6.48	7.56
30	Safety critical	9	2.7	3.6	4.5	5.4	6.3
	Highly critical	9	2.7	3.6	4.5	5.4	6.3
	Less critical	12	3.6	4.8	6	7.2	8.4
60	Safety critical	18	5.4	7.2	9	10.8	12.6
	Highly critical	18	5.4	7.2	9	10.8	12.6
	Less critical	34	7.2	9.6	12	14.4	16.8

COCOMO II has additional switches that affect how the cost estimation reflects the amount of code reuse. We considered software understanding (SU) to be higher than a nominal value (equal to 20), which implies that the system has good structure (high cohesion, low coupling), good application clarity (good correlation between program and application code), and a high degree of self-descriptiveness (good code commentary, good and useful documentation overall). The degree of unfamiliarity of the software (UNFM) is 0.2, indicating that the software is mostly familiar and is lower than the nominal value of 0.4. We set the remaining parameters to nominal values. As a result, the computed adjustment factor is 20.

Table 4 presents the estimated costs of the software development produced by COCOMO II. All costs are in millions of US$.[6] COCOMO II produces nominal cost values as well as lower bound (LB of -20%) and upper bound (UB of +25%) values (designated in the table with a superscript *a*).

Table 4: Estimated Software-Development Cost in Millions of US$, Given MSLOC and Amount of Reuse Using the "As-Is" Process

MSLOC	Reuse	Nominal[a]	Lower Bound[a]	Upper Bound[a]
27	70%	$5,920	$4,736	$7,401
	60%	$6,938	$5,551	$8,674
	50%	$7,971	$6,377	$9,964
	40%	$9,044	$7,235	$11,304
	30%	$10,070	$8,057	$12,589
30	70%	$6,648	$5,319	$8,310
	60%	$7,792	$6,233	$9,739
	50%	$8,951	$7,160	$11,188
	40%	$10,123	$8,098	$12,654
	30%	$11,307	$9,047	$14,135
60	70%	$14,247	$11,397	$17,809
	60%	$16,698	$13,358	$20,872
	50%	$19,182	$15,345	$23,977
	40%	$21,694	$17,356	$27,118
	30%	$24,235	$19,388	$30,294

On the basis of historical data from previous projects in the industry, we applied 1.55 as a multiplier to the software-development cost to derive the total system-development cost. In a follow-on ROI study [SAVI 2015b], the commercial tool SEER was used to build a SEER-SEM and SEER-H model of a Boeing 777-200 to explicitly estimate the cost of the non-software portion of the system and compare it to both publicly available data and the estimates of the original SAVI ROI study presented in this report. The follow-on study confirms that the cost multiplier of 1.55 was acceptable for the 2010 time frame. Unfortunately, the software count increases while the physical parts count remains stable, resulting in software increasing from 66% in 2010 to 88% by 2024 of the total system-development cost. By 2024, then, the cost multiplier will be 1.12.

If we plot the cost across different reuse percentages, we can see that the software-development cost grows linearly with a decrease in reuse percentage. Figure 2 illustrates this cost for the three system sizes' nominal and lower bound values.

---

6 COCOMO computes the cost at full granularity, and we use the exact values for all calculations. However, for readability purposes, we present the cost estimates rounded off to millions of US$.

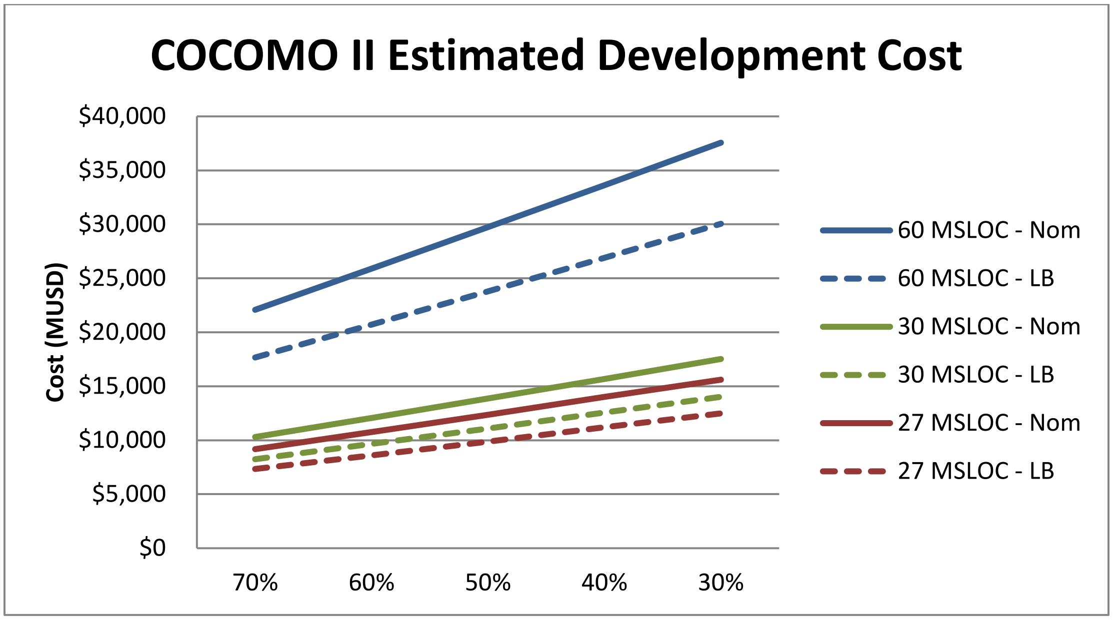

Figure 2: Total System-Development Cost as a Function of Reuse

Table 5 shows the total system-development cost. We used these numbers throughout the remainder of the ROI analysis. Again, we present the nominal, lower bound, and upper bound estimates.

Table 5: Estimated Total System-Development Cost, Including Hardware, in Millions of US$

MSLOC	Reuse	Nominal	Lower Bound	Upper Bound
27	70%	$9,176	$7,341	$11,471
	60%	$10,755	$8,604	$13,444
	50%	$12,356	$9,884	$15,444
	40%	$14,018	$11,213	$17,522
	30%	$15,609	$12,489	$19,513
30	70%	$10,304	$8,244	$12,880
	60%	$12,077	$9,661	$15,096
	50%	$13,873	$11,099	$17,341
	40%	$15,691	$12,552	$19,613
	30%	$17,527	$14,023	$21,909
60	70%	$22,083	$17,666	$27,604
	60%	$25,882	$20,704	$32,352
	50%	$29,732	$23,784	$37,164
	40%	$33,626	$26,901	$42,033
	30%	$37,564	$30,051	$46,955

5 Rework Cost-Avoidance Estimates

We based the rework cost-avoidance estimates on three figures:

1. the rework cost as a percentage of total system-development cost
2. the percentage of the rework cost that is attributable to given types of errors (in our case, we focused on requirements errors.)
3. the percentage of the rework cost that reflects the defect-removal efficiency of a virtual-integration practice based on SAVI

The rework fraction of total software-development work can be as high as 60% to 80% for very large projects [Basili 1994, 2001; Jones 1996; Cross 2002]. To determine the rework cost as a percentage of total system-development cost, we used 50% as an approximation for software-development cost, drawn from software being 66% of system-development cost and rework being around 70% of software-development cost. We also chose 30% as a more conservative number.

5.1 Phase-Based Rework Cost Percentages

To estimate the percentage of rework cost that is attributable to requirements errors, we used the percentage of defects introduced and removed in various phases as well as phase-based rework cost factors.

Researchers have carried out a number of studies to determine where defects are introduced in the development lifecycle, when these defects are discovered, and the resulting rework cost. We limited ourselves here to work previously performed by the National Institute of Standards and Technology (NIST), Galin, Boehm, and Dabney [Boehm 1981, RTI 2002, Dabney 2003, Galin 2004]. The NIST data primarily focuses on IT applications, while the other studies draw on data from safety-critical systems. Findings for the percentages of defects introduced and discovered were quite consistent across these studies, with the exception that higher leakage rates into operation are acceptable in IT systems.

Table 6 shows the percentages of defect introduction and discovery that we used for this ROI study. The Row Total column shows the percentages of defects introduced in each development phase. Each row shows the distribution of each percentage across phases, and all of the entries in a row add up to the row total. For example, 35.25% of all defects are requirements-related defects, while 16.5% of all defects are requirements errors that are detected during testing.

The percentages reflect the lower defect-leakage rates of 2.5% into operation for safety-critical systems. Since the rework cost estimates derived from the COCOMO II model include only the cost of rework through integration, we needed to take only those percentages into account. We normalized the defect percentages so that they add up to 100%.

Table 6: Phase-Based Percentages of Introduced and Detected Defects

Phase	Requirements	Design	Code	Test	Integration	*Row Total*
Requirements	1.00%	1.25%	5.50%	16.50%	11.00%	*35.25%*
Design		1.25%	5.50%	17.00%	11.00%	*34.75%*
Code			5.50%	7.50%	7.00%	*20.00%*
Test				5.00%	2.50%	*7.50%*
Integration					2.50%	*2.50%*
Sum	*1.00%*	*2.50%*	*16.50%*	*46.00%*	*34.00%*	*100.00%*

The same studies by NIST, Galin, Boehm, and Dabney provide rework cost factors [Boehm 1981, RTI 2002, Dabney 2003, Galin 2004]. For the purpose of our study, we used Dabney's data from a study analyzing ROI and defects for the NASA IV&V facility. This domain and its applications have characteristics that are closer to the avionics domain than those of the other sources discussed. This data was further corroborated by recent experiences of the SAVI participant companies.

Table 7 lists the rework cost factors for these studies with [Boehm 1981] and [Galin 2004], shown in the same column due to their similarity. The studies used different phase breakdowns, which we indicate with asterisks in the table. The table shows that, according to Dabney's data, it costs 130 times more to detect and remove a requirements fault at integration than in the requirements phase. According to Dabney, for a fault introduced in coding and removed at the time of integration, the corresponding escalation in cost is 13 times more. SAVI focuses on reducing faults attributed to requirements; for our purposes, we applied the multipliers in Column 3 of Table 7.

Table 7: Defect-Removal Cost, Given the Phase of Origin

Phase	Relative Defect-Removal Cost of Each Phase of Origin														
	Requirements			Design			Coding			Unit Test			Integration		
	[RTI 2002]	[Boehm 1981, Galin 2004]	[Dabney 2003]	[RTI 2002]	[Boehm 1981, Galin 2004]	[Dabney 2003]	[RTI 2002]	[Boehm 1981, Galin 2004]	[Dabney 2003]	[RTI 2002]	[Boehm 1981, Galin 2004]	[Dabney 2003]	[RTI 2002]	[Boehm 1981, Galin 2004]	[Dabney 2003]
Requirements	1	1	1												
Design	1	2.5	5	1	1	1									
Unit Coding	5	6.5	10	5	2.5	2	1	*	1		1				
Unit Test	10	*	50	10	*	10	10	*	5		*	1			
Integration	10	16	130	10	6.4	26	10	*	13	1	2.5	3	*	1	1
System/ Acceptance Test	15	40	*	15	16	*	20	*		10	6.2	*	*	2.5	*
Operation	30	110	368	30	44	64	30	*	37	20	17	7	*	6.9	3

We multiplied the defect percentages from Table 6 by the Dabney rework cost factors shown in Table 7. The resulting numbers represent nominal rework costs and are shown in Table 8. We then used 29.345, the total of all nominal rework cost units, to determine the rework cost for phase-specific defects as percentages of the total rework cost. Table 8 lists those percentages in the last column. The normalization of the leakage percentages in Table 6 has no effect on the resulting percentages. For this ROI study, we focused on the rework cost of requirements defects and used 79% for the requirements-related rework cost percentage in the rework cost-avoidance formula that we introduced in Section 2.

Table 8: Nominal Phase-Based Rework Costs and Percentages

Phase	Reqts.	Design	Code	Test	Integration	Row Total	Percentage
Requirements	0.010	0.063	0.550	8.250	14.300	23.173	78.97%
Design		0.013	0.110	1.700	2.860	4.683	15.96%
Code			0.055	0.375	0.910	1.340	4.57%
Test				0.050	0.075	0.125	0.43%
Integration					0.025	0.025	0.09%
Total	*0.010*	*0.075*	*0.715*	*10.375*	*18.170*	*29.345*	*100.00%*

5.2 Defect Removal Efficiency

Understanding the characteristics of defects is helpful in determining the efficiency of defect removal. We adopted Hayes's fault taxonomy [Miller 1995], listed in Table 9, as a basis for estimating the defect-removal efficiency of the SAVI approach compared to current practices.

We analyzed the types of faults (defects) in Hayes's taxonomy and evaluated how much SAVI will be able to improve a development team's ability to detect and remove a fault early and thus prevent it from flowing downstream. Table 9 shows the anticipated effects of this process. The second column denotes the fraction of faults for each fault class out of the total set of faults based on Hayes's taxonomy [Hayes 2003]. Hayes's data does not suggest distributions of subfaults. In the absence of empirical data and for the purpose of our analysis, we assume that subfaults are uniformly distributed within their major fault class. For example, we can trace 32.9% of faults to Major Fault Class 1.2, which consists of three subfault classes, where each subclass contains one-third of the faults of the major fault class or 11% (see Column 5) of all faults in that fault class.

Industry members of the SAVI project discussed and estimated the reduction of faults by detection earlier in the development lifecycle[7] using both SAVI and insights gained from the proof-of-concept demonstration project, which has shown the feasibility of detecting different types of defects earlier in the lifecycle. For each subfault class, the potential effects include the following with the chosen multiplier shown in column 4 of Table 10:

- SAVI will successfully detect all faults (multiplier = 1).
- SAVI will successfully detect some faults (multiplier = 0.5).
- SAVI will have no impact (multiplier = 0.0).

[7] The number of faults being introduced will probably not change. However, the primary goal is to detect and remove faults early in the development lifecycle.

Using SAVI practices would result in removing a total of 66% of the requirements defects earlier in the lifecycle. In addition to the derived estimates, we included a skeptical scenario in which we reduced all estimates by 50%. In this scenario, SAVI practices would help remove a total of 33% of requirements defects in their phase of origin.

Table 9: Hayes's Fault Taxonomy [Hayes 2003, p. 52]

Major Fault	Subfaults	Description of Subfaults
1.1 Incompleteness	1.1.1 Incomplete Decomposition	Failure to adequately decompose a more abstract specification
	1.1.2 Incomplete Requirements Description	Failure to fully describe all requirements of a function
1.2 Omitted/Missing	1.2.1 Omitted Requirements	Failure to specify one or more of the next lower levels of abstraction of a higher level specified
	1.2.2 Missing External Constants	Specification of a missing value or variable in a requirement
	1.2.3 Missing Description of Initial System State	Failure to specify the initial system state, when that state is not equal to 0
1.3 Incorrect	1.3.1 Incorrect External Constants	Specification of an incorrect value or variable in a requirement
	1.3.2 Incorrect Input or Output Descriptions	Failure to fully describe system input or output
	1.3.3 Incorrect Description of Initial System State	Failure to specify the initial system state when that state is not equal to 0
	1.3.4 Incorrect Assignment of Resources	[Over- or understating] the computing resources assigned to a specification
1.4 Ambiguous	1.4.1 Improper Translation	Failure to carry detailed requirements through [the] decomposition process, resulting in ambiguity in the specification
	1.4.2 Lack of Clarity	Requirement [that] is difficult to understand or has a lack of clarity, and is therefore ambiguous
1.5 Infeasible		Requirement [that] is unfeasible or impossible to achieve, given other system factors, e.g., process speed, memory available
1.6 Inconsistent	1.6.1 External Conflicts	Requirements that are pair-wise incompatible
	1.6.2 Internal Conflicts	Requirements of cooperating systems, or parent/embedded systems, which taken pair-wise are incompatible
1.7 Over-Specification		Requirements or specification limits that are excessive for the operational need, causing additional system cost
1.8 Not Traceable		Requirement that cannot be traced to previous or subsequent phases
1.9 [Reserved for Future]		Requirement that is specified but difficult to achieve (The requirements statement or functional description cannot be true in the reasonable lifetime of the product.)
1.10 Non-Verifiable		Requirements statement or functional description [that] cannot be verified by any reasonable testing methods; process exists to test satisfaction of each requirement (Every requirement is specified behaviorally.)

Major Fault	Subfaults	Description of Subfaults
1.11 Misplaced		Information that is in a different section in [the] requirements document
1.12 Intentional Deviation		Requirement that is specified at [a] higher level but intentionally deviated at lower level from specifications
1.13 Redundant or Duplicate		Requirement [that] was already specified elsewhere in the specification

Table 10: Expected Removal Efficiency of Faults and Defects When Deploying SAVI

Major Fault	Fraction	Effect with SAVI	Multi-plier	SAVI Esti-mate	Skepti-cal
1.1 Incompleteness	0.209	1.1.1 No impact	0	0	0
		1.1.2 No impact	0	0	0
1.2 Omitted or Missing	0.329	1.2.1 SAVI likely to prevent all	1	0.11	0.055
		1.2.2 SAVI likely to prevent all	1	0.11	0.055
		1.2.3 SAVI likely to prevent all	1	0.11	0.055
1.3 Incorrect	0.239	1.3.1 SAVI likely to prevent some	0.5	0.06	0.03
		1.3.2 SAVI likely to prevent all	1	0.12	0.06
1.4 Ambiguous	0.061	1.4.1 SAVI likely to prevent all	1	0.03	0.015
		1.4.2 SAVI likely to prevent all	1	0.03	0.015
1.5 Infeasible	0.014	SAVI likely to prevent all	1	0.01	0.005
1.6 Inconsistent	0.047	1.6.1 SAVI likely to prevent all	1	0.02	0.01
		1.6.2 SAVI likely to prevent all	1	0.02	0.01
1.7 Over-Specification	0.063	No impact	0	0	0
1.8 Not Traceable	0.014	SAVI likely to prevent all	1	0.01	0.005
1.9 [Reserved for Future]	-----	No impact	0	0	0
1.10 Non-Verifiable	0.005	SAVI likely to prevent all	1	0.01	0.005
1.11 Misplaced	0.007	SAVI likely to prevent some	0.5	0.01	0.005
1.12 Intentional Deviation	0.007	SAVI likely to prevent all	1	0.01	0.005
1.13 Redundant or Duplicate	0.005	No impact	0	0	0
Total	*1.000*	*NA*	*NA*	*0.66*	*0.33*

5.3 Estimate of Cost Savings Due to Rework Avoidance

In this section, we present the estimated cost savings attributed to rework avoidance through early discovery of requirements defects. The estimates are based on the formula presented in Section 2 (see page 4). We calculated these estimates by applying the factors presented in this section to the total system-development costs presented in Table 5. We are effectively calculating the cost difference between using the traditional development methods reflected in COCOMO II and the effects of using SAVI, which aims to reduce requirements and design defects through upfront modeling and validation and attempts to prevent these defects from flowing down to later phases where they may require significant rework efforts and increase cost.

*For convenience, we include calculations for reuse ranging from 30% to 70%, based on 30% and 50% of rework and 33% and 66% removal efficiency, with both lower bound (LB), nominal (NOM), and upper bound (UB) estimates (see **Error! Reference source not found.**). The resulting numbers represent millions of US$. We have plotted the rework cost-avoidance estimates in terms of changing reuse percentages based on nominal total system-development cost estimates, 70% reuse, and 33% removal efficiency in*

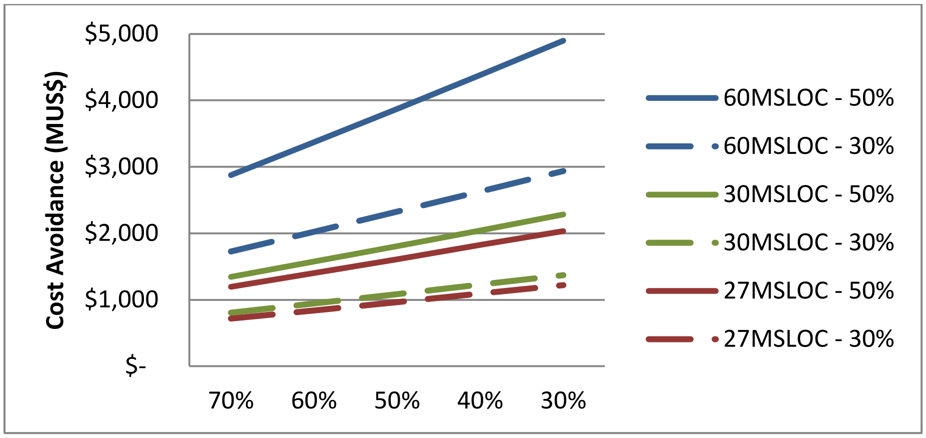

Figure 3. As expected, the plot shows linear growth in cost savings due to the linear growth in total system-development cost, as shown in Figure 2 (page 14).

Table 11: Avoided Cost as a Function of Rework and Software Reuse

MSLOC	% Reuse	% Rework	Removal Efficiency in Millions of US$					
			33%			66%		
			Nom	LB	UB	Nom	LB	UB
27	30%	30%	$1,220	$976	$1,526	$2,441	$1,953	$3,051
		50%	$2,034	$1,627	$2,543	$4,068	$3,255	$5,085
	40%	30%	$1,096	$877	$1,370	$2,192	$1,753	$2,740
		50%	$1,827	$1,461	$2,283	$3,653	$2,922	$4,566
	50%	30%	$966	$773	$1,207	$1,932	$1,546	$2,415
		50%	$1,610	$1,288	$2,012	$3,220	$2,576	$4,025
	60%	30%	$841	$673	$1,051	$1,682	$1,345	$2,102
		50%	$1,401	$1,121	$1,752	$2,803	$2,242	$3,504
	70%	30%	$717	$574	$897	$1,435	$1,148	$1,794
		50%	$1,196	$956	$1,495	$2,391	$1,913	$2,989
30	30%	30%	$1,370	$1,096	$1,713	$2,740	$2,193	$3,426
		50%	$2,284	$1,827	$2,855	$4,567	$3,654	$5,710
	40%	30%	$1,227	$981	$1,533	$2,453	$1,963	$3,067
		50%	$2,045	$1,636	$2,556	$4,089	$3,271	$5,111
	50%	30%	$1,085	$868	$1,356	$2,169	$1,735	$2,712
		50%	$1,808	$1,446	$2,260	$3,615	$2,892	$4,519
	60%	30%	$944	$755	$1,180	$1,888	$1,511	$2,360
		50%	$1,574	$1,259	$1,967	$3,147	$2,518	$3,934
	70%	30%	$806	$645	$1,007	$1,611	$1,289	$2,014
		50%	$1,343	$1,074	$1,678	$2,685	$2,148	$3,357

Table 11, Continued

MSLOC	% Reuse	% Rework	Removal Efficiency in Millions of US$					
			33%			66%		
			Nom	LB	UB	Nom	LB	UB
60	30%	30%	$2,937	$2,349	$3,671	$5,874	$4,699	$7,342
		50%	$4,895	$3,916	$6,118	$9,789	$7,831	$12,237
	40%	30%	$2,629	$2,103	$3,286	$5,258	$4,206	$6,572
		50%	$4,381	$3,505	$5,477	$8,763	$7,010	$10,954
	50%	30%	$2,324	$1,859	$2,906	$4,649	$3,719	$5,811
		50%	$3,874	$3,099	$4,843	$7,748	$6,198	$9,685
	60%	30%	$2,023	$1,619	$2,529	$4,047	$3,237	$5,059
		50%	$3,372	$2,698	$4,216	$6,745	$5,396	$8,431
	70%	30%	$1,726	$1,381	$2,158	$3,453	$2,762	$4,316
		50%	$2,877	$2,302	$3,597	$5,755	$4,604	$7,194

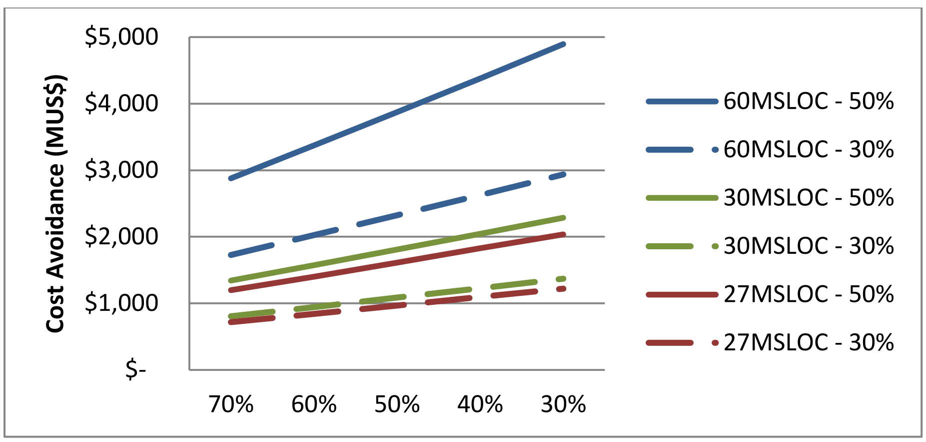

Figure 3: Rework Cost-Avoidance as a Function of Reuse for Three Project Sizes with 30% and 50% Rework

6 ROI Estimates

The investment under consideration is the cost of maturing and transitioning SAVI into existing practice by SAVI-member companies during the multi-year SAVI initiative. This investment has been estimated to be $86M.

The arithmetic and logarithmic return on investment (ROI) is calculated as $ROI_a = \frac{V_f - V_i}{V_i}$ and $ROI_l = \ln(\frac{V_f}{V_i})$, where V_f and V_i represent the value and the investment, respectively. We computed the ROI_a and ROI_l, where V_i equals $86 million, using the nominal values of cost avoidance for the various scenarios in **Error! Reference source not found.**.

Net present value (NPV), which is also known as net present worth, is useful when appraising a long-term project. It is computed as $NPV = \frac{R_t}{(1+i)^t}$, where *t* denotes the time of the cash flow, *i* is the rate of return that could be earned on an investment in the financial markets with similar risk, and R_t is the net cash flow (i.e., the amount of cash, inflow minus outflow) at time *t*. Inflow is computed as the cost avoided at time *t* (i.e., we compute how the total cost avoided is distributed over time).

Based on a project starting in 2010 (t = 0) and finishing in 2018 (t = 8), in which software starts to be developed in 2014, we aligned the development phases and computed the percentage of rework cost avoided in each phase as follows:

- Requirements, 2014, 0.04% (computed as 0.010/23.173)
- Design, 2015, 0.27%
- Implementation, 2016, 2.37%
- Test, 2017, 35.60%
- Integration, 2018, 61.71%

The values for NPV and present value (PV) are shown in Table 12 for a defect-removal efficiency of 33% and in Table 13 for a defect-removal efficiency of 66%. The tables present the data for all three system sizes, five reuse percentages, and two rework percentages used in previous calculations. The columns show the arithmetic ROI, the logarithmic ROI, NPV, and PV for the years 2010 through 2018.

Table 12: Computed NPV and PV for 33% Defect-Removal Efficiency

MSLOC	% Reuse	% Rework	Millions of US$ for a Defect-Removal Efficiency of 33%											
			ROI_a	*ROI_I*	NPV	PV								
						2010	2011	2012	2013	2014	2015	2016	2017	2018
27	30	30	13.2	2.65	$507	–$86	$0	$0	$0	$0	$2	$16	$223	$351
		50	22.6	3.16	$902	–$86	$0	$0	$0	$1	$3	$27	$372	$586
	40	30	11.7	2.54	$447	–$86	$0	$0	$0	$0	$2	$15	$200	$315
		50	20.2	3.06	$802	–$86	$0	$0	$0	$1	$3	$24	$334	$526
	50	30	10.2	2.42	$383	–$86	$0	$0	$0	$0	$2	$13	$176	$278
		50	17.7	2.93	$696	–$86	$0	$0	$0	$0	$3	$22	$294	$463
	60	30	8.8	2.28	$323	–$86	$0	$0	$0	$0	$1	$11	$154	$242
		50	15.3	2.79	$595	–$86	$0	$0	$0	$0	$2	$19	$256	$403
	70	30	7.3	2.12	$263	–$86	$0	$0	$0	$0	$1	$10	$131	$207
		50	12.9	2.63	$495	–$86	$0	$0	$0	$0	$2	$16	$218	$344
30	30	30	14.9	2.77	$580	–$86	$0	$0	$0	$0	$2	$18	$250	$394
		50	25.6	3.28	$1,024	–$86	$0	$0	$0	$1	$4	$31	$417	$657
	40	30	13.3	2.66	$510	–$86	$0	$0	$0	$0	$2	$16	$224	$353
		50	22.8	3.17	$908	–$86	$0	$0	$0	$1	$3	$27	$374	$589
	50	30	11.6	2.53	$441	–$86	$0	$0	$0	$0	$2	$15	$198	$312
		50	20.0	3.05	$792	–$86	$0	$0	$0	$1	$3	$24	$330	$520
	60	30	10.0	2.40	$373	–$86	$0	$0	$0	$0	$2	$13	$173	$272
		50	17.3	2.91	$679	–$86	$0	$0	$0	$0	$3	$21	$288	$453
	70	30	8.4	2.24	$305	–$86	$0	$0	$0	$0	$1	$11	$147	$232
		50	14.6	2.75	$566	–$86	$0	$0	$0	$0	$2	$18	$245	$387
60	30	30	33.1	3.53	$1,341	–$86	$0	$0	$0	$1	$5	$39	$537	$845
		50	55.9	4.04	$2,293	–$86	$0	$0	$0	$1	$8	$66	$894	$1,409
	40	30	29.6	3.42	$1,192	–$86	$0	$0	$0	$1	$4	$35	$480	$757
		50	49.9	3.93	$2,043	–$86	$0	$0	$0	$1	$7	$59	$800	$1,261
	50	30	26.0	3.30	$1,044	–$86	$0	$0	$0	$1	$4	$31	$425	$669
		50	44.0	3.81	$1,797	–$86	$0	$0	$0	$1	$6	$52	$708	$1,115
	60	30	22.5	3.16	$897	–$86	$0	$0	$0	$1	$3	$27	$370	$583
		50	38.2	3.67	$1,553	–$86	$0	$0	$0	$1	$6	$45	$616	$971
	70	30	19.1	3.00	$753	–$86	$0	$0	$0	$1	$3	$23	$315	$497
		50	32.5	3.51	$1,312	–$86	$0	$0	$0	$1	$5	$39	$526	$828

Table 13: Computed NPV and PV for 66% Defect-Removal Efficiency

MSLOC	% Reuse	% Rework	Millions of US$ for Defect-Removal Efficiency of 66%											
			ROI_a	ROI_l	NPV	PV								
						2010	2011	2012	2013	2014	2015	2016	2017	2018
27	30	30	27.4	3.35	$1,100	–$86	$0	$0	$0	$1	$4	$33	$446	$703
		50	46.3	3.86	$1,891	–$86	$0	$0	$0	$1	$7	$54	$743	$1,171
	40	30	24.5	3.24	$979	–$86	$0	$0	$0	$1	$4	$29	$400	$631
		50	41.5	3.75	$1,689	–$86	$0	$0	$0	$1	$6	$49	$667	$1,052
	50	30	21.5	3.11	$853	–$86	$0	$0	$0	$1	$3	$26	$353	$556
		50	36.4	3.62	$1,479	–$86	$0	$0	$0	$1	$5	$43	$588	$927
	60	30	18.6	2.97	$731	–$86	$0	$0	$0	$0	$3	$23	$307	$484
		50	31.6	3.48	$1,276	–$86	$0	$0	$0	$1	$5	$38	$512	$807
	70	30	15.7	2.81	$611	–$86	$0	$0	$0	$0	$2	$19	$262	$413
		50	26.8	3.33	$1,076	–$86	$0	$0	$0	$1	$4	$32	$437	$688
30	30	30	30.9	3.46	$1,246	–$86	$0	$0	$0	$1	$5	$37	$501	$789
		50	52.1	3.97	$2,134	–$86	$0	$0	$0	$1	$8	$61	$834	$1,315
	40	30	27.5	3.35	$1,106	–$86	$0	$0	$0	$1	$4	$33	$448	$706
		50	46.5	3.86	$1,901	–$86	$0	$0	$0	$1	$7	$55	$747	$1,177
	50	30	24.2	3.23	$968	–$86	$0	$0	$0	$1	$4	$29	$396	$625
		50	41.0	3.74	$1,671	–$86	$0	$0	$0	$1	$6	$48	$661	$1,041
	60	30	21.0	3.09	$832	–$86	$0	$0	$0	$1	$3	$25	$345	$544
		50	35.6	3.60	$1,443	–$86	$0	$0	$0	$1	$5	$42	$575	$906
	70	30	17.7	2.93	$697	–$86	$0	$0	$0	$0	$3	$22	$294	$464
		50	30.2	3.44	$1,219	–$86	$0	$0	$0	$1	$4	$36	$491	$773
60	30	30	67.3	4.22	$2,768	–$86	$0	$0	$0	$2	$10	$79	$1,073	$1,691
		50	112.8	4.73	$4,671	–$86	$0	$0	$0	$3	$16	$131	$1,788	$2,818
	40	30	60.1	4.11	$2,469	–$86	$0	$0	$0	$2	$9	$70	$961	$1,514
		50	100.9	4.62	$4,172	–$86	$0	$0	$0	$3	$15	$117	$1,601	$2,523
	50	30	53.1	3.99	$2,173	–$86	$0	$0	$0	$1	$8	$62	$849	$1,338
		50	89.1	4.50	$3,679	–$86	$0	$0	$0	$2	$13	$104	$1,416	$2,231
	60	30	46.1	3.85	$1,881	–$86	$0	$0	$0	$1	$7	$54	$739	$1,165
		50	77.4	4.36	$3,192	–$86	$0	$0	$0	$2	$11	$90	$1,232	$1,942
	70	30	39.1	3.69	$1,592	–$86	$0	$0	$0	$1	$6	$46	$631	$994
		50	65.9	4.20	$2,711	–$86	$0	$0	$0	$2	$10	$77	$1,051	$1,657

In Figure 4, we examine the trend for Arithmetic ROI (Figure 4), logarithmic ROI (Figure 5), and NPV (Figure 6) as a function of reuse. As expected, the plots show linear progression in all three cases.

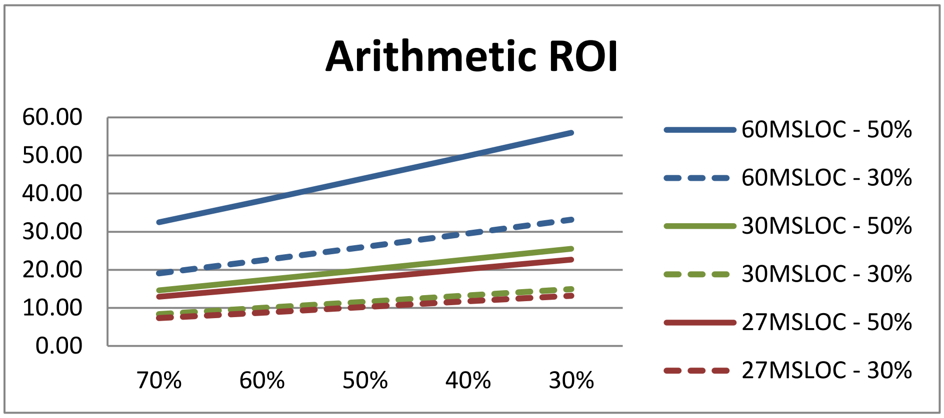

Figure 4: Projected Arithmetic ROI as a Function of Reuse for Three Project Sizes with 30% and 50% Rework

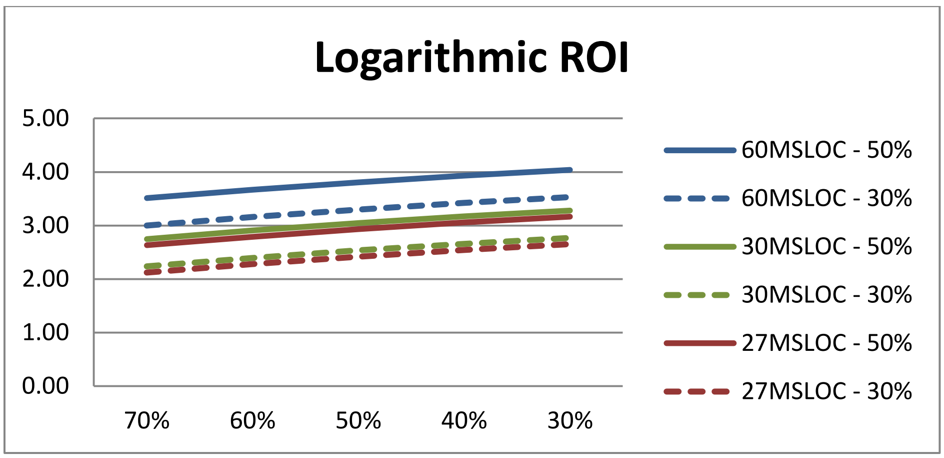

Figure 5: Projected Logarithmic ROI as a Function of Reuse for Three Project Sizes with 30% and 50% Rework

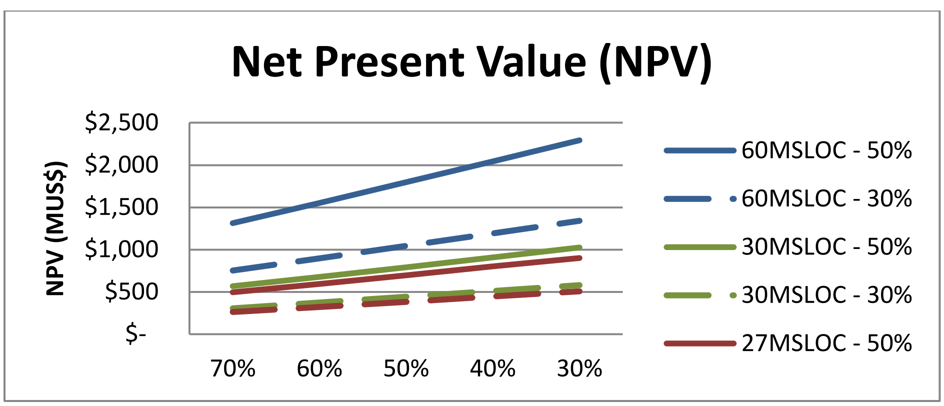

Figure 6: Computed NPV as a Function of Reuse for Three Project Sizes with 30% and 50% Rework

7 Discussion on the Use of COCOMO II

Achieving cost efficiency and high levels of reuse requires that developers design a system for targeted levels of reuse and avoid costly alterations and modifications. COCOMO II supports reuse in two ways by providing: (1) a proactive design-for-reuse switch and (2) a specification of the complexity of achieving reuse when invoking a system into a new system (e.g., percentage of code, design, integration modified, software understanding, and automatic-translation capability). Both approaches increase the EAF and, thus, the cost. We used the second approach with nominal values for reuse. We did not set a specific design-for-reuse switch, since different organizations adopt different strategies for this purpose. Reuse directly affects development cost as well as any development gain. Computed gains should be adjusted according to any increase or decrease in development cost.

We used the projected cost of one man-month in 2014 of $28,200/month. Unfortunately, COCOMO II does not support any compensation for annual inflation, but at least two alternative approaches do incorporate this compensation. One alternative is, for a project of *n* years, to further decompose each system into *n* subsystems to reflect the development cost for each year. This decomposition can also be mapped to system-development phases and work-breakdown structures in COCOMO II. Another alternative, a low-fidelity approach, is to use the estimated average salary over the same period, which assumes that work is uniformly distributed over time. Our choice to use COCOMO II was motivated by its ability to decompose a system and characterize cost drivers at the subsystem level for avionics systems. COCOMO II is also a proven tool, having been used widely in industry and calibrated against a number of projects.

The COCOMO II data used in this report focuses on the *system*-development cost—a cost that begins with the design of the system and continues until its first delivery and deployment. Thus, our data does not address ongoing lifecycle costs such as maintenance. For long-lived systems (those exceeding 10 years of use) and systems experiencing frequent changes, the sustainment cost is significant. For example, consider the lifecycle costs for a computer system for which the hardware has become obsolete and the software needs to be revised and recertified on new target hardware. The system may also be subject to changes in federal regulations that require incorporating new software functionality. A model-driven approach as outlined by SAVI will undoubtedly affect system-maintenance cost in a positive way, but it is unclear how much of an effect this approach will have.

There are alternatives to COCOMO II for system cost estimation. The REVised Intermediate COCOMO (REVIC) is a cost-estimation tool developed by R. Kile and the U.S. Air Force Cost Analysis Agency that is specific to DoD needs [DoD 1995]. It has been calibrated using completed DoD projects (development phase only). REVIC takes the same inputs as COCOMO II. On average, the values predicted by the effort and schedule equations in REVIC are higher than those predicted in COCOMO II. We did not use REVIC as a foundation for this study for the following reasons:

- COCOMO II is a newer model than REVIC, which was developed and released in the early 1990s.

- REVIC consistently provides higher estimates than COCOMO II; if the goal is to use conservative estimates and avoid inflated gains, COCOMO II is better suited because REVIC produces higher projected gains.
- REVIC is calibrated for DoD projects, and our focus in the SAVI initiative is commercial avionics.[8]

A second alternative, the Constructive Systems Engineering Cost Model (COSYSMO), based on COCOMO II, can help planners reason about the economic aspects and consequences of systems engineering on projects [COSYSMO 2011]. For the purpose of our application, COCOMO II was deemed sufficient.

Cost estimation using function-point analysis, created by Albrecht [Albrecht 1979], is another viable technique with significant applications in industry [Garmus 2001, Jones 2007]. Albrecht's objective was to create a metric for software productivity and quality

- in any known programming language
- in any combination of languages
- across all classes of software

Function-point analysis is intended for use in discussions with clients, contracts, large-scale statistical analysis, and value analysis. SLOC metrics are intrinsically language dependent. For example, considering software productivity should take context into account: Does the software use a low-level language or a high-level language, and can a given high-level language be further classified into a procedural, logic-based, or object-oriented language? A function point consists of the weighted totals of five external aspects of software applications: types of inputs to the application, outputs that leave the application, inquiries that users can make, logical files that an application maintains, and interfaces to other applications.

Given the data available to us, we can apply function-point analysis in our context. The analysis would be driven by function points as the primary metric of the complexity and size of a system-development effort. There are guidelines and data from projects to aid in estimating the number of function points. Furthermore, we can use conversion factors to convert between SLOC and the number of function points. We conducted a subset of our calculations here using Capers Jones's data, including some based on defects per function point from industrial projects [Jones 2007]. From those calculations, we observed that the computed cost reduction and ROI numbers are slightly higher than those derived from COCOMO II. In the interest of space, we do not include those calculations in this document.

Our COCOMO II analysis is agnostic to the organizational structure and business drivers of the main contractor, suppliers, and subcontractors. It reports the numbers for the entire project but does not elaborate on the benefits to the various suppliers and subcontractors. While it is easy to see that the SAVI approach benefits contractors at all levels, it would be worthwhile to conduct a refined ROI analysis from the perspective of the entire acquisition and development lifecycle,

8 Commercial avionics systems have safety-critical requirements. In that respect, they present the same challenges as many weapon systems acquired by the DoD.

broken down by stakeholders and contractors. We should also exercise the effects of more efficient defect removal on virtual integration. Such an analysis should reflect current and new acquisition practices.

8 Conclusion

Software in avionics continues to grow in complexity and size, evidenced by the number of subsystems, their dependencies, and increased functionality. Current state-of-practice methods, processes, and techniques do not scale well, causing projects to overrun schedules and budgets, primarily due to the scale and complexity of systems. The cost of developing large-scale software of the projected size—that will also satisfy high safety-criticality requirements as well as be reusable—is indeed truly large, especially when considering that the total system development cost of a large airframer (aerospace manufacturer) is in the range of $10 to $20 billion

Thus, software is projected to become the dominant cost of developing a new product.

The ROI analysis discussed in this report clearly indicates that an organization could realize significant financial gains by using the SAVI approach.

Our analysis produced the following outcomes and observations:

- Approximately 70% of defects are requirements and design defects, but less than 10% are detected in these early phases. The rework cost of correcting defects introduced in requirements and design but detected late in the development lifecycle is one to two orders of magnitude higher than the cost of correcting them before coding. Requirements-related rework cost amounts to 79% and design-related rework costs 16% of the total-rework cost. In this study, we only considered avoidance of requirements-related defects.
- In the most conservative scenario, for a 27-MSLOC system, the smallest cost avoidance is $717 million (out of an estimated $9.176 billion cost of development, a 7.8% cost savings). This is with 70% reuse, rework cost as 30% of total system-development cost, and a defect-removal efficiency of 33%. The arithmetic and logarithmic (continuously compounded) ROIs are 7.3 and 2.12, with an NPV of $263 million.
- The nominal cost reduction for a 27-MSLOC system is $2.391 billion (out of an estimated $9.176 billion, a 26.1% cost savings), occurring at 70% reuse, rework cost as 50% of total system-development cost, and 66% removal efficiency. The arithmetic and logarithmic ROIs are 26.8 and 3.33, with an NPV of $1.076 billion.
- The cost reduction is linear to the rework cost and removal efficiency. With other factors held constant in a scenario, each unit's increase in removal efficiency for requirements errors resulted in a cost reduction. For example, for the 27-MSLOC system with the highest degree of reuse, each 1% increase in removal efficiency resulted in a cost reduction of $22 million.
- Cost reduction for a given system size is also linear to the amount of reuse.
- Comparing the cost reduction for systems of different size, we observed that the cost reduction for 60 MSLOC was more than twice that for 30 MSLOC. This is to be expected given the more-than-linear increase in interaction complexity in a larger system.

The predicted returns were considered higher than anticipated. This led to several follow-on activities. First, one of the SAVI-system-integrator members obtained an independent assessment by its organization's cost estimating group, which agreed with the findings of this report. Second, this initial ROI study was followed by a second SAVI ROI study [Ward 2011, SAVI 2015a, SAVI 2015b].

In the second study, a Monte Carlo algorithm was used to drive the COCOMO II cost estimation, resulting in a reduced variation of results. In addition, the commercial tool SEER was used to build a SEER-SEM and SEER-H model of a Boeing 777-200 to explicitly estimate the cost of the non-software portion of the system and compare it to both publically available data and the estimates of the original SAVI ROI study presented here. The second study confirms that the cost multiplier of 1.55 was acceptable for 2010. Unfortunately, the software count increases while the physical parts count remains stable, resulting in a software increase from 66% in 2010 to 88% of the total system-development cost by 2024.

The second study also considered tailoring the ROI analysis to reflect a subcontractor. This includes adjusting the scaling factor used to estimate the total cost relative to the software cost, the degree of software reuse, the overall investment in the SAVI technology, and personnel cost factors.

References

URLs are valid as of the publication date of this document.

[Albrecht 1979]
Albrecht, A. J. Measuring Application Development Productivity. Pages 83–92. *Proceedings of the Joint IBM/SHARE/GUIDE Application Development Symposium*. Monterey, California. October 1979.

[Basili 1994]
Basili, V. & Green, S. Software Process Evolution at SEL. *IEEE Software*. Volume 11. Number 4. July/August 1994. Pages 58–66.

[Basili 2001]
Basili, V. et al. *Building an Experience Base for Software Engineering: A Report on the First eWorkshop*. CeBASE. 2001. http://www.cs.umd.edu/~basili/presentations/profes01.pdf

[Boehm 1981]
Boehm, B. W. *Software Engineering Economics*. Prentice Hall. 1981.

[Boehm 2000]
Boehm, B. W. et al. *Software Cost Estimation with COCOMO II*. Prentice Hall. 2000.

[COCOMO II]
COCOMO II. *Center for Systems and Software Engineering Website*. March 14, 2018 [accessed] http://sunset.usc.edu/csse/research/COCOMOII/cocomo_main.html

[COSYSMO 2011]
Constructive Systems Engineering Cost Model (COSYSMO). *Massachusetts Institute of Technology COSYSMO Website*. 2011. March 14, 2018 [accessed] http://cosysmo.mit.edu

[Cross 2002]
Cross, S. E. Message from the Director. In *The Software Engineering Institute 2002 Annual Report*. Software Engineering Institute, Carnegie Mellon University. 2002. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30178

[Dabney 2003]
Dabney, J. B. *Return on Investment of Independent Verification and Validation Study Preliminary Phase 2B Report*. NASA. 2003.

[DoD 1995]
Department of Defense. *The Parametric Cost Estimating Handbook.* Joint Government/Industry Initiative. 1995.

[Fairley 2005]
Fairley, R. E. & Willshire, M. J. Iterative Rework: The Good, the Bad, and the Ugly. *IEEE Computer*. Volume 38. Number 8. September 2005. Pages 34–41.

[Feiler 2009]
Feiler, P. H. et al. *System Architecture Virtual Integration: An Industrial Case Study*. CMU/SEI-2009-TR-017. Software Engineering Institute, Carnegie Mellon University. 2009. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9145

[Feiler 2010]
Feiler, P. et al. System Architecture Virtual Integration: A Case Study. *Embedded Real Time Software and Systems Conference (ERTS2010)*. Toulouse, France. May 2010.
http://web1.see.asso.fr/erts2010/Site/0ANDGY78
/Fichier/PAPIERS%20ERTS%202010%202/ERTS2010_0105_final.pdf

[Galin 2004]
Galin, D. *Software Quality Assurance: From Theory to Implementation*. Pearson/Addison-Wesley. 2004.

[Garmus 2001]
Garmus, D. & Herron, D. *Function Point Analysis: Measurement Practices for Successful Software Projects* (Information Technology Series). Addison-Wesley. 2001.

[Hatton 2005]
Hatton, L. *Estimating Source Lines of Code from Object Code: Windows and Embedded Control Systems*. University of Kingston. 2005.

[Hayes 2003]
Hayes, J. H. Building a Requirement Fault Taxonomy: Experiences from a NASA Verification and Validation Research Project. Pages 49–59. *14th International Symposium on Software Reliability Engineering (ISSRE)*. Denver, Colorado. November 2003.

[Jones 1996]
Jones, C. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill. 1996.

[Jones 2007]
Jones, C. *Estimating Software Costs: Bringing Realism to Estimating*. McGraw-Hill. 2007.

[Lutz 1993]
Lutz, R. R. Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems. Pages 126-133. In *Proceedings of the IEEE International Symposium on Requirements Engineering*. San Diego, California. January 4-6, 1993. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=324825

[Miller 1995]
Mirsky, S. M. et al. *Guidelines for the Verification and Validation of Expert System Software and Conventional Software.* NUREG/CR-6316. U.S. Nuclear Regulatory Commission. 1995.

[Park 1992]
Park, R. E. *Software Size Measurement: A Framework for Counting Source Statements*. CMU/SEI-92-TR-020. Software Engineering Institute, Carnegie Mellon University. 1992. https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=11689

[Potocki de Montalk 1991]
Potocki de Montalk, J. P. Computer Software in Civil Aircraft. Pages 10–16. In *Proceedings of the Sixth Annual Conference on Computer Assurance: Systems Integrity, Software Safety and Process Security (COMPASS'91)*. Gaithersburg, Maryland. June 1991.

[Redman 2010]
Redman, D., et al. Virtual Integration for Improved System Design: The AVSI System Architecture Virtual Integration (SAVI) Program. *Analytic Virtual Integration of Cyber-Physical Systems Workshop, 31st IEEE Real-Time Systems Symposium (RTSS 2010)*. San Diego, California. November 30-December 3, 2010.

[RTI 2002]
RTI International. *The Economic Impacts of Inadequate Infrastructure for Software Testing.* Planning Report 02-3. NIST. 2002.

[SAVI 2015a]
Chilenski, J. J. & Ward, D.T. [editors] ROI Estimation. In *SAVI AFE 59 Report Summary Final Report*. Aerospace Vehicle Systems Institute. Pages 29-31. 2015. http://savi.avsi.aero/wp-content/uploads/sites/2/2015/08/SAVI-AFE59-9-001_Summary_Final_Report.pdf

[SAVI 2015b]
Chilenski, J. J. & Ward, D.T. [editors]. EPoCD Use Case Demonstrations. Aerospace Vehicle Systems Institute. Pages 9-28. 2015. http://savi.avsi.aero/wp-content/uploads/sites/2/2015/08/SAVI-AFE59S1-8-002_Summary_Final_Report.pdf

[U.S. Department of Labor]
Consumer Price Index, All Urban Consumers (CPI-U), U.S. City Average. *U.S. Department of Labor, Bureau of Labor Statistics*. https://www.bls.gov/news.release/cpi.t01.htm

[Ward 2011]
Ward, D. & Helton, S. Estimating Return on Investment for SAVI (a Model-Based Virtual Integration Process). *SAE International Journal of Aerospace.* Volume 4. Number 2. October 18, 2011. Pages 934-943. https://saemobilus.sae.org/content/2011-01-2576

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE April 2018		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE ROI Analysis of the System Architecture Virtual Integration Initiative			5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Jörgen Hansson, Peter Feiler, Steven Helton			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2018-TR-002
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER n/a
11. SUPPLEMENTARY NOTES			
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12B DISTRIBUTION CODE
13. ABSTRACT (MAXIMUM 200 WORDS) The System Architecture Virtual Integration (SAVI) initiative is a multiyear, multimillion dollar program that is developing the capability to virtually integrate systems before designs are implemented and tested on hardware. The purpose of SAVI is to develop a means of countering the costs of exponentially increasing complexity in modern aerospace software systems. The program is sponsored by the Aerospace Vehicle Systems Institute, a research center of the Texas Engineering Experiment Station, which is a member of the Texas A&M University System. This report presents an analysis of the economic effects of the SAVI approach on the development of software-reliant systems for aircraft compared to existing development paradigms. The report describes the detailed inputs and results of a return-on-investment (ROI) analysis to determine the net present value of the investment in the SAVI approach. The ROI is based on rework cost-avoidance attributed to earlier discovery of requirements errors through analysis of virtually integrated models of the embedded software system expressed in the SAE International Architecture Analysis and Design Language (AADL) standard architecture modeling language. The ROI analysis uses conservative estimates of costs and benefits, especially for those parameters that have a proven, strong correlation to overall system-development cost. The results of the analysis, in part, show that the nominal cost reduction for a system that contains 27 million source lines of code would be $2.391 billion (out of an estimated $9.176 billion), a 26.1% cost savings. The original study, reported here, had a follow-on study to validate and further refine the estimated cost savings.			
14. SUBJECT TERMS virtual integration, software development, software defects			15. NUMBER OF PAGES 45
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102