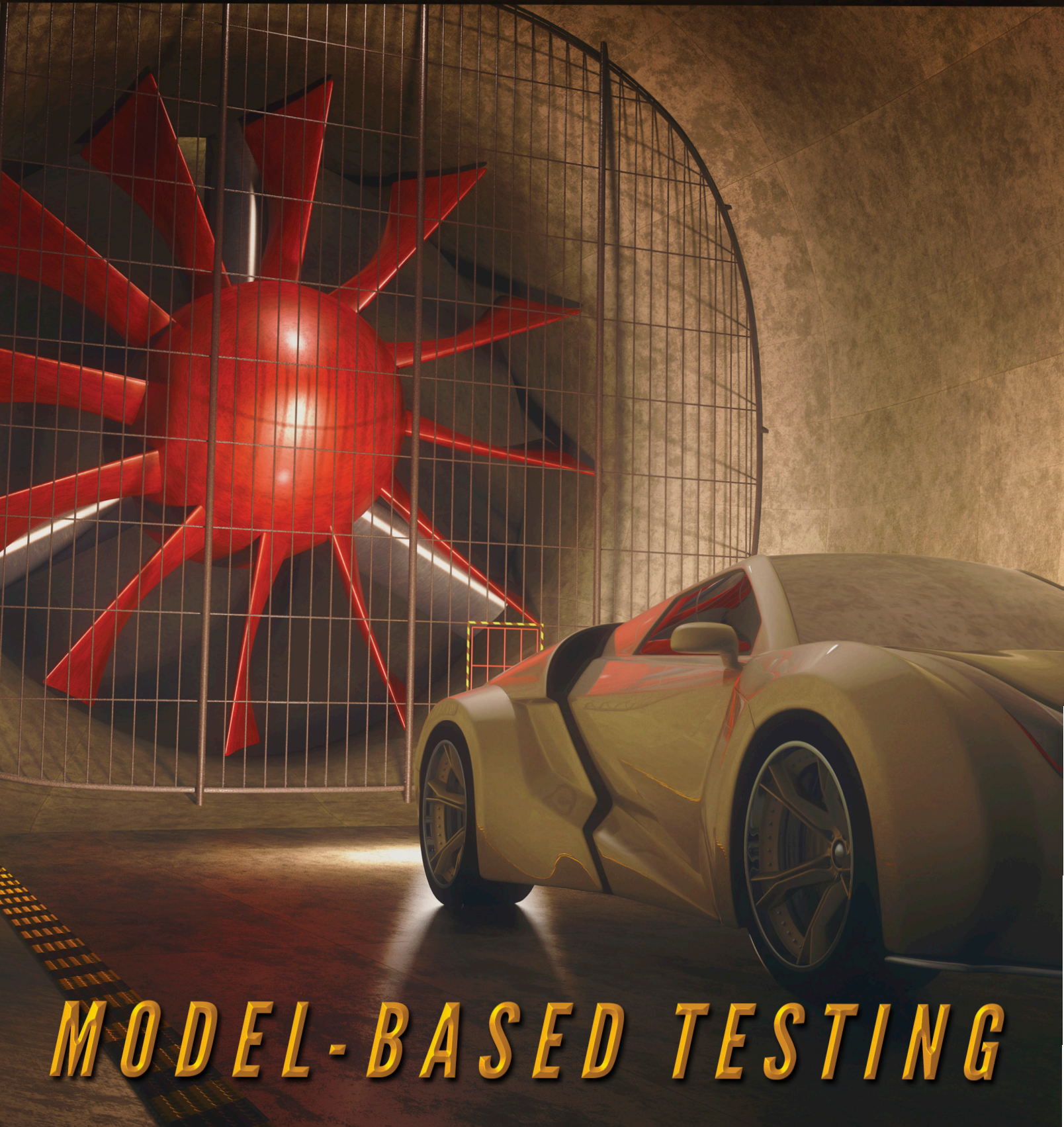
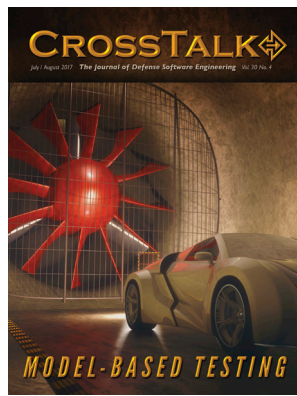


CROSSTALK

July / August 2017 **The Journal of Defense Software Engineering** Vol. 30 No. 4



MODEL-BASED TESTING

Cover Design by
Kent Bingham

Departments

- 3 From the Sponsor
- 37 Upcoming Events
- 38 BackTalk

Model-based Testing

- 4 Testing in O&M: Software Does Not Wear Out, It Degrades**
This article briefly defines verification, validation and testing; outlines testing activities during operations and management, and concludes with the impact of test automation.

By **Jon Hagar**

- 8 Software Estimates That Work**
We propose a three-pronged approach for initial estimating of large new projects for systems and software development organizations that have experienced resource problems. The process itself will generally be unique to each organization and its particular culture, projects, circumstances, people, etc.

By **Louis M. Taff**

- 13 Principles of Software Testing**
There is a myth that if we were really good at programming, there would be no bugs to catch. This is not true. In this paper, I will discuss fundamentals of software testing that expel some of these myths. I will then discuss some specific software testing techniques.

By **Theresa Lynn Sanderfer Brown**

- 17 Combining Agile Practices with CMMI Process Areas: A Case Study**
This case study details how a large enterprise-level program used Agile methods, including Scrum, and implemented the capability maturity model integration (CMMI) at the same time.

By **Dick Carlson and Earl Soukup**

- 22 Using Contracts to Reduce Cybersecurity Risks**
We identify cybersecurity risks inherent in developed and acquired software products as well as software-related service issues and describe how to mitigate these risks through contractual terms and conditions. Finally, we will discuss the conditions that may affect agreed-upon contracts.

by **C. Warren Axelrod**

- 31 Integration of Big Data: Misconceptions, Problems, and Needed Capabilities**
Based on our experience and interactions with both customers and vendors, we discuss here the common misconceptions about Big Data integration and cloud technologies, analyze problems that need to be addressed, and suggest capabilities of the solutions.

by **J. Johnson, J. Folger, T. Stevens and T. Malyuta**

- 34 Getting to Understand Some of the Operations and Maintenance Drivers of Cost**
This article will look at specific cost drivers, including licensing, design and documentation, replenishment planning, data collection, and impacts of the operational environment. It will then look at tools to reduce cost and improve availability.

By **Jonathan W. Powell**

CROSSTALK

NAVAIR Jeff Schwab
309 SMXG Kelly Capener
76 SMXG Mike Jennings

Publisher Justin T. Hill
Article Coordinator Heather Giacalone
Managing Director David Erickson
Technical Program Lead Thayne M. Hill
Managing Editor Mary Harper
Copy Editor Breanna Olaveson
Senior Art Director Kevin Kiernan
Art Director Mary Harper

Phone 801-777-9828

E-mail Crosstalk.Articles@hill.af.mil

Crosstalk Online www.crosstalkonline.org

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); and U.S. Air Force (USAF). USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Ogden-ALC 309 SMXG and Tinker-ALC 76 SMXG.

The USAF Software Technology Support Center (STSC) is the publisher of **CROSSTALK** providing both editorial oversight and technical review of the journal. **CROSSTALK'S** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit www.crosstalkonline.org/subscribe to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CROSSTALK** editorial board prior to publication. Please follow the Author Guidelines, available at www.crosstalkonline.org/submission-guidelines. **CROSSTALK** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CROSSTALK**.

Trademarks and Endorsements: **CROSSTALK** is an authorized publication for members of the DoD. Contents of **CROSSTALK** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK Online Services:

For questions or concerns about crosstalkonline.org web content or functionality contact the **CROSSTALK** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CROSSTALK is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing luminpublishing.com. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

CROSSTALK would like to thank **76 SMXG** for sponsoring this issue.

“All models are wrong; some models are useful.”

“Discovering the unexpected is more important than confirming the known.”

– Statistician George Box



The theme of this issue of CrossTalk is the exciting area of Model Based Testing. Without a doubt every software product manager would jump at the opportunity for delivering a product in less time and lower cost, while knowing that it has undergone a testing regimen sufficient of ensuring the highest quality. Often the effort related to shifting our approaches or processes is interpreted as waste; in regards to implementing model based testing, I can assure you this effort is well worth the investment.

Modeling is a necessary and beneficial part of every software effort. Regardless of the tools or documentation in front of them, every programmer works to convert an internal mental model of a process into working code. From the informality of this internal model to the structured formality of a tool like the Unified Modeling Language (UML), every working piece of software is built upon layers of abstraction and modeling. Settling on the right models and abstractions to solve a particular problem are the meat and potatoes of software engineering.

Modeling is the foundational element essential for the programmer to build a working piece of software. Methods for modeling and testing continue to evolve at a rapid pace in the world of software development. From the formal, top-down modeling favored by UML and the waterfall lifecycle model to the informal models that are continually developed and updated by Agile practitioners, the best models can vary across projects and teams. In the same vein, rigorous testing to requirements and formal Validation and Verification are the best choice for many situations, while Test-Driven Development or test-first programming continue to grow in popularity. In the quest for continual improvement, it will likely prove that adopting the best ideas from the wide variety of modeling and testing practices will result in better processes and products.

I hope that this issue of CrossTalk informs and assists a software community that continues to adapt to address the ever growing complexity of systems. Our teams are being asked every day to achieve a greater level of user experience with impeccable quality in a rapidly changing environment at a lower cost. I would like to encourage all of our community partners to redouble our focus on building relationships focused on learning, sharing, and improving.

Michael Jennings
Group Director, 76 Software Maintenance Group

Testing in O&M

Software Does Not Wear Out, It Degrades

Jon Hagar, Grand Software Testing

Abstract. Software verification and validation remain active during software-system evolution in operations and management. There are actions in test planning and risk analysis that need to take place. This article briefly defines verification, validation and testing. It also outlines testing activities during operations and management, including integrity and risk assessment, regression testing, and handling third-party changes introduced into the operations and management phase. This article concludes with the impact of test automation.

Introduction

Most long-term software development teams understand that a majority of the costs spent on typical products happens during the operations and maintenance (O&M) phase, which can last for years. Additionally, most organizations understand that verification and validation (V&V) along with testing are necessary activities for software products. Unfortunately, V&V and testing may be seen as less important in the O&M phase since the software is “in the field and working.”

However, V&V and testing can provide vital information about any software degradation during the O&M phase. In fact, some projects have more V&V and testing efforts in O&M than in development because of software “evolution.”

The evolution of software in the O&M phase that can necessitate V&V and testing includes:

- Updates to operational concepts, which subject software to new environmental uses that may trigger long-hidden errors.
- Changes to software to add new features, which can have side effects that impact existing logic due to coupling.
- Changes to the software to fix bugs, which in turn will need testing.
- Updates to software elements from third-party vendors, which also require testing.
- Changes to hardware that impact the software.

Years ago, a spacecraft had been in orbit. During its O&M activities it needed a small software update, so a patch was uploaded that consisted of only a couple words. The change was reviewed (using peer review verification), but not subjected to real validation or testing.

After the upload, the spacecraft went dark (stopped communicating) for a few minutes until the system's safing software activated and re-established communications. Thereafter, the team and the whole organization was instructed that no change to either software or hardware in the O&M phase should ever be fielded without being validated, verified or tested first.

Unlike hardware in the O&M phase, which with repeated use can “wear out,” software does not wear out. However, some software experts find that as software evolves, the possible negative effect on the software as a whole can be very similar to the effects of hardware wearing out. The code becomes hard to maintain. It may break and, therefore, be unstable. A set of solutions to minimize the wear-out factor can be provided by V&V or testing. Some of these solutions are examined briefly below.

Defining V&V and Testing

V&V and testing provide information to stakeholders. Verification answers the question “Did we build the product right?” Validation answers the question “Did we build the right product?”

When you verify software, you evaluate whether it satisfies the conditions defined at the beginning of an engineering activity (e.g., checking requirements implemented in a design). When you validate software, you evaluate a product to determine whether it satisfies the user or customer needs.

Classically, V&V uses four approaches: test, inspection (e.g., visual examination), demonstration (e.g., executing tests on a delivered system), and analysis (e.g., modeling) to gather information. V&V and testing provide information on these qualities:

- Meeting all requirements.
- Creating trusted software with no errors.
- Meeting timing expectations.
- Verifying that the user interface works.

V&V and testing include a variety of activities throughout the life cycle to provide information on these qualities and others.

There are generic industry standards on V&V and testing, including IEEE 1012 V&V plans [1] and ISO 29119 software testing [2]. Such standards can be used as a starting point, but they must be tailored for any project, even during O&M. There are other useful industry references to consider, including “Black Box Testing: Techniques for Functional Testing of Software and Systems” [3], “The Art of Software Testing” [4], “Lessons Learned in Software Testing,” [5] and “Systematic Software Testing” [6]. Refer to these standards and references for more information.

V&V and Testing Activities within O&M

While V&V and testing should occur during initial development, many of the qualities assessed remain of interest during the O&M phase. V&V and testing efforts should be reduced and tailored for O&M, but dropping them to zero effort is not advised. A conceptual O&M V&V and testing process flow is shown in Figure 1.

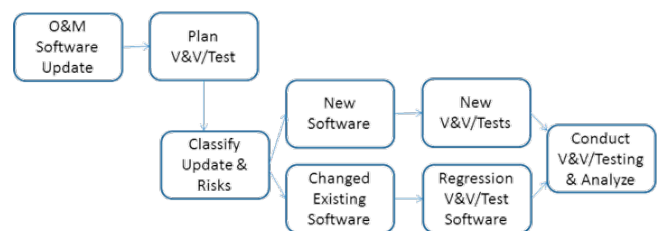


Figure 1: Basic V&V and Testing Flow for Software O&M Evolution

Teams within the O&M cycle should conduct V&V and test planning and life cycle activities as the software evolves for all changes. There is no one best set of activities or tests for O&M, so critically thinking testers are still needed. Planning, classifying the update, and identifying changes in risks for testing are all necessary for an O&M update.

Further, not all changes are equal. Even small changes can have large impacts, as the spacecraft story illustrated. New team members may not understand historic software designs and features, so errors can be introduced even during a small change.

Many programmers also suffer from the “not invented here” syndrome, so they will change software to make it better even when the change is not necessary. These changes usually happen outside the configuration control system unless good configuration management controls are in practice. Such added refactoring changes can be dangerous unless properly controlled, since their impact may not be known or fully understood. This results in incomplete V&V and testing. As Figure 1 shows, risk analysis and V&V are necessary for changed and new software. The following are a few generic recommended test activities (further explained in “Software Test Attacks to Break Mobile and Embedded Devices” [7]).

Integrity and Risk Assessment V&V and Test Planning Activity for O&M

IEEE 1012 [1] defines integrity levels and addresses how different levels impact V&V efforts. If integrity levels were used during initial development, they may evolve and change during the O&M phase. If integrity levels were not used, they can be introduced during the O&M phase. Refer to IEEE 1012 if you are interested in using integrity levels to manage O&M-phase software activities.

Next, ISO 29119 [2] is a risk-based testing standard made up of five parts, which can be used during O&M. The ISO 29119 set of standards defines process activities, documentation and techniques. New risk assessments may be needed during the O&M phase. Information about software qualities can change, introducing or reducing risk. If risk-based testing was not used, the O&M phase can be a good time to introduce it, allowing V&V and testing to focus on the software's higher-risk qualities and areas. In risk-based testing, not all areas and qualities of software will need the same levels of activities. Risk-based testing can be used to manage O&M V&V and testing costs and scheduling.

Updated risk areas that may change during O&M include:

–**Security and privacy threats.** Not many years ago, concerns about privacy were minimal for many software O&M efforts. Almost daily now, we hear about software systems being hacked, which costs millions of dollars and can destroy a company's integrity. Thus, software that was once viewed as safe from a security perspective can become a high risk.

–**Safety impacts to humans from new use of software systems.** Fitness watches were designed to help with health and exercise, but they are now being used to make medical decisions [8]. Thus, fitness watches have become more of a safety risk than developers might have originally planned.

–**Hazards introduced by system changes.** A programmable logic control (PLC) for power systems was installed in an isolated network, but then a router was added to the network to

allow communications. Now the PLC is threatened by hacking from a virus, such as STUXNET [9]. Electric utilities are now at risk due to just such a scenario.

–**Performance needs growth.** Initially, web systems were focused on functions and design. Now most websites worry about crashing under high volumes as they become popular. This changes the V&V and testing focus from functional testing to performance analysis and V&V.

–**Environment and hardware changes.** An existing vendor provided a new computer system and supporting software to a project, but the vendor's team did not understand the new environment the software or system was to be used in. Project management thought their software was proven because the vendors were an existing third party. That is, until failures started happening during V&V and testing.

These examples are risk areas that went from “low risk” during initial development to “high risk” in O&M phases.

In integrity and risk assessment during O&M, ask what is changing. Changes can be in the software, hardware, users, threats and environment. Be cautious during O&M V&V and testing activities.

Regression Testing Activities Important for O&M

Most software V&V teams are familiar with regression testing [10]. This testing helps you discover if you broke something that was already working during development – which happens as the software changes in areas that were already tested. Regression testing happens in software partially because of coupling [11] or lack of team understanding of design.

Teams develop different strategies for regression testing, including:

- Rerunning all testing.
- Rerunning tests in the code areas related by function.
- Rerunning a standard regression test suite.
- Rerunning what the team thinks needs to be run.

Each of these strategies has positive and negative considerations [7]. I will not define a complete regression test strategy and approach, but there is debate in the test community about the validity of regression testing [12].

The validity question comes from the pesticide paradox [13]. In the pesticide paradox, the ability of a test case to provide information diminishes the more times it is run without changes. Running regression tests over and over is therefore of questionable benefit, but most organizations employ some level of regression testing and analysis. The debate about regression tests will go on.

I ran a test with a mix of regression items 2, 3 and 4 on a personal progress and did risk-based testing planning. I changed “old” tests to some extent to avoid aspects of the pesticide paradox, while still addressing regression. I made sure a test existed that could assess the error or changes in the software. During O&M phases, I did a lot more testing because I employed V&V and test automation (see automation section below).

During one O&M cycle, I had the following percentage allocations:

- Standard regression tests: 20 percent.
- Expert team-selected regression tests: 30 percent.
- New tests (including developer tests): 30 percent.
- Tests traced to coupling that needed to be executed: 10%.
- Historic system tests not run over a long period of time: 10%.

These percentages are nothing magical. In fact, on other O&M cycles the percentages looked different. The key was that I applied critical thinking and planned tests early in the O&M cycle. The stakeholders accepted the V&V and test plans, and I modified test plans as needed during the cycle.

You may notice the last category percentage seems new. The team added this, and it was different than standard regression testing concepts. I learned over years of O&M that it was a good idea to pull out historic tests that had not run for a while and then rerun them with changes for the new O&M cycle. I did this during most O&M cycles, and slowly over the years (it was a long-term project), the team would cycle through the majority of historic system tests.

We were looking for a regression test that had escaped previous test cycles. In at least one case, we found an unexpected regression error using this method. If I'd had the time and high levels of automation, I could have avoided this last step. However, I did not have the extra time and budget, so cycling over the composite test suite was a compromise.

Introducing New V&V and Testing in O&M

In the O&M phase, many projects will add new functionality to the existing software. This in effect resets the testing efforts back to where they were during development, so new tests are needed. This makes sense to teams and managers. However, there are a variety of overlooked sources where new tests may become necessary during O&M, including:

- New operational concepts for the software system, where the existing tests do not address the use cases.
- Changed usage environments, where software may encounter external situations not tested during development.
- Updates to hardware, where the new hardware behaves differently than what was originally tested.
- Changes to incoming or output data, where data boundaries and classes have not been tested.

In new testing, different test plans, risks, designs and implementations may become necessary. Risk-based testing should be redone, and the team may want to consider new exploratory tests [7] in addition to the planned tests.

The Need in O&M V&V and Testing to Address Third-Party Software Changes

An interesting aspect of O&M V&V and testing is third-party software and hardware. These components can be commercial-off-the-shelf (COTS) or customized software and hardware. It is common to have such third-party elements change and be updated during the O&M phase. While it may be tempting to think that the third-party vendor owns the V&V and testing of such elements, this can be dangerous for the product.

On one project, I had some vendor hardware components. One of them wore out and was replaced. The configuration management numbers were the same, but in our test lab, I started having test failures during our regression tests. At first, I thought it was a software issue. After much analysis V&V, the team found the vendor had changed a transistor in the hardware!

They should have rolled the configuration part number, but they did not because the part was considered to be equivalent (which

it was not). If I had not been running regression tests in the lab with all the updated elements, including the third-party elements, I might have fielded a bad release during the O&M cycle.

In third-party testing, consider running new and regression tests. Having a good configuration control system for hardware and software is a good start; however, the team must still “trust but verify.” The development project owns the software system, which includes all of the third-party elements.

V&V and Test Automation

Much of the IT industry loves the idea of V&V and test automation, as if it is a panacea that will solve many problems (e.g., schedule, budget, testing completion and regression). Automation has a mixed track record. Sometimes it works, saves time or money, and is a good idea. At other times, automation efforts have failed, because it takes more initial setup time (learning curves) and money (tooling and consulting fees) up front during development to make automation work. Often the payback is only realized after several test and regression cycles.

My experience is that V&V and test automation's real payback can occur during O&M. If I must do new and regression tests manually, my labor costs remain high during each O&M cycle. If I have automation, rerunning regression suites and modified new tests can be very cost effective. I can spend the savings on new and exploratory tests. There will be some O&M costs to maintain automation tests and tooling, but these can be reasonably managed.

The automation payback gets better as a team moves into the cloud, where they can add test execution resources quickly. Automation efforts will still include the costs of maintaining test automation, running new tests and thinking critically, but automation during O&M efforts becomes quite attractive.

Of course, automation must be planned, and it should be started during initial development. My experience is that it takes three to five O&M regression V&V and testing cycles to start to see a payback on automation. However, if the team factors in the costs of automation and the savings during O&M and development, automation becomes a viable path.

Summary

I have outlined V&V and testing impacts to O&M with a focus on software, but you can never forget the hardware and system in test planning. Many of my students wanted to get jobs doing new development because they thought it was more fun. However, I spent most of my career doing O&M V&V and testing. Most of us will spend much more time working in O&M than in new development. O&M comes with its own V&V and testing challenges, some of which I have outlined in this paper.

For more information, see the references section below. O&M V&V and testing is a big subject, and a team must address bugs, regression, new testing, third-party vendors and automation during O&M. V&V and testing assessment includes checking for software degradation during evolution and looking for errors that may still exist in software even after years of use. You should now have a starting point for O&M V&V and testing budgeting, scheduling and detailed planning.

REFERENCES

1. 1012 – IEEE Standard 1012–2012. “IEEE Standard for Software Verification and Validation”™
2. 29119 – IEEE Standard on Software Testing – Five parts covers all of software testing (published September 2013)
3. Beizer, Boris. (1995.) “Black Box Testing: Techniques for Functional Testing of Software and Systems.” John Wiley & Sons Inc., USA.
4. Myers, Glenford. (1979.) “The Art of Software Testing.” Wiley.
5. Kaner, C.; Bach, J. & Pettichord, B. (2002.) “Lessons Learned in Software Testing.” John Wiley & Sons.
6. Craig, R. & Jaskiel, S. (2002.) “Systematic Software Testing.” Artech House Publishers.
7. Hagar. (2013.) “Software Test Attacks To Break Mobile and Embedded Devices.” CRC press.
8. <http://www.mirror.co.uk/news/uk-news/teens-life-saved-fitbit-after-7154991>
9. <https://en.wikipedia.org/wiki/Stuxnet>
10. https://en.wikipedia.org/wiki/Regression_testing
11. [https://en.wikipedia.org/wiki/Coupling_\(computer_programming\)](https://en.wikipedia.org/wiki/Coupling_(computer_programming))
12. http://kaner.com/pdfs/gui_regression_automation.pdf
13. https://en.wikipedia.org/wiki/Paradox_of_the_pesticides

ABOUT THE AUTHOR

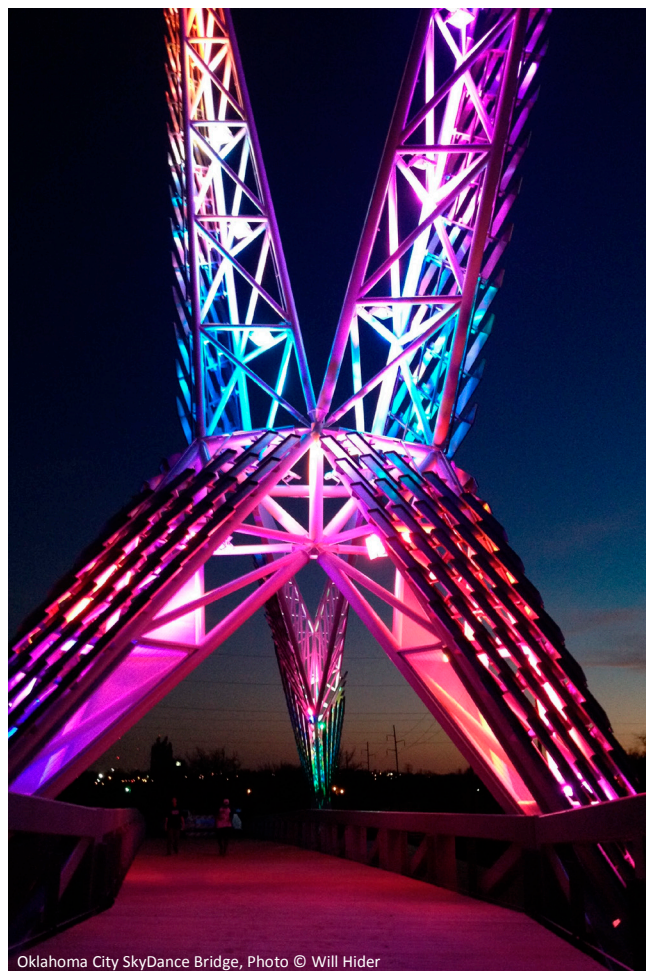


Jon Hagar is a systems-software tester consultant at Grand Software Testing. He supports software product integrity, verification and validation, specializing in embedded, IoT and mobile software systems.

Jon has worked in software engineering, particularly testing, for four decades. Embedded projects he has supported include control system (avionics and auto), spacecraft, mobile-smart devices and ground systems (IT). He also worked on attack-based testing in the mobile/IoT space.

Jon wrote “Software Test Attacks to Break Mobile and Embedded Devices” and has published an IoT testing e-book and more than 50 papers. He is the lead editor and author on international standards (ISO 29119), co-chair of the OMG UML test profile standard, and a contributor to IEEE 1012 V&V plans.

Jon.d.hagar@gmail.com



Oklahoma City SkyDance Bridge, Photo © Will Hider

WE ARE HIRING

ELECTRICAL ENGINEERS AND COMPUTER SCIENTISTS

As the largest engineering organization on Tinker Air Force Base, the 76th Software Maintenance Group provides software, hardware, and engineering support solutions on a variety of Air Force platforms and weapon systems. Join our growing team of engineers and scientists!

BENEFITS INCLUDE:

- Job security
- Potential for career growth
- Paid leave including federal holidays
- Competitive health care plans
- Matching retirement fund (401K)
- Life insurance plans
- Tuition assistance
- Paid time for fitness activities

Tinker AFB is only 15 minutes away from downtown OKC, home of the OKC Thunder, and a wide array of dining, shopping, historical, and cultural attractions.



Send resumes to:
76SMXG.Tinker.Careers@us.af.mil
US citizenship required

Software Estimates That Work

Louis M. Taff

Summary. We lay out a path for initial estimating of large new projects for systems and software development organizations that have experienced resource problems. The paper is targeted mainly at organizations not yet underway with efforts to improve estimating (following, e.g., the Software Engineering Institute's Capability Maturity Model); however, those already working on process improvements may find suggestions as well.

We propose a three-pronged approach. The first is to implement a standard estimation *process*, and the second is to deeply involve development staff — those who do the work — and other stakeholders in this process. The process itself will generally be unique to each organization and its particular culture, projects, circumstances, people, etc. A reasonably formalized process offers benefits such as better estimates, more consistent estimates, accountability and manageability of the estimate process, and a baseline for continuous incremental improvement of estimates.

Depending on the process, additional benefits may be derived, such as better technical designs and better relationships between staff. Better estimates in turn will lead to project-staffing profiles that avoid “back-end loading” projects and will yield higher software quality. Involvement of the staff should build “buy-in” to the estimates and improve the probability of meeting them. We have validated this approach on a large project (hundreds of staff-years) in a traditional waterfall development environment. It may not be directly applicable to more recently evolved Agile processes because their requirements may not be known sufficiently as they begin. Finally, the third prong is to put in place a method to measure the effort expended on projects in sufficient detail to capture historical data for future estimates.

Background

The software community has been notorious for its poor record in estimating the costs of system development. The results of poor estimates are well known — cost and schedule overruns, canceled projects, poor morale and poor software quality because of the rush to finish on time — and ultimately due to insufficient early staffing because of an estimate that was too low.

We attempt here to make the case that estimating system-development costs can be improved by formalizing estimation into a *process* — a planned sequence of inputs, work activity and outputs. It is difficult to know how many organizations have a defined process for estimating. A member of an organization with which we consulted stated before it defined a process: “Today there is no single method employed for estimating projects' resources, and the methods currently employed are somewhat informal and vary from estimator to estimator and project to project. These estimates are not based on a well-defined estimation process and a quantification of software size such as lines of code. It is common knowledge that we do not know where these estimates come from and how they are derived. As a result, the process of estimation is highly subjective and difficult to measure and improve. More critically, we do not deploy any feedback mechanism to check actual project spending against our estimates. Therefore, we do not learn from the past.”

We feel that this unnamed organization is not unique. Vigder and Kark [2], in a Canadian study with a military focus, found “definite patterns in which projects are procured, costed, and executed. These patterns show that

- Software cost estimation is done too early in the procurement process and is based on — usually — wrong specifications.
- Once established, the estimates are very difficult to change.
- Parametric models for software cost estimation are rarely used within either the military or the industry developing systems for the military.
- Business-oriented software producers are more likely to use some parametric models.
- There are no historical data on which to base the estimates for the new software projects.
- Very often the price for the software development ... does not reflect the actual estimates for the software as a function of the requirements, but is a function of many other non-technical factors.”

In addition to the ill-defined methods that produce such estimates, there are further drawbacks to not having a process. The estimates are not well documented, and it's not clear what is included in them and what, if any, assumptions were made. The time and effort used to make the estimates was probably “borrowed” from other work, which may be delayed because of it, and the time might not be accounted for anywhere. If the estimates are needed by some deadline (i.e., if there is a schedule), it may not be easily known whether the deadline will be met; the progress in coming up with the estimates is hard to track.

Why Make Estimates? The Business Case

It's useful to consider why we make estimates in the first

place. For any project, software or otherwise, the criterion for deciding to go forward is some kind of business case — it is perceived that the value of the project's results exceeds its costs. This is true whether we are estimating one of many features for a new release, an entire software project, a bid on a job, or simply adding a new room to our house.

The costs in a business case may include many items in addition to software development. Depending on whether the developed system is to be sold or licensed or used only internally, there may be costs for requirements development, third-party software acquisition, marketing, sales, deployment, customer service, operation, data entry, maintenance, etc. And there will be trade-offs — extra attention and costs in human-interface development may reduce larger expenses later in operation and data entry. Thoughtfully designed screens, higher performance or more functions may increase development costs but yield more sales. This last notion illustrates that inherent in this business case is an estimate of the benefits as well as the costs. Notwithstanding, for large software projects, development is usually the dominant cost.

To estimate all costs, we need to know what the product will do after it's built — we need to know its requirements in sufficient detail to estimate the project's costs and its benefits as well. Some level of requirements is one of the major inputs for estimation. As an aside, in our experience we have not seen as much attention given to estimation and its failures on the benefits side of the business case as on the costs side.

Staffing

A second major purpose of estimates is to set the levels of staffing and other resources for the project once the decision is made to go forward. We have pointed out [1] that this initial staffing may have a strong influence on the quality of the product. For if the resources are underestimated, the early staffing profile will be too low — and the architectural and design phases will not receive the attention appropriate for the project's true size. As a result, the project will be “back-end loaded” to meet its schedule, and quality will suffer in a rush to finish on time.

Estimates and Buy-In

Thus, initial estimates are used to set the budgets and resources of managers and staff. If the estimates were made with methods that appear to these people to be arbitrary, can they be expected to “buy in” to them and strive to meet them? We feel that they cannot. In our experience, to gain buy-in they or someone they trust must participate in the estimate process. Some staff may have experienced an underestimated project with an end game of rushing, panic, and low-quality results. These people may mistrust any estimates and staffing profiles, overestimating if asked and generally poisoning the project atmosphere. Therefore, *a primary goal of the estimate process needs to be to gain the buy-in — agreement, acceptance, ownership and commitment — of the managers and staff who will commit to the content and schedule and do the work.*

Although not realistic in most projects, in an ideal situation the same individual makes and commits to an estimate, then personally executes the project and benefits (or suffers) from

meeting the commitment (or not). This is a competitive bidding model. The same individual feels pressured to bid both low (to get the work) and high (to profit).

Problem Statement

In summary, most organizations need a method to generate software estimates that:

- Has buy-in from all stakeholders.
- Is consistent from project to project.
- Is based on the same documented requirements that are used for other purposes in the business case.
- Produces well-documented estimates.
- Includes all assumptions.
- Can be tracked and managed.
- Improves with experience.

What's Needed Is a Process What Is a Process?

We feel that a big step toward meeting this need is establishing a formal process for generating estimates. We can define a process as a specified sequence of work activities and/or processes with inputs and outputs. This recursive definition allows processes to incorporate subprocesses. The availability of inputs and the production of (presumably objective) outputs gives us milestones with which to measure progress and manage — one of the benefits of a defined process. A formal process gives visibility to the effort needed for estimation — it can be budgeted, scheduled and tracked by project managers. If the process is written down and followed it should be repeatable over time. If, in addition, a method is in place for measuring the costs of the development process, the groundwork is laid for incremental improvement of the accuracy of the estimates. Finally, once a process is established, it can be augmented to produce additional outputs. For example, many feel that some measure of risk is useful to management.

Project Size

The principles we lay out scale up to large efforts (2,000–4,000 staff-years), as evidenced by our “Estimating” process [1].

Expectations of Cost and Accuracy

We wrote above that staff need time to do estimation. We have noted [1] that estimates with an accuracy of ± 20 percent may cost up to 10 percent of the total effort of the entire project. This is a significant cost, and it should prove to be a useful barrier in filtering out requests for estimates that are not serious. In our Estimating process [1], this cost buys much more than just an estimate — it gives a high-level design, a breakdown of the cost into component parts, exposure of the prospective project across the entire development organization, and more. Similar benefits will apply to any process designed to give this level of accuracy.

Alternatively, lower accuracy is presumably achievable at lower cost. The appropriate trade-off between estimate cost and accuracy generally depends on the circumstances of each project and organization, the project size, the consequences (i.e., the risk of under- or overestimation), etc. If multiple projects are underway, underestimation might be alleviated by

borrowing resources from other projects. Clearly, however, if the project will consume all the resources of the organization, underestimation is extremely serious.

How We Would Go About It Build Consensus

If we were to create a process for estimating, we might first find everyone who wants the estimates. We'd need to discuss with each of these people what the estimates are to be used for, what their inputs are (what the estimates are to be based on), what their expectations are for accuracy, other outputs desired from the estimating process, straw timelines for both the estimates and the project itself, and how the cost of making the estimates is to be funded. We'd then need to talk through a proposal as to how the process would work in our particular environment, with our unique culture and business and development processes. This would include finding someone to *own* the process — that is, someone to make the estimates happen. It also includes identifying all stakeholders, especially those who are going to make the estimates and those who will be expected to commit to them. At some point the support of upper management needs to be secured. Depending on the environment, this can be earlier (to help get everyone's buy-in) or later (once a realistic straw plan is formulated).

The estimating process we describe in reference [1] included people from marketing (customer needs), product management (business case), systems engineering (requirements),

architecture and design/development — all the stakeholders of the business. We advise involving people from areas not always included in the early stages, such as documentation, project management, product support/maintenance and (if not included in design/development) system test/verification. Their roles are to ensure a common understanding of requirements and to estimate if necessary. If the product is to be deployed within the enterprise, prospective users (i.e., customers) should be included as well to verify that the requirements meet their needs. Various stakeholder roles within the estimation process are different from each other, however. For example, the business side is not involved to “negotiate” an estimate but to ensure that what is estimated fulfills business and marketing needs. If during the process it becomes clear that the cost will be too high, these people must have the authority to cut back the project scope to get an affordable estimate while ensuring that the business case remains positive.

List the Deliverables and Their Owners

Some possible milestones and deliverables associated with an estimating process are compiled in Table 1. First, we need to identify estimators trusted by the development staff and ask them what inputs they need. If these are not all available, then as part of the process, staff and budget need to be found to produce them from what is already available. This may include considerable work, such as producing a more detailed specification proposal and/or a design proposal. We term these documents “proposals” and suggest that they shouldn't be binding on those who later implement the project, particularly if they are not the people who produced them. The design proposal can function as an existence proof, bounding the cost.

We suggest that the process output, the estimate, by itself, *not* be considered a commitment. Commitments are made by those with the authority to decide how to use resources and usually with influence on the reward system. Therefore, we have included a separate commitment milestone with management owners in Table 1.

How Are Estimates Made?

Vigder and Kark [2] note that “the bulk of the current literature and research on cost estimation is devoted to formal models, particularly as relates to new system development.” However, they “found that formal models are *not* in general used by estimators as a primary tool for cost estimating. By an overwhelming majority, informal analogy was the most commonly used estimating method for all types of software and for all organizations. Estimators used their past projects as a basis for estimating the cost of future projects.” A similar conclusion was reached by Hihn and Habib-Agahi [4]. Indeed, this method is the basis of our Estimeeting process[1].

Analytical methods such as function points, noncommentary source lines (NCSL), and other objective measures require training in their use and may require more detailed requirements than are available when the estimates are needed. These methods also need to gain the confidence of the managers and workers expected to commit to them. If the conditions of training, detailed requirements, and confidence exist, analytical methods may be an important ingredient in the accuracy of and buy-in to a formal process.

Possible Milestone	Possible Owner
Estimate process owner identified	Upper management
Process funding established	Management
Market requirements	Marketing and/or customer(s)
Technical requirements proposal	Systems Engineering
Allowable interval (schedule) of project	Marketing, product and project management
Identify available staff	Development management
Inventory of staff experience level	Development management
Identify estimators trusted by staff	Development management
Compile estimators' needs.	Estimate process owner
Survey of size/complexity of existing software base, if any	Development management
List of special hardware available (e.g., labs, communication facilities, etc.)	Development management
High-level architecture/design proposal	Architects/designers
Work items/work-breakdown structure enumerated.	Designers and/or Estimators
Assumptions listed.	Estimators
Open issues to be resolved.	Estimators
Technical staff weeks, e.g., by specialty (coders, testers, documenters, etc.) with a measure of risk – e.g., min/max/most probable – for each estimate.	Estimators
Equipment needed (testing labs, instruments, etc.)	Estimators
Achievable end date with reasonable built-in contingency and assumptions about staffing.	Estimators or managers
Commitment and sign-off to the (possibly adjusted) estimate by responsible manager(s)	Managers

Table 1: Possible Milestones for Estimating Process

Forms and Checklists

We strongly recommend what we feel is an important additional task, namely the production of appropriate forms and checklists to record and guide the process. This captures the knowledge of the enterprise, incorporating the issues and tasks that have proven important in the past and helping ensure that nothing is forgotten. We like to quote Donaldson [3]:

“...provided you are reasonably systematic it is not difficult to estimate the duration of an activity, the problem is to identify the right activities.”

Forms and checklists serve additional purposes as well in that they *document* the estimates without anyone needing to write a report. If the appropriate reference source information is on the forms, the estimates become traceable as well. Forms and checklists lend themselves to process improvement also, because new knowledge can be added to the forms easily for future estimation. The particular content of the forms and checklists depends on the estimation process.

What's in the Estimates?

In addition to focused development, staff members typically engage in myriad activities not directly related to producing software. They may variously participate in training, traveling, interviewing job candidates, consulting and reviewing documents on other projects, general meetings, team building, making phone calls, answering email, fighting project and nonproject fires, etc. These activities often interrupt project time and incur get-back-on-track costs. The question is how time spent on them is accounted for. Ideally, most such activities would be listed individually in timekeeping records, but this is unrealistic because the durations spent on them happen in small pieces, because no one can track at that granularity level, and because it's unlikely that anyone really cares. Management must tell project staff how to treat these intervals. Staff might be asked to estimate the total time spent this way each time card interval and include it in a catch-all category. If, alternatively, they are told to basically ignore it, estimators need to adjust their estimates for it based on their experience and on measurements of previous projects.

Example: The Estimeeting Process

We have previously described a process in the telecom field [1] built around estimating meetings we call “Estimeetings.” This process is based on breaking down the work architecturally into impacts on a number of existing subsystems. Estimators representing the subsystems estimate work in their own areas of expertise. This process is particularly suited to multiple releases of a product with continuity of knowledgeable people.

Notes on Measurements

If our process is written down, it will be repeatable and will lend itself to incremental improvement — doing better next time than

Form	Filled Out By	Given To	Reason
Estimate request	Product manager, marketing	Estimate coordinator	Asks for estimate, authorizes funding for it
Function or subsystem estimate	Estimator	Estimate coordinator	Lists work items, needs, assumptions, issues and estimate(s) for function/subsystem.
Estimate summary	Estimate coordinator	Development manager	Adds function/subsystem estimates and gives best technical estimate
Committed estimate	Development manager	Product manager, marketing	Gives committed estimate.

Table 2: Possible Forms in an Estimating Process

last time. Improvement implies knowing how well we did — it implies *measurement*. We can thus distinguish between the benefits of simply having an estimation process — and these benefits are real — and having a measurement method. That is, estimation and measurement are separate and have distinct benefits. A measurement method may be even more worthwhile than an estimation process. If accurate, the benefits of measurement include:

- The determination of the real total costs of the project. Knowing how accurate the originate estimates turned out is a key input in estimating future projects.
- The determination of the costs of each subprocess or function of the development process, ideally by stage (design, unit testing, integration, etc.).
- Applied to the estimation process, costs of the estimates themselves can be measured.

A method for measurement of costs is not within the scope of this paper, but we do have a few remarks on the subject. The most straightforward way to record cost data is to use the work time reporting mechanism (time cards). Time reporting categories and subcategories would reflect the activities and tasks estimated.

However, there may be issues with the use of time cards for measurements. First, the organization's budgeting or funding mechanism may not encourage straightforward time reporting. This can happen if time-reporting categories cannot easily record the plethora of different activities estimated separately. Further, if a project runs over budget, staff may be asked to charge their time to a different project. Depending on the environment, as a financial practice this can range from “doesn't matter” to being illegal. But for measurement purposes, it is unacceptable. Second, there may be issues with reporting overtime. Some staff may not be authorized for overtime but work the hours anyway, not reporting them on their time cards (see below). This practice will lead to continued underestimation.

More subtle personnel issues may play a role as well. For example, a staff member who, based on his or her estimate, has made a commitment to “finish the module by month's end” may be working evenings and weekends to get the job done, and may not report the unpaid overtime because of pride or embarrassment at having underestimated the job. Some staff may fear being penalized for spending too little or too much time on some special function. Or there may be a macho culture of status for those who consistently work late. These issues illustrate the difficulty of accurate measurement, important though it is. They also illustrate an ethical bind on management — the conflicting desires 1) to meet or exceed budget

goals (and increase competitiveness in winning future projects) and 2) to fairly compensate staff who might not be reporting all their time (and prevent burnout and departures).

Perhaps a way to avoid some of these concerns is to use an anonymous system separate from time cards. The system needs to track whether individuals have made their weekly entries but would not associate their inputs with them. The measuring method should measure every activity that was estimated. If significant activities took place that were not estimated, these need to be measured and added to the estimation process – for example, to the forms and checklists. Some mechanism needs to be included to make this easy. Many commercial products exist for time reporting, comparison with estimates, integration with project management software, etc.

Function/Subsystem Estimate Form					
Product					
Feature estimated					
Function/Subsystem					
Target Release					
Estimate Date					
Estimate Based On					
Estimator					
#	Work Item	Min	Avg	Max	Note
1					
2					
...					
#	Notes / Issues / Assumptions / Special Equipment or Labs, Etc.				

Table 3: Simple Function/Subsystem Estimation Form. Possible mandatory/optional breakdown is not included. The minimum/average/maximum ranges are intended to provide a measure of risk. Example functions or subsystems might be screens/human interface, database, communications, math library, etc. A large project will require many of these forms.

Estimate Summary Form			
Product			
Feature estimated			
Target Release			
Estimate Date			
Estimate Coordinator			
Function / Subsystem	Min	Avg	Max
Total estimate			
Notes			

Table 4: Possible Summary Form for Assembling an Estimate. For changing an existing product, the list of functional areas or subsystems can be hard-coded into the form to ensure none is forgotten.

Conclusion

We have built a case that making estimates for significant software developments is best accomplished through a formalized process of estimating that involves all stakeholders. We cannot give a one-size-fits-all approach, but we feel that a repeatable process with input from all the stakeholders and documented outputs will go far in relieving confusion and misunderstandings, will focus attention on the importance of estimation and will achieve support from the people who must execute the plan. We further advise that a method be established for measuring development costs, though we find it difficult to give detailed guidance on how to achieve this.

Acknowledgement

It is a pleasure to acknowledge the contribution of the ideas, insights and perspectives of my Estimeetings co-author Jim Borchering. It will be clear to readers of reference [1] that I have borrowed heavily from it in constructing this paper.

ABOUT THE AUTHOR



Lou Taff created the first structured process for estimating software development for a large switching project at AT&T Bell Laboratories, where he was Distinguished Member of Technical Staff. At AT&T he planned and engineered new services for the AT&T network and held the supervisory rank of technology consultant. Subsequently with Ericsson, he worked on wireless systems and platforms. Taff holds an S.B. in physics from MIT and a Ph.D. in nuclear physics from Iowa State University; he initially worked in physics at the University of Groningen in the Netherlands and in the computing department at Fermilab. He is a co-inventor of four patents and has authored over 20 published articles and many proprietary reports. He now maintains a consulting practice in which he assists clients in implementing software estimating processes.

taff@softwareestimatingprocesses.com

REFERENCES

1. Taff, L. M.; Borchering, J. W. & Hudgins, W. R. Jr. (August 1991.) "Estimeetings: Development Estimates and a Front-End Process for a Large Project." IEEE Transactions on Software Engineering 17, No. 8. 839-849.
2. Vigder, M. R. & Kark, A. W. (February 1994.) "Software Cost Estimation and Control." National Research Council Canada. Institute for Information Technology, Rpt. No. 37116. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.9301>.
3. Donaldson, H. (1978.) "A Guide to the Successful Management of Computer Projects." London: Assoc. Business Press.
4. Hihn, J. & Habib-Agahi, H. (1991.) "Cost estimation of software intensive projects: a survey of current practices." 13th International Conference on Software Engineering. IEEE Comput. Soc. Press. Los Alamitos, California. 276-87.



Principles of Software Testing

Theresa Lynn Sanderfer Brown, Lynn Brown Software Consulting

Abstract. There is a myth that if we were really good at programming, there would be no bugs to catch. This is not true. In this paper, I will discuss fundamentals of software testing that expel some of these myths. I will then discuss some specific software testing techniques.

What are the principles of software testing?

Once source code has been generated, software must be tested to uncover and correct as many errors as possible before it is delivered to the customer. The goal is to design a series of test cases with a high likelihood of finding errors.

There are techniques that provide systematic guidance for designing tests that exercise the internal logic and interfaces of every software component. They also exercise the program's input and output domains to uncover errors in program function, behavior and performance. The program must be executed several times to find and remove all errors before it gets to the customer.

In order to find the highest possible number of errors, software tests must be conducted systematically, not randomly. The following principles should guide every software test program:

1. A test case must include:
 - A description of the input data to the program.
 - A precise description of the correct output of the program for that set of input data.
2. Don't allow programmers to test their own programs.
3. Don't allow programming organizations to test their own programs.
4. Thoroughly inspect the results of each test.
5. Write test cases for invalid and unexpected and valid and expected input conditions.

Examining a program to see if it doesn't do what it's supposed to do is only half the battle.

1. Avoid throwaway test cases unless the program is truly a throwaway program.
2. Don't assume no errors will be found when you plan a testing effort.
3. If you found a number of errors already in a section, there are probably more errors.
4. Remember that testing is an extremely creative and intellectually challenging task.

For conventional applications, software is tested from two different perspectives: "white box" (for internal program logic) and "black box" (for software requirements). When you begin testing, you should try hard to break the software. Design disciplined test cases and review them thoroughly. You can also evaluate test coverage and track error detection activities.

In his book "Software Engineering: A Practitioner's Approach," Roger S. Pressman lists the following attributes of a good software test:

1. "A good test has a high probability of finding errors." To achieve this goal, the tester must understand the software and develop a mental picture of how the software might fail. To do this, chart the flow of the function and step through each box, considering the potential outcomes. Use process failure mode and effect analysis for a given function (also called a value stream). At each major step, consider:
 - The components and subcomponents.
 - The component's functionality.
 - What could go wrong with the component.
 - What it would take to prevent the negative action. This same principle is very applicable to testing software.
2. "A good test is not redundant." Testing time and resources are limited. There is no point in conducting a test with the same purpose over and over.
3. "A good test should be 'best of breed.'" In a group of tests with similar intents, time and resource limitations may allow you to execute only a subset of these tests. In such cases, you should use the test with the highest likelihood of uncovering a whole class of errors.
4. "A good test should be neither too simple nor too complex." Although it is sometimes possible to combine a series of tests into one test case, the possible side effects may actually hide errors. In general, each test should be executed separately but thoroughly.

White Box Testing

As previously mentioned, one type of testing is white box testing. This allows you to examine the internal structure of the program. White box testing derives test data by examining the program's logic. The goal at this point is exhaustive input testing, which causes every statement in the program to execute at least once. If one test case exhausts all possible paths of flow through the program, it is possible you have completely tested the program.

Though you can test every path in a program, the program might still be loaded with errors. There are three explanations for this:

1. An exhaustive path test does not guarantee that a program matches its specification. For example, if you were asked to write an ascending-order sorting routine but mistakenly produced a descending-order sorting routine, exhaustive path testing would be of little value. Although the program ran fine, it would still have one bug. It would be the wrong program, as it did not meet the specifications.

2. A program may be incorrect because of missing paths. Exhaustive path testing, of course, would not detect the absence of necessary paths.

3. An exhaustive path test might not uncover data-sensitivity errors. Suppose you have to compare two numbers for convergence, seeing if the difference between the two numbers is less than some predetermined value. Leaving off the absolute value sign in this equation would cause a different result. Unfortunately, you may not detect that error just by executing every path through the program.

Black Box Testing

In black box testing, you focus on finding circumstances in which the program does not behave according to its specifications. Test data is derived solely from the specifications, not from the program's internal structure. If you use this approach to find all errors in the program, the criterion is once again exhaustive input. In black box testing, you should test all *valid* inputs and all *possible* inputs.

Equivalence Partitioning

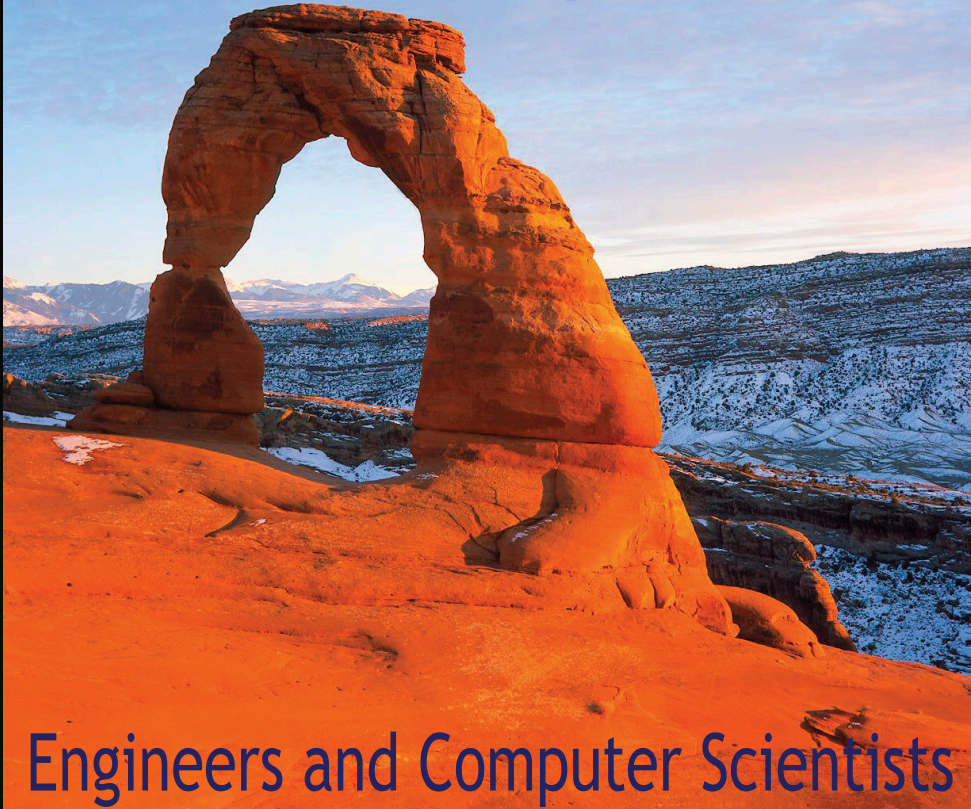
Equivalence partitioning is a black-box testing method that divides the input domain of a program into blocks of data from which the test case can be derived. An ideal test case single-handedly uncovers multiple errors. Equivalence partitioning strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed. Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions.

Boundary Value

In boundary value analysis, you select test cases at the "edges" of the class. Boundary conditions are directly on, above and beneath the edges of input equivalence classes and output equivalence classes. Boundary-value analysis differs from equivalence partitioning in two ways:

1. Rather than selecting any element in an equivalence class as representative, boundary-value analysis requires that one or more elements be selected so that each edge of the equivalence class is the subject of a test.
2. Boundary value test cases are derived by considering the result space (output equivalence classes), rather than just focusing attention on the input conditions (input space).

Hiring Expertise



Engineers and Computer Scientists

The Software Maintenance Group at Hill Air Force Base is recruiting **civilians** (U.S. Citizenship Required). Benefits include paid vacation, health care plans, matching retirement fund, tuition assistance, paid time for fitness activities, and workforce stability with 150 positions added each year over the last 5 years.

Become part of the best and brightest!

Hill Air Force Base is located close to the Wasatch and Uinta mountains with skiing, hiking, biking, boating, golfing, and many other recreational activities just a few minutes away.



Send resumes to:

309SMXG.Recruiting@us.af.mil
or call (801) 777-9828



www.facebook.com/
309SoftwareMaintenanceGroup

Zero Crossing Testing

Zero-value testing includes the input values of zero, zero crossing, approaching zero from either direction or similar values for trigonometric functions. The fundamental principle behind zero crossing tests is zero crossing, the point when the result of the function will change signs (either from positive to negative or from negative to positive). When this happens, unexpected results occur.

Orthogonal Array Testing

There are many applications in which the input domain is relatively limited. That is, the number of input parameters is small and the values each of the parameters may take are clearly bounded. When these numbers are very small (e.g., three input parameters taking on three discrete values each), it is possible to consider every input permutation and exhaustively test the input domain processing.

However, as the number of input values grows and the number of discrete values for each data item increases, exhaustive testing becomes impractical or impossible. Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.

For example, consider a system that has three input items, X, Y and Z. Each of these input items has three discrete values associated with it. There are $3^3 = 27$ possible test cases. The orthogonal array testing method is particularly useful in finding errors associated with regional faults rather than all possible faults. Orthogonal testing is also beneficial at finding error categories associated with faulty logic within a software component.

Fault-Based Testing

Fault-based testing occurs when the tester intentionally injects faults and executes tests to ensure the product or software will correctly stop at erroneous information.

Testing for Real-Time Systems

The time-dependent, asynchronous nature of many real-time applications adds a new and potentially difficult element to the testing mix: time. The test case designer has to consider conventional test cases, event handling (i.e., interrupt processing), the data timing, and the parallelism of the tasks (processes) that handle the data.

In many situations, test data provided when a real-time system is in one state will result in proper processing, while the same data provided when the system is in a different state may lead to error. Also, the relationship between real-time software and its hardware environment can also cause testing problems. Software tests must consider the impact of hardware faults on software processing. Such faults can be extremely difficult to simulate realistically.

Task Testing: The first step in testing real-time software is testing each task independently. Conventional tests are designed and executed for each task.

Behavioral Testing: Using system models created with automated tools, it is possible to simulate the behavior of a real-time system and examine its behavior as a consequence of external events. These analysis activities can serve as the design basis for test cases conducted when the real-time software has been built.

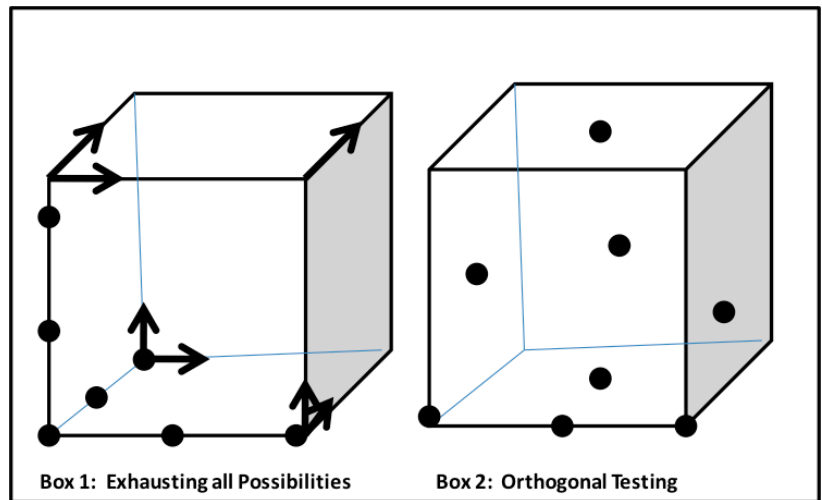


Figure 1: Orthogonal Testing Representation

Intertask Testing: Once errors in individual tasks and system behavior have been isolated, testing shifts to time-related errors. Asynchronous tasks known to communicate with one another are tested with different data rates and processing loads to determine if intertask synchronization errors will occur. Tasks that communicate via a message queue or data store should be tested to uncover errors in the sizing of these data storage areas as well.

System Testing: Software and hardware are integrated, and a full range of system tests is conducted in an attempt to uncover errors at the software/hardware interface. System-level tests of real-time systems should address the following questions:

- Are interrupt priorities properly assigned and handled?
- Is processing for each interrupt handled correctly?
- Does the performance of each interrupt-handling procedure conform to requirements?
- Does a high volume of interrupts arriving at critical times create problems in function or performance?

Extreme Testing

In the 1990s, a new software development methodology called extreme programming (XP) was born. Extreme programming is a lightweight, agile development process that supports programming languages such as Java, Visual Basic and C#. The XP model relies heavily on module unit and acceptance testing. In general, you must run unit tests for every incremental code change, no matter how small, to ensure the code base still meets its specifications. In fact, testing is so important in XP that the process requires you to create the unit (module) and acceptance test first and then create your code base. This form of testing is called, appropriately, “extreme testing.”

XP focuses on implementing simple designs, communicating between developers and customers, constantly testing your code base, refactoring to accommodate specification changes, and seeking customer feedback. XP tends to work well for small- to medium-size development efforts in environments with frequent specification changes and where near-instant communication is possible. It differs from traditional development processes in the following ways:

- It avoids the large-scale project syndrome in which the customer and programming team meet to design every detail of the application before coding begins.
- It avoids coding unneeded functionality. The software team focuses on the task at hand, adding value to a software product and focusing only on the required functionality that helps create quality software in short time frames.
- It primarily focuses on testing. Traditional software development models suggest you code first, then create interfaces later. In XP, you must create the unit test first and then create the code to pass the tests. There are 12 principles that are noteworthy to be mentioned in this dissertation:

1. Planning and requirements

- Marketing and business development personnel gathering work together to identify the maximum business value of each software feature.
- Each major software feature is written as a use case or user story.
- Programmers provide time estimates to complete each user story.
- The customer chooses the software features based on time estimates and business value.

2. Small, incremental releases

- Strive to add small, tangible, value-added features and release a new code base often.

3. System metaphors

- Your programming team identifies an organizing metaphor to help with naming conventions and program flow.

4. Simple Designs

- Implement the simplest design that allows your code to pass its unit tests. Assume change will come, so don't spend a lot of time designing; just implement.

5. Continuous Testing

- Write unit tests before writing their code module. Each unit is not complete until it passes its unit test. In addition, the program is not complete until it passes all unit tests and acceptance tests are complete.

6. Refactoring

- Clean up and streamline your code base. Unit tests help ensure that you do not destroy the functionality in the process. You must rerun all unit tests after any refactoring.

7. Pair Programming

- You and another programmer work together, at the same machine, to create your code base. This allows for real-time code review, which dramatically increases bug detection and resolution.

8. Collective ownership of the code

- All code is owned by all programmers. No single programmer is dedicated to a specific code base.

9. Continuous integration

- Every day, integrate all changes, after it passes the unit tests, back into the code base.

10. 40-hour work week

- No overtime is allowed. If you work with dedication for 40 hours per week, the overtime will not be needed. The exception is the week before a major release.

11. On-site customer

- You and your programming team have unlimited access to the customer so you may resolve questions quickly and decisively, which keeps the development process from stalling.

12. Coding Standards

- All code should look the same. Developing a tailored system coding standard helps meet this principle.

In conclusion, it is important to follow the principles of software testing and try a variety of testing techniques, including white-box and black-box testing. It is also crucial to learn more about the modern techniques of extreme programming and extreme testing. I hope you've learned more about software testing and that these principles can serve as the basis and a good foundation for a healthy test program.

REFERENCES

- Myers, G. J. (2004.) "The Art of Software Testing." Word Association, Inc.,
 Pressman, R. S. (2005.) "Software Engineering: A Practitioner's Approach, Sixth Edition." McGraw-Hill International Edition.
 Wikipedia, The Free Encyclopedia: Software Testing, Zero Crossing.

ABOUT THE AUTHOR



Lynn Brown has worked with software for more than 20 years and received her master's degree in software engineering from the University of Alabama in Huntsville, Alabama, a school known for its accomplishments in science and technology. Most recently, Lynn worked as a lead in software safety evaluations for the Army Navy Transportable 2 (ANTPY-2) radar. Lynn is a certified quality engineer, holds A+ certification, and has a degree in electrical engineering. She aspires to become a certified software quality engineer. A native of Athens, Alabama, Lynn enjoys dance, art and her work in the ministry.

Combining Agile Practices with CMMI Process Areas

A Case Study

By Dick Carlson (Lead Author) and Earle Soukup (Co-author)

Introduction. This case study details how a large enterprise-level program used Agile methods, including Scrum, and implemented the capability maturity model integration (CMMI) at the same time. The purpose of the program was to modify a large commercial software application to replace an existing and obsolete enterprise-level data management system. The application of the CMMI to level 3 was a company mandate.

The CMMI is a process improvement training and appraisal program administered by the CMMI Institute, a subsidiary of ISACA, an international professional association focused on IT governance. Often the CMMI is required by many DoD and U.S. government contracts, especially in software development.

The lead author, Dick Carlson, was hired as the program's Agile coach to transition the program from a failing, document-laden system with high overhead activity to Agile methods, using Scrum. He would also guide the program in meeting CMMI requirements.

The effort proved to be a challenge for Scrum, as it had successfully managed small projects but was not yet known for managing larger programs. The program was expected to continue for another seven years to modify, design and develop major modifications before software improvements were deployed.

The company's leadership team drove implementation of the large program. Its IT organization used "Macroscope," an integrated, evolving and adaptable body of knowledge designed to help reduce risks and maximize the value of business transformation and information technology initiatives. However, the misuse of the "Macroscope" model's implementation led to extensive cost overruns, schedule slippages and large-scale program planning. As a result, it took more than a year to prepare and execute.

Senior management insisted that the program comply with the CMMI, and that its execution activities would align with the organization's overarching process improvement goals. Management hired Mr. Carlson to guide the program in adopting a hybrid process improvement approach, using Agile methods and aligning with specific CMMI practices.

AUTHOR'S NOTE: The company's name is protected by a strict code of ethics regarding specific business practices and customer identification. Therefore, references to the company, its customers and the name of any program cannot be disclosed.

The Process

Due to the magnitude of the program, seven discrete project teams were formed. Each team was chartered to a distinct engineering domain, where its relationship with other teams of same or similar domains was critical to developing and implementing the application.

[NOTE: The application's out-of-the-box functionality did not meet the enterprise's operational needs. By company mandate, the vendor agreed to modify many of the application's functions to comply with the enterprise's operational requirements.]

Project team activities were monitored closely using the Scrum of Scrums technique in scaling multiple teams and large teams. They created a foundational team responsible for establishing critical work areas; fabricating test benches; building initial architectures critical to new and modified application components; establishing, updating and regulating a variety of standards across all teams; and coordinating certification events.

Each project team was assigned a Scrum master and product owner. This was necessary to:

- Ensure teams remained focused.
- Make sure requirements were well-defined and developed when they were needed.
- Help team members remain engaged and productive.
- Maintain product backlogs, schedules and release plans.

The program had one program manager, three project managers and three program and project engineers. Mr. Carlson was required to report directly to the program director and coordinate closely with the program manager, all project managers, and other key stakeholders.

The program's product development plans were required to identify the program's quality requirements, which included:

- Using Scrum framework for new feature development and a hybrid Agile and waterfall approach for product sustainment.
- Resolving errors during development sprints.
- Recording escaped errors (defects), analyzing them for root cause and tracking them to closure.
- Establishing and maintaining CMMI maturity level 3 practices, as appropriate.
- Maintaining quality standards for multiple government accreditations.
- Documenting applicable procedures for implementing Agile methods and verifying through quality checklists, collaborative peer reviews and supplier quality activities.
- Modifying current processes significantly to accommodate the hybrid methods.
- Participating in rigorous risk management.
- Participating in configuration and data management.

Benefits

Mapping the CMMI and Scrum

CMMI practices are designed to provide value across a range of different situations and thus are stated in general terms. Because CMMI does not endorse any particular approach to development, little approach-specific information is provided. Therefore, those who don't have prior experience implementing CMMI in situations similar to their current situation may find it difficult to interpret the practices.

The following tables identify relevant CMMI practices for complying with maturity level 2 (using text taken from the model). They also show how Scrum practices were implemented in each CMMI specific practice (SP) on the large project. To be appraised at maturity level 2, the Scrum implementation had to show evidence of

PP	CMMI Practice	Scrum Practice
SP 1.1	Establish a top-level work breakdown structure (WBS) to estimate the scope of the project	The standard tasks used in Scrum are combined with specific project tasks listed in sprint backlogs, which are quantifiable
SP 1.2	Establish and maintain estimates of the attributes of the work products and tasks	Story points are used to estimate the difficulty (or relative size) of a user story (requirement)
SP 1.3	Define the project life-cycle phases upon which to scope the planning effort	The Scrum model contains the framework, lifecycle, and process
SP 1.4	Estimate the project effort and cost for the work products and tasks based on estimation rationale	Ideal time estimates (similar to billable hours or full-time equivalents)
SP 2.1	Establish and maintain the project's budget and schedule	- Scrum estimates reflect ideal time - Estimates of work that will be completed in each release - Sprint backlog identifies work that must be completed during a sprint - A sprint task board shows planned vs. actual schedule and the amount of work completed
SP 2.4	Plan for necessary resources to perform the project	- Scrum estimates in ideal time - Release plan, sprint backlog and team agreements are produced
SP 2.6	Plan the involvement of identified stakeholders	Scrum roles (including team, Scrum Master, Product Owner) [Note: The stakeholders listed in Scrum may not be the complete list of stakeholders for the project, e.g., customers, users, and other teams]
SP 2.7	Establish and maintain the overall project plan content	- Scrum release plan - Sprint backlog - Sprint task board [Note: The term "plan" in CMMI refers to additional plan components such as risks and data management that are not called out specifically in Scrum.]
SP 3.1	Review all plans that affect the project to understand project commitments	- Sprint planning meeting - Daily Scrum or stand-up meeting - Sprint task board - Product roadmap - Product vision - Release plan
SP 3.2	Reconcile the project plan to reflect available and estimated resources	- Daily Scrum meeting - Product roadmap - Product vision - Release plan
SP 3.3	Obtain commitment from relevant stakeholders responsible for performing and supporting plan execution	- Sprint planning meeting - Daily Scrum meeting [Note: The stakeholders listed in Scrum may not be the complete list of stakeholders for the project]

Table 2. Project Planning

CMMI practices being performed. A discussion of the program's CMMI maturity level 3 processes follows later.

CMMI Maturity Level 2 Process Areas

Requirements Management

The purpose of requirements management (REQM) is to manage the project's products and product components requirements. It also helps to identify inconsistencies between those requirements and the project's plans and work products. (see Table 1.)

REQM	CMMI Practice	Scrum Practice
SP 1.1	Develop an understanding with the requirements providers on the meaning of the requirements	Review of product backlog (requirements) with the Product Owner and team
SP 1.2	Obtain commitment to the requirements from the project participants	Conduct sprint planning sessions that seek team member commitment
SP 1.3	Manage changes to the requirements as they evolve during the project	- Add requirements changes to the product backlog - Manage changes in the next Sprint planning meeting and throughout the duration of the project
SP 1.5	Identify inconsistencies between the project plans and work products and the requirements	- Daily standup meeting identifies issues - Sprint planning sessions address inconsistencies - Sprint burndown chart tracks effort remaining - Release burndown chart tracks story points that have been completed, which shows how much of the product functionality is left to complete

Table 1. Requirements Management

Project Planning

The purpose of project planning (PP) is to establish and maintain plans that define the project's activities. (See Table 2.)

Project Monitoring and Control

The purpose of project monitoring and control (PMC) is to understand the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan. (See Table 3).

Other Level 2 Process Areas

Configuration Management (CM)

CM is not specifically called out in Scrum. However, in an Agile environment it is easy to add a layer of CM to protect work products. Even for groups that like to use white boards, you can be creative and at least establish some basic protection by labeling items (e.g., "V1.1" or "Story dated YYYY/MM/DD"), by taking a photo, or by using other documenting techniques. The CM process area requires more than just versioning, but versioning is a good and fundamental start.

Product and Process Quality Assurance (PPQA)

Some basic PPQA activities are being accomplished naturally when the Scrum master checks that the Scrum process is being followed. Other PPQA activities are completed when a team performs code reviews, document reviews and validation testing. The Scrum master also removes process barriers and inefficiencies. However, Scrum does not specifically call

out a level of *objective* process and product check, nor does it say that particular standards or processes should be defined and used. Therefore, Scrum does not automatically implement PPQA. However, refinements can be implemented to support a PPQA claim.

Supplier Agreement Management

There are no practices in Scrum that deal with selecting and managing suppliers.

Measurement and Analysis

The purpose of measurement and analysis (MA) is to develop and sustain a measurement capability to support management information needs. There are no practices in Scrum that establish a measurement program similar to the expectations of MA. However, the measures in Scrum can be used to implement MA (see Table 4).

MA	CMMI Practice	Scrum Practice
SP 1.2	Specify measures to address the measurement objectives	- Sprint burndown chart tracks effort remaining - Release burndown chart tracks completed work, and shows how much of the product functionality is left to complete [Note: These two measures could be used to track the progress of declared project objectives, such as "On time," and "On budget."]
SP 1.4	Specify how measurement data will be analyzed and reported	The Scrum process describes the purpose and use the Sprint and Release burndown charts [Note: CMMI expects clearly defined analysis]
SP 2.1	Obtain specified measurement data	Daily Scrum meeting where sprint and release burndown data are collected
SP 2.2	Analyze and interpret measurement data	Daily Scrum meeting inspects sprint and release burndown data
SP 2.4	Report results of measurement and analysis activities to all relevant stakeholders	Daily Scrum meeting where sprint and release burndown charts are reviewed. [Note: Not all interested stakeholders will attend Scrum meetings.]

Table 4. Measurement and Analysis

Generic Practices

The table below shows the practices that have the clearest relationship to Scrum measurements. Approximately half of the level 2 generic practices of requirements management, project planning, and project monitoring and control are implemented by Scrum (see Table 5 on page 20).

CMMI Maturity Level 3 Compliance Requirements

There are two main areas where Scrum has gaps for level 3 compliance. One area is in the CMMI expectation that project data and lessons learned are shared among projects via a common process asset library (or repository). Another is the expectation that the engineering phases of requirements, design, implementation, verification, integration and validation are well-defined and implement level 3 engineering practices.

These CMMI concepts can be implemented in a Scrum environment, but they do not come with the common Scrum definition. The Scrum Alliance suggests implementing "communities of practice" to reach across teams to share lessons learned and to conduct "retrospectives" within a team. These activities can populate an asset library and thereby record best practices and tailoring guidelines.

PMC	CMMI Practice	Scrum Practice
SP 1.1	Monitor the actual values of the project planning parameters against the project plan	- Sprint burndown chart tracks effort remaining - Release burndown chart tracks completed story points. This shows how much of the product functionality is left to complete. - Project task board is used to track user stories (requirements) that are done, in progress, or those needing verification.
SP 1.2	Monitor commitments against those identified in the project plan	- Discussions on team commitments at the: • Daily Scrum meeting • Sprint planning meeting - Sprint burndown chart tracks effort remaining - Release burndown chart tracks story points that have been completed. This shows how much of the product functionality remains to be complete.
SP 1.5	Monitor stakeholder involvement against the project plan	- Discussions at the: • Daily Scrum meeting • Sprint planning • Sprint demo [Note: The stakeholders listed in Scrum might not be the complete list of stakeholders for the project, e.g., customers, or other impacted teams.]
SP 1.6	Periodically review the project's progress, performance, and issues	- Daily Scrum meeting - Sprint review meeting - Retrospective (lessons learned and opportunities for improvement)
SP 1.7	Review the accomplishments and results of the project at selected project milestones	- Sprint review meeting - Retrospective (lessons learned and opportunities for improvement)
SP 2.1	Collect and analyze the issues and determine the corrective actions necessary to address the issues	Notes from the: • Daily Scrum meeting • Sprint planning • Sprint demo • Retrospective (lessons learned and opportunities for improvement) [Note: Some teams track their outstanding actions on the product backlog. It doesn't matter where or how the items are tracked, as long as they are tracked]
SP 2.2	Take corrective action on identified issues	Actions from: • Daily Scrum meeting • Sprint demo • Retrospective (lessons learned and opportunities for improvement)
SP 2.3	Manage corrective actions to closure	Tracking of actions from: • Daily Scrum meeting • Sprint demo • Retrospective (lessons learned and opportunities for improvement) [Note: This assumes that teams will track (and not lose actions)]

Table 3. Project Monitoring and Control.

Despite the upside potential for a community of practice, there are challenges that have not received sufficient attention from either the business or academic community. If formal, rational structures result when organizations attempt to conform to their understanding (correct or incorrect) of social reality, industries with less formal, less rational structures may have had less time to implement structural conformance. This will likely correlate to industry age.

Communities of practice may benefit from a flatter, horizontally linked organization. For an organization to seriously consider moving from a more vertical to a more horizontal structure, the firm will likely need to reassess its core competencies, capabilities and culture. It should also consider the challenges

GP	CMMI Practice	Scrum Practice
GP 2.2	Establish and maintain the plan for performing the REQM/PP/PMC process	The Scrum lifecycle definition and the schedule to perform Scrum
GP 2.3	Provide adequate resources for performing the REQM/PP/PMC process, developing the work products, and providing the services of the process	The resources and schedule time allocated to perform Scrum planning, monitoring and requirements activities
GP 2.4	Assign responsibility and authority for performing the process, developing the work products, and providing the services of the REQM/PP/PMC process	The resource assignments allocated to perform Scrum planning, monitoring and requirements activities
GP 2.5	Train the people performing or supporting the REQM/PP/PMC process as needed	Scrum team training for the aspects of Scrum that relate to REQM/PP/PMC, planning and monitoring
GP 2.6	Place designated work products of the REQM/PP/PMC process under appropriate levels of control	[Note: Scrum does not explicitly require CM to be done. However, this can be performed using a digital camera, backed up drive, or shared drive with versioning and controls turned on.]
GP 2.7	Identify and involve the relevant stakeholders of the REQM/PP/PMC process as planned	The list of Scrum team members [Note: The stakeholders listed in Scrum might not be the complete list of stakeholders for the project, e.g., customers, other impacted teams.]
GP 2.8	Monitor and control the REQM/PP/PMC process against the plan for performing the process and take appropriate corrective action	Scrum Master monitors that the Scrum process is followed.
GP 2.10	Review the activities, status, and results of the REQM/PP/PMC process with higher level management and resolve issues	[Note: This CMMI practice is implemented if the project or senior manager has regular visibility into how well Scrum is working.]

Table 5. Generic Practices

that may threaten the present continuity in its relationship with all stakeholders in such an undertaking.

Therefore, the following level 3 components are not readily implemented by Scrum without additional work:

- Organizational process focus (OPF).
- Organizational process definition (OPD).
- Organizational training (OT).
- Product integration (PI).
- Risk management (RSKM).
- Requirements development (RD).
- Technical solution (TS).
- Some engineering-specific practices (e.g., requirements validation and verification data analysis).
- Generic goal 3 (i.e., using an organization-wide and tailored process with measurements).

Although program teams used the Scrum of Scrums practice, leaders got waivers against CMMI level 3 requirements. Fulfilling the requirements could compromise the enterprise's involvement in enterprise-wide appraisals of its activities.

Scrum Implementation Results

Using the Scrum approach to manage all program activities resulted in the following outcomes:

- High business value was delivered faster (as much as 50 percent improvement in productivity compared to previous Macroscopic work efforts).
- Applying Lean principles helped the company to better understand and implement Scrum practices.

- Scrum masters documented each team member's stand-up responses (helpful for root-cause analyses) to the following questions:
 - What have you completed or implemented since the previous stand-up meeting?
 - What are you planning to complete or implement next?
 - Do you have impediments preventing you from completing what you plan to do next?
 - Do you have any issues?
- Impediments were turned into action plans. Core Scrum team members reviewed and determined appropriate actions by either assigning someone to carry out the action or by escalating the action to management.
- Scrum masters conducted mini retrospectives at the end of each sprint week to capture potentially good ideas before they were forgotten.
- Relationships with key stakeholders improved, which helped to build trust and grow knowledge.
- Team members created a Scrum dashboard with all vital information about the team, their agreements, and their commitments. It also included sprint activities and goals, definitions of "done," tasks, sprint metrics, announcements, action items, parking lot items and retrospective inputs for activities that needed corrective action or improvement.
- Team members developed a guide for Scrum masters that provided definitive guidance while they performed their duties.
- Scrum teams developed a sprint collaboration peer review process, used by all teams to document specific actions and comments on reviewed documents, design objects, code and other work products.

The Results

The modified application's operational success was confirmed in early 2013, and the application was successfully implemented in late 2014. Significant follow-up work continues as the application owner improves the product and organizational changes necessitate revisions.

Case Study Summary

It will take time for an Agile-CMMI hybrid implementation to work effectively across the enterprise, as the CMMI does not prescribe implementation details. A crucial change causes a significant cultural shift. Combining the CMMI and Scrum is not easy, as two cultures must be brought together. Both sides must learn to understand each other and look for all potential connection points. As teams overcome the different cultural aspects, the quest for process improvement and software quality will be combined.

While the team met essentially all of the CMMI level 2 requirements for this program, the analysis of the effort presented in this case study confirmed that several of the CMMI level 3 project procedural requirements were met. To meet these requirements, a company must have defined processes as standard and repeatable business practices. CMMI levels as high as 4 and 5 are only achievable with a consistent, stable and collaborative effort from all stakeholders. They also occur when all levels of management impart their commitment and loyal support.

REFERENCES

Agile (<https://www.agilealliance.org>).
 CMMI (<https://www.sei.cmu.edu/cmmi/>).
 Macroscopic (<http://macroscopic.ca.fujitsu.com/Products/the-macroscopic-methodology.html>).
 Scrum (<https://www.scrumalliance.org>).

SUGGESTED READING

“CMMI” for Development, Version 1.3.” Software Engineering Institute Report, 2010.
 “Essential Scrum: A Practical Guide to the Most Popular Agile Process.” Kenneth S. Rubin, 2012.
 “Understanding the Agile Manifesto.” Larry Apke, 2015.
 “Limitations of Communities of Practice: A Consideration of Unresolved Issues and Difficulties in the Approach” (<http://www.knowledgemobilization.net/wp-content/uploads/2014/02/8.-Limitations-of-Communities-of-Practice-.pdf>).

ABOUT THE AUTHORS



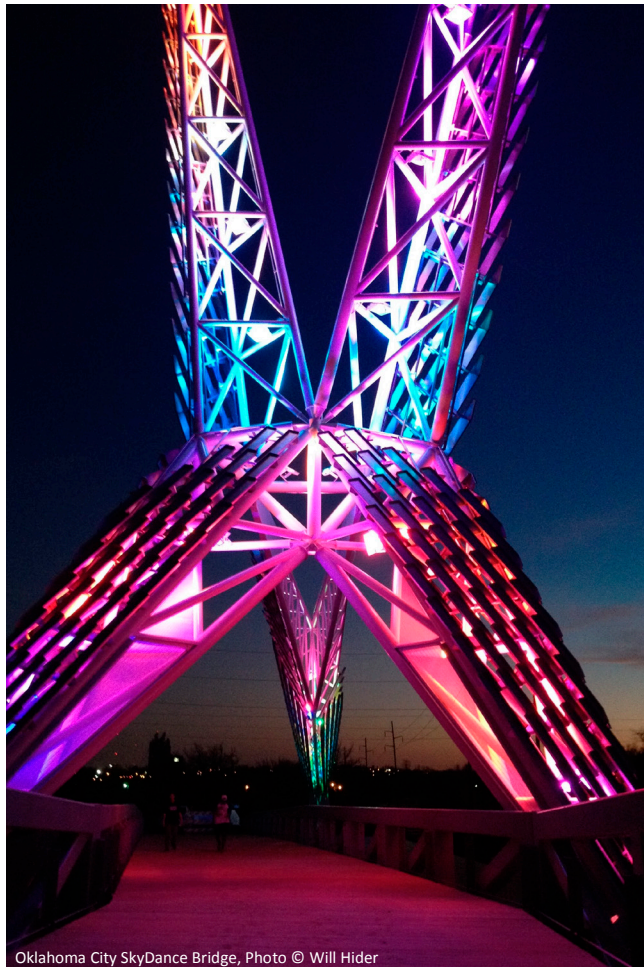
Dick Carlson has a Bachelor of Science degree in business management. He has held certifications as a Scrum Professional, Scrum Master, and Scrum Product Owner, and in Lean-Agile Project Management. He is an accomplished software engineering process analyst, and has shared successful experiences of Agile, Lean, and Scrum implementation at conferences, workshops, and symposia. Dick's engineering career spans more than 40 years, and he has taught courses in mathematics, electronics, CMMI, configuration management, data management, Agile, Lean, and Scrum for more than 30 years.

dcarlson@iascar.us



Earle Soukup holds a Bachelor's and Master's degree in Electrical Engineering, a Juris Doctor in Law, certificates for software development, project management, functional management, systems engineering, and Agile and Lean including being a Certified Scrum Master. He was a development and test engineer, manager, and project manager for both hardware and software products. Also he developed and taught courses in mathematics, electronics, ethics, Lean, and Agile. Mr. Soukup is an accomplished analyst.

erlsoukup@gmail.com



Oklahoma City SkyDance Bridge, Photo © Will Hider

WE ARE HIRING

ELECTRICAL ENGINEERS AND COMPUTER SCIENTISTS

As the largest engineering organization on Tinker Air Force Base, the 76th Software Maintenance Group provides software, hardware, and engineering support solutions on a variety of Air Force platforms and weapon systems. Join our growing team of engineers and scientists!

BENEFITS INCLUDE:

- Job security
- Potential for career growth
- Paid leave including federal holidays
- Competitive health care plans
- Matching retirement fund (401K)
- Life insurance plans
- Tuition assistance
- Paid time for fitness activities

Tinker AFB is only 15 minutes away from downtown OKC, home of the OKC Thunder, and a wide array of dining, shopping, historical, and cultural attractions.



Send resumes to:
76SMXG.Tinker.Careers@us.af.mil
US citizenship required

Using Contracts to Reduce Cybersecurity Risks

C. Warren Axelrod, Ph.D.

Abstract. Would we achieve higher standards for software and data security if contractors and subcontractors accepted stringent cybersecurity requirements in software development agreements, vendors signed off on similar requirements in software license agreements, and service providers included cybersecurity components in their offerings?

One might expect that contractual provisions would improve the current situation, but we frequently see that results do not turn out as well as anticipated, usually for ignored or unrecognized reasons.

In this article, we will identify cybersecurity risks inherent in developed and acquired software products as well as software-related service issues. We will then describe how to mitigate these risks through contractual terms and conditions. Finally, we will discuss the conditions that may affect agreed-upon contracts.

Background

There are many publications about cybersecurity risks and mitigating them, and quite a few resources about provisions that should be included in general IT agreements. However, there is much less information on translating cybersecurity risk-mitigation requirements into meaningful contractual clauses, negotiating provisions embodied in those clauses, and enforcing those provisions.

The overall risk assessment and contract development and management processes are represented in Figure 1. We will discuss each phase as it applies to software cybersecurity for various categories of software. We will also:

- Discuss identifying logical and physical software-related risks.
- Suggest various mitigation strategies.
- Translate those strategies into cybersecurity requirements.
- Describe the costs and benefits of each recommendation.
- Prioritize the risks based on cost-benefit analyses.

Each of these steps is necessary so that customers can enter into meaningful and productive contract negotiations with prospective suppliers. Once the terms and conditions of the relationship between customer and supplier have been established and agreed upon, they can draft a contract and further negotiate it. Customers and vendors must make sure contractual terms are followed once the contract has been executed. They must also take action when deviations occur.

Different Types of Software

Software is generally program code that produces computer capabilities, including:

- Creating documents and spreadsheets.
- Managing financial transactions.
- Controlling nuclear power plants.

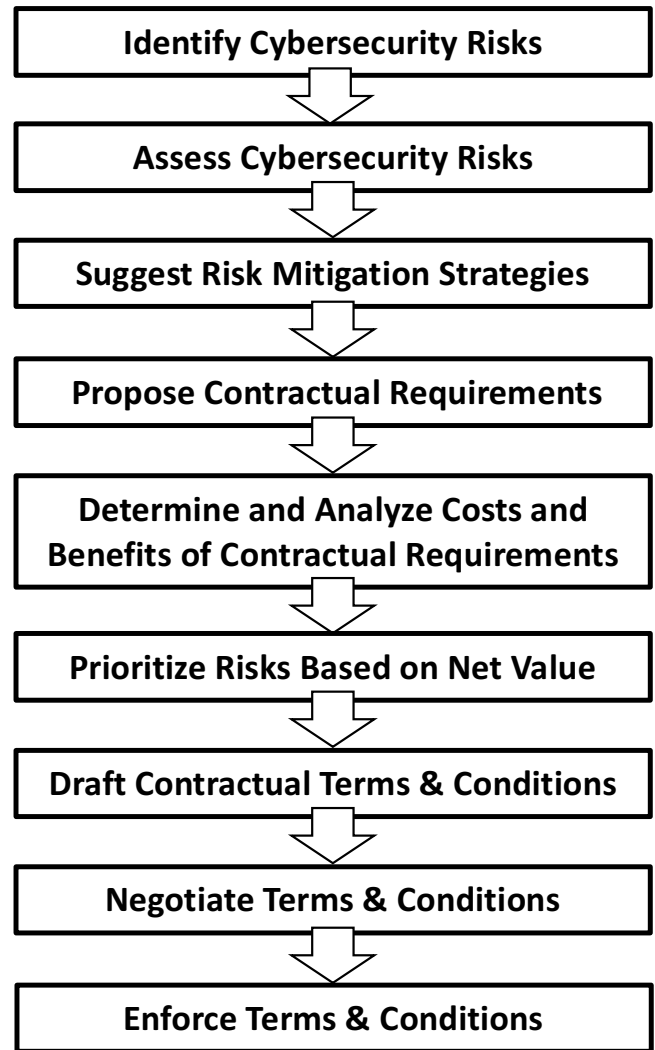


Figure 1. The process of developing and enforcing software cybersecurity clauses in contracts

Software is ubiquitous and is continuing to disperse across the world. Indeed, some far-sighted major manufacturing companies, such as General Electric [1], are changing their business models to accommodate and benefit from the emerging software-intensive world.

The various categories of software have been around for more than half a century. They include:

- Custom-built software programs.
- Internal development, testing and implementation.
- External development, testing and implementation.
- Hybrid internal/external development, testing and implementation.
- Pre-packaged software products.
- Commercial off-the-shelf (COTS) software.
- Government off-the-shelf (GOTS) software.
- Customized COTS and GOTS.
- Open-source software.
- Embedded software.
- Software-based services.
- Direct services.
- Software as a Service (SaaS) in the cloud.

Different types of contracts apply to each of the above software categories, and these contracts handle cybersecurity in a variety of ways. These depend on the context in which the software is run and their business criticality.

The Importance of Contracts

Those who negotiate and enforce computer contracts are familiar with the all-too-common vendor statement: “We’ll put this contract in a drawer (or file folder) and never reference it again.” Of course, this is often the case. When contractual terms need to be invoked, however, that statement often indicates an adversarial situation. This is also a ploy to make customers feel more comfortable with boilerplate vendor agreements and not push as vigorously as they might otherwise for stiffer contractual terms and conditions.

However, not standing up for certain contract provisions that are important to your organization is a mistake. This approach is commonly used with take-it-or-leave-it contracts, where vendors are not willing to negotiate at all. The above sentence may be accompanied by this reassurance: “Well, our major customers have already agreed to this.” Such assertions may not necessarily be true and should be verified if possible.¹

Contracts serve the very important purpose of mitigating risks — not only for customers but also for suppliers — and of specifying each party’s responsibilities and liabilities when a covered event occurs or when there is a breach of contract.² It is up to each party to anticipate what those risks might be and to specify how to handle them. When it comes to computer software, risks have escalated rapidly over time, with massive changes happening at an accelerating pace during recent years.

Evolving Software Cybersecurity Risks

The software marketplace has changed dramatically over the past several decades. Prior to the 1980s, software licenses were often bundled into overall computer-system purchases.³ You could choose from a limited selection of commercial off-the-shelf products — mostly compilers, utilities and other system software. The other option was building your own applications and system software or retaining third parties that provided software development and system integration services.

A lot of proprietary system and communications software — and some application software products — were managed under all-encompassing take-it-or-leave-it contracts. These were generally nonnegotiable and came from companies like IBM and Digital Equipment Corporation (DEC).⁴

Even when acquiring third-party software from less dominant vendors, the best you could expect was for the source code to be stored in escrow against the possibility that the vendor would go out of business or no longer support the product. In this case, customers would then have to support the product themselves, but at least they would have more readable source code to work with.⁵

In subsequent decades, third-party reviews of vendors’ products, services and facilities became more common. Customers often used these to reassure themselves — and their auditors — about the quality of the product or service.⁶

On the liability side, software vendors have notoriously avoided responsibility for ensuring that software actually works as intended. At most, they might agree to pay customers up to the

purchase price of the product if it failed to live up to expectations. However, they have studiously avoided committing to pay for consequential damages attributable to the software.

This may be changing as more software is used to control safety-critical systems, such as those operating autonomous vehicles,⁷ especially in the wake of a fatal accident involving a Tesla car running on “autopilot.”

With the advent of personal computers and ubiquitous internal and external networks, a new category of software purchaser arose — the individual. It wasn’t that commercial and government entities didn’t continue to make up much of the market by value. They did and still do.

However, we began to see a change in the balance between vendor and customer, where individuals were forced to acquire software products on a take-it-or-leave-it basis. A single person didn’t have any negotiating power and was unable to confront vendors successfully by himself.

Along with the proliferation of distributed computing came concerns about security and privacy. Previously, the main concern had been protecting the confidentiality of data for business reasons.⁸ Personal and business computer systems became victims of increasing numbers of sophisticated attacks in which hackers stole information for espionage, identity theft and fraud. Over time, it became apparent that the vast majority of successful attacks came through the application layer, yet customers had little control over the security of the applications they used.

Today, with the burgeoning use of the cloud, application software is bundled into SaaS (software as a service) and system software (e.g., operating systems) into PaaS (platform as a service) offerings. Because of this, both organizations and individuals remain relatively powerless when it comes to asserting their requirements and desires for higher-quality software.

Custom-built software is another matter. If software is developed in-house, an organization ostensibly has control over the quality and security of that software. Yet building security into applications came late and still has relatively little momentum. Security is often subject to competitive features, marketability, convenience, performance and other factors.⁹

When software is built by contractors, who might further subcontract some of the work, and the developers incorporate commercial and open source products into the system, more problems occur. We don’t know the provenance of each component (see [2]), don’t have adequate oversight of external development shops, and can’t test all components in all expected operational contexts.

Detailing the Risks

Logical and physical computer-related risks have evolved dramatically in the half-century since computers were introduced into common government, business and academic use. The risks have proliferated and become much more complex in the 25 years since the advent of the internet. Relatively speaking, omnipresent cybersecurity risks are a new phenomenon and have only really taken hold in the past decade or so.

In Table 1, we show the progression of pre-internet technologies and how logical and physical risks have changed from one computer era to the next. The model of the highly-centralized, well-guarded data center overseen by specially-trained technicians gave

*Table 1.
Logical and
Physical
Risks by
Technology
Category in
the Pre-Inter-
net Era*

Technology	Logical Risks	Physical Risks	Comments
Mainframes	- Availability—mean time between software errors and failures, mean time to update software - Performance of software - Confidentiality—file and data access management - Data and file integrity	- Availability—mean time between malfunctions and failures, mean time to repair - Performance management - Capacity management - Physical barriers to access to equipment	Little-to-no attention paid to protecting personal privacy, e.g., IRS printed Social Security numbers on address labels
Minicomputers	- Availability—as for mainframes - Security concerns—access rights of super-users in particular - Data and file integrity	As for mainframes	- Focus on cost of processing power vs. mainframes (target of about 20% lower) - No security software (Unix was very open in its initial versions)
Personal computers	- Dearth of knowledgeable, inexpensive technical support—difficult to justify high support costs relative to cost of software - Vulnerability of users who are not knowledgeable about what risks they might be taking or are aware but do not know how to deal with them - Availability may be reduced because of insufficient resilience and recovery - Security concerns—viruses, stealing of access codes (e.g., passwords posted on or near device), “shoulder surfing,” takeover of unattended devices - Pirated software is common - Noncompliance consequences minimal	- Lack of readily-available low-cost technical support for individuals - Devices often placed in locations that are not physically secured - Portable devices, in particular, may be lost (e.g., left in a taxicab) or stolen	- PC era saw appearance of computer viruses that initially were transmitted via removable media, e.g., floppy disks - Early users had to be largely self-supporting with respect to technical issues
IT outsourcing	- Trustworthiness of service provider - Protection within shared software environments - Loss of control over: - how systems are supported - which applications are run - when applications are run - updating of technologies - reporting of data breaches - patching of software systems - Reliability and availability of networks and systems - Identity and access management - Software assurance - Secure development lifecycle - Secure coding practices - Implementation of DevSecOps¹⁰	- Trustworthiness of service provider - Isolation of systems and networks within shared physical environments - Viability of service provider and customer - Business intentions of service provider and customer - Capability maturity level	Source: Axelrod [2]
Local-area networks	- Access management—outsider could hijack approved user credentials and gain access to LAN - Possible for any LAN user to view and capture other users’ traffic since all messages pass through all nodes	- Outsider able to plug into organization’s internal LAN from internal locations, e.g., conference room - Hacker able to access internal LAN via a wireless access point from outside of nearby locations	Recommended practice is to install an encrypted protocol, such as WEP, on wireless access points
Private wide-area networks	- Possibility of unauthorized access and use - A private network is a specific target for hackers	May not provide the level of redundancy expected unless specified in the contract and paid for	Dedicated networks are more secure than shared networks or public networks such as the Internet

way to more distributed computing. This was frequently operated by staff with less training. Then, when the personal computer was introduced, non-technical individuals controlled the local systems.

Computer networks were initially limited in scope and geography and were supported by relatively few specialists. But as the use of minicomputers and personal computers expanded, so did the interconnections within physical locations and among them. However, they were still clearly specified and known to technical support engineers. Under these circumstances, it was still possible to identify counterparties, negotiate agreements with them, and invoke contractual terms when necessary.

Then, as shown in Table 2, along came the internet. No one knew much about the source of software products being used, especially those embedded within services, and the specific network topologies applicable to particular users and organizations.

As a result, it has become much more difficult to identify responsible counterparties and hold them accountable through the previously effective contractual methods. This has exposed the need to revisit contractual relationships and come up with ways to negotiate and enforce terms and conditions. These are necessary to protect individuals and organizations.

Table 2. Logical and Physical Risks by Technology Category in the Post-Internet Era

Technology	Logical Risks	Physical Risks	Comments
Public wide-area networks, e.g., Internet	- Proliferation of malware, e.g., viruses, worms, Trojan horses - Account hijacking of bank accounts, brokerage accounts, etc. - Cyber bullying over social networks - Need to balance strong encryption, privacy and anonymity - Takedown of networks using DDoS (distributed denial of service) attacks - Upsurge in “ransomware,” whereby victims are warned that their data will be destroyed unless they pay a ransom - Use of phishing in order to obtain customer credentials (username, passwords, etc.)	- Sniffing over physical and wireless links to determine user credentials, message content, and transaction details - Cutting of undersea and underground cables either accidentally or intentionally	Adverse circumstances, such as hacker attacks, are increasing faster than are ways to protect against them
Web applications	- Lifecycle—requirements, specifications, design, development, testing, deployment and disposal - Security architecture, secure coding, and functional security testing - Identity and access management compromise—credential theft - Functional manipulation and destruction - Security—threats, exploits, attacks, vulnerabilities, weaknesses - Application denial of service - Data manipulation and theft: <i>Privacy</i>—read, copy, manipulate, destroy nonpublic personal data <i>Confidentiality</i>—read, copy, manipulate, destroy sensitive personal, corporate and government data	Getting physical control of devices makes for ability to hack applications more intensely	- There is a substantial body of work on application security - There are organizations, such as OWASP (Open Web Application Security Project) that provide many resources such as tools, guidelines, and the like - There are websites that emphasize the importance of building security in at various phases of the system development lifecycle (SDLC) - There are online resources that list weaknesses, vulnerabilities, etc., such as CWE (Common Weaknesses Enumeration)
Mobile devices	As for personal computers, plus: - Security concerns—less developed security software as compared to PCs - Apps are common source of malware - Social networks, in particular, are susceptible to privacy violations - Debate as to whether vendors should provide backdoors or ability of law enforcement to easily break encryption for cell phones	As for personal computers, plus: Smaller, ubiquitous, more portable devices are more likely to be lost, stolen or damaged than larger laptop computers	- Primary requirements of mobile devices are convenience, ease-of-use and portability, which often conflict with security - Rapid growth in numbers of mobile devices and apps means that: - greater numbers of technology-naïve users - more intense use of devices¹¹ - more threats and vectors for malware, fraud, etc.
Cloud services	As for IT outsourcing, plus: - Transactional versus relationship association in many cases, especially with individuals and small groups - Portability issues may make transferring from one provider to another, or to in-house facilities, difficult - Interoperability issues may create problems with respect to interfacing systems across multiple cloud service providers and/or internal systems - Security capabilities may be available but it is generally up to the customer to implement them	As for IT outsourcing, plus: - May not know the country in which data center resides, although some providers allow customer to choose location - Not knowing location may present major issues for customers who must comply with local laws and regulations	The Cloud Security Alliance is an excellent source of resources with regard to security issues of cloud services¹²
Cyber-physical systems	- Control—takeover of control by internal and external parties - Security concerns—as for Web applications - Safety concerns—software errors or compromise may allow hijacking and lead to malfunctions and failures	- Logical access to physical equipment via cyber attack - Physical access to facilities and equipment - Safety concerns—hijacking of physical equipment may lead to malfunctions and failures	See Axelrod [5]
Internet of things (IoT)	- Control, security and safety—as for cyber-physical systems - Embedded software is often custom built by engineers not knowledgeable in cybersecurity leaving products very vulnerable to hacking	- Logical access to physical “things” via cyber-attack—this can be very dangerous if systems control vehicles in the sky, on and in the sea, and on land, - Physical access to facilities and “things”—enormous number of devices makes it impossible to control them all physically	- Intentional takeover of vehicle controls has been demonstrated to be feasible and to potentially lead to injury or death - Unintended software and equipment errors can also lead to accidents.
Artificial intelligence (AI)	- Control—unanticipated takeover of control by software - Security—ability to change and adapt may compromise security - Safety—complexity of software may lead to unanticipated physical behavior	- Physical access may not be a straightforward option - Safety—complexity of hardware may lead to unanticipated physical behavior	Concerns about the dangers of ineffectively controlled AI have been expressed by the likes of Elon Musk and Stephen Hawking

Software Contracts and Cybersecurity

We now examine typical provisions included in software contracts by software category. It is important to realize that software products (or pre-packaged software) are typically licensed rather than purchased, so that the vendor retains ownership. The organization paying for the development usually owns custom-built software, whether internal staff or third parties are developing it. Open source software belongs to everybody or nobody. Yet even as it is freely available, those who use the software have contractual requirements. Embedded software can fall into any of the above categories, but the system or service provider generally handles conditions of use and support. The latter is retained in-house, managed by a third party or situated in the cloud.

Figure 2 illustrates how cybersecurity has been increasingly included in software contracts, for both products and services, over the past several decades and well into the future. The diagram intersperses the various technologies that have emerged over this same period. Considering that the internet did not become popular until the 1990s, the lack of awareness about cybersecurity for many years is quite understandable.

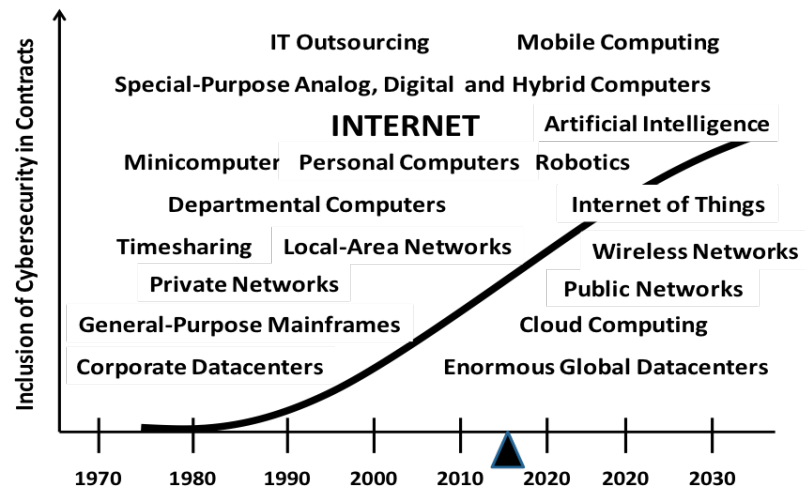


Figure 2. Increasing Inclusion of Security in Software and Service Contracts

Type of Software	Brandon & Segelstein [7]	Monassebian [8]
Prepackaged	• Performance • Financial • Installation • Support • Warranties	• Lower license fees • Availability in marketplace • Verifiable capability • Regular updates
Custom	• Performance • Financial • Installation • Staff	• Integration of updates • Taxes • Confidentiality of proprietary business methods • User involvement • Support for appropriate fees
Customized	As for “Custom” software	• Requirements • Responsibilities • Vendor personnel • Revenue sharing • Title and confidentiality for a fee

Table 3. Items Included in Contracts in the Late 20th Century

Until about 15 years ago, information security was barely mentioned in software products and software-intensive service acquisition agreements. In the last decades of the 20th century, software agreements focused on the items shown in Table 3, per [7] and [8].

To the extent that it was addressed at all, security was mostly directed at confidentiality by safeguarding the vendor’s intellectual property, such as source code. Buyers had little protection other than (perhaps) having the vendor place the source code in escrow so they could access it if the vendor went out of business or no longer supported the products or services.

While not ideal, working on source code is better than trying to reverse-engineer object code (or machine language). Other platforms (operating systems, system utilities, firmware and hardware) and data (media, devices, formats) issued were generally not considered. Yet these issues could, and did, present situations that were difficult to resolve.¹³

In 21st-century contracts, inserting sections about security has become the norm. In fact, security has become one of the major factors considered, as evidenced by the explosion of suggested security requirements from researchers, government agencies, industry groups and the like.

Often, when a software manufacturer upgrades its products, it informs customers that they will no longer support the obsolete software after a given date. However, in some cases, the manufacturer will agree to continue to fix any security bugs found for a longer period and may charge for doing so.¹⁴

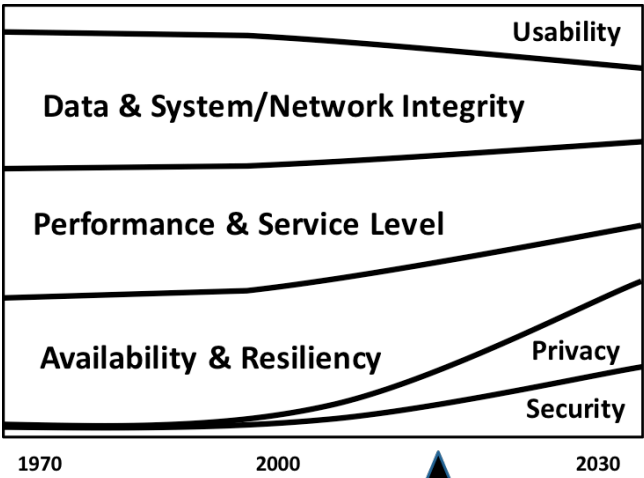


Figure 3. Increasing Proportion of Security and Privacy Sections in Contracts

Cybersecurity requirements generally fall into the following categories:

- Security triad (confidentiality, integrity, availability of data and systems).
- Privacy.
- Functionality.
- Errors and weaknesses.
- Liability or lack thereof.

Many publications in cybersecurity, supply chain assurance and vendor management treat contracts almost dismissively.

Mentions of vendor contracts and service agreements are often covered by a perfunctory reference to contracts as mechanisms to mitigate known security risks.

However, in the ISACA publication “Vendor Management Using COBIT® 5” [7], there is an entire section (chapter 5) on binding documents, which include:

- Requirements overview.
- Call for tender (RFP).
- Vendor contract.
- SLA (service level agreement).
- OLA (operating level agreement) and underpinning contracts.
- Vendor performance reports.

Nevertheless, while the vendor contract includes sections on information confidentiality and intellectual property, there is little reference to cybersecurity as such.

In chapter 6 of the ISACA report [7] “Managing a Cloud Service Provider,” there is significant focus on security and data privacy, with a section on “Security Risk and Threats Related to Operating in the Cloud.” Here we find a discussion on whether the provider or the client assumes data and application security risk. There is a very helpful list under “Risk Factors by Service Model.” Here are some examples of cybersecurity risks:

- Inadequate patch management.
- Lack of visibility of technical security measures in place.
- Insufficient virtual machine security maintenance.
- Inadequate provider authenticity.
- Vulnerabilities relating to service architecture.
- Lack of visibility into software systems development life cycle (SDLC).
- Inadequate identity and access management.

The “Overview of Threats and Mitigation Actions” section provides some slightly modified responses to these risks as they relate to contractual requirements:

- Vulnerable access management. “A contractual agreement is necessary to officially determine who is allowed to access the enterprise’s information, naming specific roles for [providers] employees and external partners.”
- Unauthorized access. As above plus “All controls protecting the enterprise’s information assets must be clearly documented in the contract agreement or SLA.”
- Virtual machine security maintenance. “Request [provider’s] internal SLA for ... vulnerability management, patch management and release management when new ... vulnerabilities are discovered. The SLA must contain detailed specifications about vulnerability classification and actions taken according to the severity level.”
- Application attacks. As above plus: “... which must align with corporate policies and procedures for similar events.”
- Collateral damage. “... include the [customer] in [provider’s] incident management process that deals with notification of collateral events ... Include contract clauses and controls to ensure that the [customer’s] contracted capacity is always available and cannot be directed to other [customers] without [written] approval.”

- Asset ownership. “Include terms in the contract with [provider] that ensure that the [customer] remains the sole legal owner of any asset migrated to the [provider, or any asset developed and paid for by the customer].”
- Asset disposal. “Include terms in the contract that require, upon contract expiration or any event ending the contract, a mandatory data wipe carried out under the [customer’s oversight].”
- Asset location. “Include terms in the service contract to restrict the moving of [customer information] assets to only those areas known to be compliant with the [customer’s] own regulation.”

There are many additional areas where formal agreement is required. Here is a more comprehensive list of requirements that should be included in software and/or a service agreement:

Risk assessments: Conduct risk assessments prior to and periodically throughout the term of an agreement. If unacceptable risks are encountered, explicit remediation and mitigation processes, as should have been detailed in the agreement, need to be activated. Their successful completion should be certified in writing.

Confidentiality: Ensure that certain categories of data – customer, client and employee personal information, intellectual property and business plans – should be protected to the extent necessary to minimize the risk of access, manipulation, destruction, etc.

Integrity: Ensure that data are not corrupted, destroyed or lost through unauthorized access, intentional or accidental misuse, software errors or hardware failures.

Availability: Ensure that authorized users and designated others (e.g., lawyers investigating evidence) have access to data and processing facilities whenever needed, subject to regular maintenance and other valid reasons for downtime.

Privacy: Meet legal and regulatory requirements for protecting and not disclosing personal data (especially financial, health and lifestyle data), whether accidentally, intentionally or through unauthorized access and use.

Functional capabilities: Ensure the system only performs allowable functions and provides authorized access to specific data. It does NOT perform any other functions, especially those that might lead to unauthorized and nefarious activities by insiders or outsiders.

Nonfunctional capabilities: Ensure that non-functional features – security, scalability, reliability, maintainability and recoverability – have been considered and that terms are included in the service-level agreement (SLA). Acceptable levels of each aspect must be stated explicitly, and remedies when the standards are not met must be spelled out.

Quality assurance: Ensure that the product or service fully meets predetermined functional and nonfunctional criteria and provide written evidence that it has done so.

User acceptance: Make sure the user community generally finds the product or service acceptable from an operational perspective. This might involve testing a prototype or final product for a customized product or service or providing a free or discounted trial period for an off-the-shelf product or service.

Performance: Determine that the product or service can meet pre-specified performance criteria, such as transaction throughput and response time, under various load situations (number of users and transactions processed per second).

Context: Frame other functional and nonfunctional requirements, such as privacy, availability, performance and resiliency, in terms of the product or service's intended operating context. Ensure any modifications for the particular context that may be required have been specified and included as part of the overall agreement.

Personnel: Specify requirements for all personnel involved in implementing and operating the product or service in terms of number, location, skills and expertise. This includes both the supplier and customer organizations. Ensure appropriate arrangements have been made for training before the software is implemented.

Security errors, weaknesses and vulnerabilities: Verify that any software, either as a product or integrated into a service, does not contain common security defects that have been listed by such entities as OWASP, SANS and MITRE. Provide written proof that any such defects have been addressed and resolved.

Liability: Specify the responsibilities of each party in advance of the agreement if the product or service fails to meet the above criteria. State the remedies and penalties (if any), particularly about direct and consequential damages.

For more information about these sections and additional areas, refer to the references and suggested reading sections at the end of this article.

Optimizing the Agreement from All Sides

Those who buy and sell software products and software-based services have their own motivations and preferences. Sellers want to minimize their liability and are reluctant to commit to damages beyond the cost of the product or service. They want to maximize long-term net income and shareholder value. If that means doing as little as possible to satisfy the terms of license and service agreements, then so be it — unless they can gain additional revenue to pay for any extra service or liability, of course.

Buyers, on the other hand, want sellers to take on as much liability as possible and fix any issues under a no-charge or minimal-cost warranty.

Clearly the goals of each party are in conflict. In such circumstances, the objective is arriving at an optimal point of agreement. This is often one where there is aggregated “least regret” on the part of all parties involved. Parties involved in the optimization process include not only the direct buyer and seller, but parent organizations, subcontractors and secondary suppliers.

It is becoming much more complex to even identify all the parties. Software design, development, and testing is outsourced, frequently offshore. Additionally, almost all software products and services contain open-source programs, for which a very different set of rules apply.

Attempting to bring all these considerations together in a set of documents can be a daunting process, and the chances of omitting important areas are high. Nevertheless, a software contract or service agreement that contemplates these factors will avoid many potential issues.

Negotiating the Contract

There are several factors influencing the outcome of a negotiation. Perhaps the most significant is the relative negotiating power of both parties, which is a function of relative size,

competition and market share. When the difference in relative size is very large (e.g., an individual or a single small company negotiating with a very large software or IT services supplier) the opportunity to achieve desired terms and conditions in contracts is minimal, if not impossible.

Only when individuals and smaller organizations (and sometimes large organizations) band together can they expect to make some headway in negotiations. Otherwise, it is a “take-it-or-leave-it” situation, where the smaller entity, be it a customer or an organization, is forced to accept the terms offered or go elsewhere — if there are alternative products and services.

Enforcing the Contract

No matter what is written in the contract, either party has the choice to pursue a particular infringement of the contract. This decision not only involves the operational and financial impact of an infringement but the relationship between buyer and seller and the litigation appetite of each.

In addition, all parties enter into an agreement with preconceived ideas about the likelihood that there will be a breach of contract — and the likelihood that the aggrieved party will take action. It often takes more than a single transgression to trigger legal action or arbitration. However, it is important for the parties to understand that arguments about enforcement may well arise.

The more specific the agreement in terms of what constitutes a failure to perform (by any party) the better. It is also important to recognize remedies that might satisfy the performance requirements before legal action might be initiated. The preferred form of resolution, legal action or arbitration, should be specified ahead of time in the contract.

Conclusions

The nature of software-related risks has changed radically over the past half-century as computer system and network technologies and their use have evolved from isolated mainframes to massively interconnected mobile devices. During the early days of computing, the number of suppliers was limited usually to just a few (e.g., IBM and AT&T).

While these large companies were difficult to negotiate contractual terms and conditions with, at least you knew who to go to if you had a problem. Today, there are so many intertwined entities involved in even the simplest of products or services that it has become a major task just to identify the players, much less negotiate contracts with them.

In this article, we have identified computer-related risks and documented their evolution as they have morphed into software cybersecurity risks. We have listed some, but not all, of the requirements and verification processes that should be specified in software contracts and service agreements. This will help you reach a point where a reasonable and necessary level of cybersecurity can be achieved.

Disclaimer: The author is not an attorney and does not intend to provide legal advice in this article. Readers should consult with their legal departments or outside lawyers to discuss which terms and conditions should be presented for negotiation with other parties and to develop appropriate contractual wording of such terms and conditions.

ABOUT THE AUTHOR



C. Warren Axelrod is a research director with the U.S. Cyber Consequences Unit. Previously, he was the business information security officer and chief privacy officer for U.S. Trust.

Warren co-founded the FS/ISAC and represented financial services in the National Information Center during the Y2K computer problem. He testified before Congress in 2001 on cybersecurity.

In 2007, Warren was honored with the prestigious Information Security Executive Luminary Leadership Award. He won a 2003 "Computerworld Premier 100" award and the 2009 Cangemi Best Article Award for an article on security metrics in the "ISACA Journal."

Warren wrote "Engineering Safe and Secure Software Systems" and "Outsourcing Information Security." He also co-edited "Enterprise Information Security and Privacy."

Warren received a Ph.D. in managerial economics from Cornell University. He received a bachelor's degree in electrical engineering and a master's degree in economics and statistics from Glasgow University. He is a CISSP and CISM.

REFERENCES

1. Lohr, S. (August 28, 2016.) "G.E., The Software Start-Up." Sunday Business Section, The New York Times. 1 & 6.
2. Axelrod, C. W. (2004.) "Outsourcing Information Security." Norwood, Massachusetts. Artech House.
3. U.S. Government Accountability Office (GAO). (March 2016.) "Vehicle Cybersecurity: DOT and Industry Have Efforts Under Way, but DOT Needs to Define Its Role in Responding to a Real-world Attack." GAO-16-350. <http://www.gao.gov/assets/680/676064.pdf>.
4. Axelrod, C. W. (September/October 2015.) "Software Security Assurance: SOUP to NUTS." *CrossTalk*, 37-43. <http://static1.1.sqspcdn.com/static/f/702523/26502340/1441086786230/201509-Axelrod.pdf?token=onHis1RGmFEG0HTIPBm1aylVhfg%3D>.
5. Axelrod, C. W. (2012.) "Engineering Safe and Secure Software Systems." Norwood, Massachusetts. Artech House.
6. Stewart, J. B. (May 5, 2016.) "Facebook Has 50 Minutes of Your Time Each Day. It Wants More." The New York Times. <http://www.nytimes.com/2016/05/06/business/facebook-bends-the-rules-of-audience-engagement-to-its-advantage.html>.
7. ISACA. (2014.) Vendor Management Using COBIT® 5. <http://www.isaca.org/knowledge-center/research/researchdeliverables/pages/vendor-management-using-cobit5.aspx>
8. Foley, J. (December 22, 2004.) "It's the End of The Road for Windows NT." *InformationWeek*. <http://www.informationweek.com/its-the-end-of-the-road-for-windows-nt/d/d-id/1029231?>
9. The author recalls following up with the claimed customers only to discover that some of them did not even do business with the vendor. Therefore, it is an important part of due diligence to check on even the most casual of claims and to make sure that the people with whom you are dealing are who they say they are.
10. Researchers frequently take the customers' viewpoint and fail to consider the needs of vendors adequately. In Axelrod [2], the author does evaluate both sides' requirements and considers how certain factors, such as the relative sizes and the industry standing of each party, affect the eventual outcome.
11. Note that customers do not actually purchase and own software products; they license them. This means that customers are not allowed to resell software even if it was originally bundled into equipment that they are selling. Sometimes the software license may be transferred to the new equipment owner, but in other cases, the new owner must license software directly from a vendor. Whereas open-source software is free, it is still subject to user agreements, which the new licensor must execute.
12. "Even very large companies have been able to make only minor changes to terms and conditions of dominant vendors' contracts. And in such cases, they might only do so under the threat of withholding payments or cancelling the acquisition, if the latter is indeed feasible."
13. If you deal with custom-built software or customized off-the-shelf software, you might find yourself hiring developers or engaging consulting firms with specific knowledge of your particular software. For purely off-the-shelf software, such expertise might not be available at any price. Having to persuade programmers to join your organization under such circumstances can be a particularly grueling experience.
14. An impartial review is often linked to who paid for the review and the reputation of the reviewing and reviewed entities. The best case is when the reviewer is funded by customers, as with the Consumers Union, and purchases software products anonymously from a retailer. While reviews vendors paid for may be objective and credible, it is natural for some bias to be introduced. Potential customers need to consider these issues.
15. The cybersecurity aspect of vehicle systems is discussed extensively in a GAO publication [3].
16. In the early days of computing, companies would worry that others might be able to get access to their customer account lists because those lists could be used by other organizations to steal customers. Organizations were generally not motivated by privacy issues to protect data.
17. Sometimes, poorly implemented security features further encourage software developers to bypass security. A recent example is the chip-enabled credit card, which appears to have a slower verification and processing system than the magnetic-stripe swipe card it replaces. Some claim that this is more perception than reality.
18. DevOps is a relatively new term that is defined as "the blending of an enterprise's applications development and systems operations teams." DevSecOps further extends this collaborative model to include information security.
19. As of May 2016, the number of Facebook users stood at 1.65 billion active users worldwide, and the average time spent on Facebook per day is 50 minutes, according to [6]
20. For details, see www.cloudsecurityalliance.org
21. As an example, tax-collecting agencies require information to be retained for specific lengths of time, which may exceed the useful lives of computer systems on which the data are stored and processed." When a computer system is replaced, such data must be converted to the format of its replacement system. If that is not possible, the former configuration must be recreated so the required data can be extracted.
22. A situation such as this arose when Microsoft notified customers that Windows NT 4.0 would no longer be supported after a December 31, 2004, deadline. As John Foley [8] wrote on December 22, 2004: "The clock is winding down on Windows NT, the operating system that propelled Microsoft into the computer server market 11 years ago. The last day Microsoft will provide general support for Windows NT Server 4.0 is Dec. 31, after which companies need custom contracts if they want the safety of Microsoft security fixes and other help."

NOTES

SUGGESTED READING

Axelrod, C. W. (September/October 2015). "Software Security Assurance: SOUP to NUTS." *Crosstalk*, 37-43. <http://static1.1.sqspcdn.com/static/f/702523/26502340/1441086786230/201509-Axelrod.pdf?token=onHis1RGmFEGOHtPBm1ayVhfg%3D>.

Department of Defense Enterprise Software Initiative. (April 2015). "Best Practices for Negotiating Cloud-Based Software Contracts—Guidance on the differences between purchasing perpetual software and renting Software as a Service." DoD ESI White Paper. <http://www.esi.mil/contentview.aspx?id=99>.

Government Accountability Office (GAO). (May 2014). "Federal Software Licenses: Better Management Needed to Achieve Significant Savings Government-Wide." <http://www.gao.gov/assets/670/663560.pdf>.

Department of Homeland Security. (May 18, 2012). "Software Assurance in Acquisition and Contract Language, Software Assurance Pocket Guide Series: Acquisition & Outsourcing." Vol. 1, Version 1.2. https://buildsecurityin.us-cert.gov/sites/default/files/AcquisitionAndContractLanguage_Pocket-GuideV1%202_05182012_PostOnline.pdf.

Department of Homeland Security. (June 16, 2009). "Software Supply Chain Risk Management & Due-Diligence, Software Assurance Pocket Guide Series: Acquisition & Outsourcing." Volume 2, Version 1.2. https://buildsecurityin.us-cert.gov/sites/default/files/DueDiligenceMWV12_01AM090909.pdf.

BITS. (January 2012). "Software Assurance Framework." <http://fsroundtable.org/wp-content/uploads/2015/05/BITSSoftwareAssurance0112.pdf>.

BITS. (2008). "Key Considerations for Managing Subcontractors." <http://fsroundtable.org/wp-content/uploads/2015/07/VMKeyConsider061908.pdf>.

Department of Homeland Security. (October 28, 2008). "Software Assurance in Acquisition: Mitigating Risks to the Enterprise: A Reference Guide for Security-Enhanced Software Acquisition and Outsourcing." https://buildsecurityin.us-cert.gov/sites/default/files/SWA_in_Acquisition_102208.pdf.

DISA Field Security Operations. (January 23, 2015). "Application Security and Development: Security Technical Implementation Guide (STIG), Version 3, Release 10." <http://iase.disa.mil/stigs/app-security/app-security/Pages/index.aspx>.

Open Web Application Security Project (OWASP). (March 2016). "OWASP Secure Software Contract Annex." The OWASP Foundation. https://www.owasp.org/index.php/OWASP_Secure_Software_Contract_Annex.

Manico, J., et al. (October 2015). "OWASP Application Security Verification Standard 3.0." The OWASP Foundation. <https://www.owasp.org/images/6/67/OWASPApplicationSecurityVerificationStandard3.0.pdf>.

Li, J. (November 12, 2009). "OWASP In Action: Tools for the DISA ASD STIG." The OWASP Foundation. https://www.owasp.org/index.php/OWASP_in_Action:_Tools_for_the_DISA_ASD_STIG.

Pelgrin, W., et al. (January 2009). "Application Security Procurement Language." SANS. <http://software-security.sans.org/appseccontract>.

Stewart, J. B. (May 6, 2016). "Facebook Time: 50 Minutes a Day." CLXV.57,224 The New York Times, B1, B8.

Office of the Comptroller of the Currency (OCC). (May 8, 2008). "Information Security: Application Security." OCC BULLETIN 2008-16. <http://www.occ.treas.gov/news-issuances/bulletins/2008/bulletin-2008-16.html>.

Federal Financial Institutions Examination Council (FFIEC). (April 2012). "IT Examination Handbook: Audit." <http://ithandbook.ffiec.gov/it-booklets/audit.aspx>.

Federal Financial Institutions Examination Council (FFIEC). (September 2016). "IT Examination Handbook: Information Security." <http://ithandbook.ffiec.gov/1136>.

Federal Financial Institutions Examination Council (FFIEC). (April 2004). "IT Examination Handbook: Development and Acquisition." <http://ithandbook.ffiec.gov/it-booklets/development-and-acquisition.aspx>.

Federal Financial Institutions Examination Council (FFIEC). (October 2012). "Federal Regulatory Agencies' Administrative Guidelines: Implementation of Interagency Programs for the Supervision of Technology Service Providers." http://ithandbook.ffiec.gov/media/153533/10-10-12_-_administrative_guidelines_sup_of_tsp.pdf.

Common Criteria Recognition Arrangement (CCRA). (September 2012). "Common Criteria for Information Technology Security Evaluation—Part 1: Introduction and General Model." Version 3.1, Revision 4. <https://www.commoncriteriaportal.org/cc/>.

CALL FOR ARTICLES



If your experience or research has produced information that could be useful to others, **CROSSTALK** can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for the areas of emphasis we are looking for:

Software Architecture

January/February 2018 Issue

Submission Deadline: Aug 10, 2017

Cyber Defense Practices

March/April 2018 Issue

Submission Deadline: Oct 10, 2017

30th Anniversary of CROSSTALK

July/August 2018 Issue

Submission Deadline: Oct 10, 2017

Please follow the Author Guidelines for **CROSSTALK**, available on the Internet at <www.crosstalkonline.org/submission-guidelines>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and Back-Talk. To see a list of themes for upcoming issues or to learn more about the types of articles we're looking for visit <www.crosstalkonline.org/theme-calendar>.

Integration of Big Data

Misconceptions, Problems, and Needed Capabilities

J. Johnson, EOIR Technologies
J. Folger, EOIR Technologies
T. Stevens, EOIR Technologies
T. Malyuta, New York City College of Technology

Abstract. The authors have been working on the problems of data integration for a number of years, in particular, integration of what is called Big Data – data defined by Gartner's famous three V's: volume, velocity and variety. Based on our experience and interactions with both customers and vendors, we discuss here the common misconceptions about Big Data integration and cloud technologies, analyze problems that need to be addressed,¹ and suggest capabilities of the solutions. The solution to which the authors contributed is described in a number of publications [1, 2, 5, 6].

Cloud Technologies and Integration Capabilities

Assuming that cloud storage technologies by themselves provide integration solutions is probably the main misconception. In general, no technology alone provides a solution to a problem; a technology can only be a part of the solution – a platform for the solution's implementation.

Yes, many cloud stores can accommodate data of different structures (they are semi-structured stores), and we can push into such a store data diverse in structure and semantics from different systems. But co-location of data is not data integration. Figure 1 schematically illustrates data about persons from different sources in the Big Table of Google – the storage solution that supports many Google applications and that gave birth to a number of similar cloud stores. Depending on the data integration requirements that are discussed below, representing data and metadata or data provenance will definitely require specific data structures and support of data management policies that go beyond what is offered by a specific storage technology.

Row Key	Column Data		
1000001	First Name: John	Last Name: Jones	DOB: 01/01/81
XYZ123	Name: John Jones	Age: 35	
1000002	Full Name: J. Jones	Country of Birth: Canada	

Figure 1. Data in the Google Big Table.

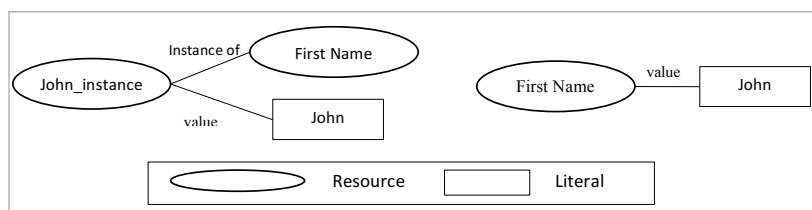


Figure 2. RDF data representation.

Most confusion is caused by RDF and OWL technologies – RDF-based languages support linkage of resources, and it is assumed that once we represent data in RDF/OWL, we immediately get an integrated store. Often, when building an integrated store, we are asked why data shouldn't just be represented in RDF/OWL. However, particular data (and metadata or provenance) can be represented in RDF/OWL in a number of ways, and in order to correctly interpret data from different sources, we need an approach to help us represent and manage all data in a particular way. In other words, we need an integration solution. Figure 2 schematically illustrates different ways of representing a fragment of data from Figure 1 in RDF; note that structural and semantic diversity of data in Figure 1 will result in structural and semantic diversity in RDF representations.

As the preceding figures show, these technologies do not provide structural homogeneity, nor do they provide semantic alignment of diverse source data. In addition to having a unified structural representation, we also need a solution to resolve semantic heterogeneity.

Moreover, an integrated solution should allow users and applications to perform traditional data management tasks. Depending on requirements, these data management tasks can be simple (e.g., retrieve specific data from diverse data collections) or more complex (e.g., perform different data manipulations with tracking provenance of data and data changes).

What is Data Integration?

In [3] we defined integration as the alignment or harmonization of multiple heterogeneous data resources in such a way that search and analysis procedures can be applied to their combined content as if they formed a single resource. For Big Data, it is also important that diverse structures and semantics of sources are not changed unless desired, because: 1) considering the volume and velocity factors, the integration process must be agile and require minimal human involvement and 2) because of the variety factor, in many cases we do not know the content of a source to be integrated or do not have time to analyze it.

What Can Co-Location Provide?

Clearly, as simple illustrations in the figures above show, co-location does not provide the capabilities we require from an integrated store. Therefore, we do not consider a store that just accommodates a number of sources without providing data and semantics management capabilities across these sources to be an integrated store. As we stated earlier, co-location is not integration. Consider the simplest possible scenario when you only need to query co-located data:

- We cannot expect it to be more than a Google-type search.
- Without special support (which, again, is provided by the integrated solution), we do not know how to formulate the request for nor how to handle the results of searches on structurally and semantically diverse data. As an example, executing a cross-source search quickly breaks down without some level of semantic alignment of sources, resulting in missed or incorrect interpretation of the results.
- For anything more than that, we will need additional capabilities not provided by the cloud storage technology (e.g., to track

data provenance, to align diverse semantics of source data, or to support required data management policies).

Depending on requirements, we need to build an integration solution where a particular chosen cloud store will be one of the technological components.

Data Integration Problems

Data integration requirements can be very different in terms of the following:

- Variety of integrated data domains and, respectively, variety of data semantics: from one or few to an unlimited number. In some cases, there is a need to integrate data from different sources about one domain (for example, data about potential customers); in other cases, data to be integrated can be from different domains (for example, intelligence data).
- Types of data manipulations on the integrated contents. In some cases, users need to analyze integrated data – data that is not modified and is not necessarily kept in the store (which requires some data management). For example, a company might need to analyze the latest competitors' sales data. In other cases, data is collected and managed; this is often the case in the intelligence community. In the case of the latter, data management requirements can be quite different as well.
- Traditional issues of the integrated store: performance, scalability, security, consistency, concurrent access, the need to normalize data (e.g., country codes, date formats), and so on.
- Issues specific to Big Data: a priori unknown structure and semantics of data, the need to quickly and properly accommodate data with minimal human effort to support necessary use cases and workflows.

Dimensions of the Solution

The solution addressing these requirements in their totality needs to be based upon the following:

- An appropriate data representation approach that allows diverse source data structures to be adequately represented in the integrated store.
- A storage platform suitable for implementing this data representation approach and meeting the performance, scalability and other requirements.
- A method of dealing with diverse semantics of sources.
- Data and semantics management policies defined by and leveraging the previous three components.

Representation Approach and Implementation Platform

The most important (and often ignored) issue is that we need to start the discussion of a solution not with the storage structures of a particular technology (e.g., triples of RDF/OWL technologies, documents/attributes of MongoDB or relations of relational databases), but with the way we want to use them for our chosen data representation. In other words, we need a conceptual approach to data (metadata and provenance) representation – an approach that meets the integration requirements and is agnostic of a particular implementation. Only with such an approach can we consider a technology that will allow

for its efficient implementation to meet such requirements as performance, scalability and security. Unfortunately, this well-established methodology to building data stores (usually based on relational databases) is neglected when using cloud storage technologies.

Semantic Alignment

We support the school of thought that a proper semantic alignment of different data models should be based on a collection of scientifically sound ontologies [3, 4, 5]. These ontologies can be applied at “arm's length” (without changes to the source data and semantics). This approach, while it does not allow for full semantic alignment, is applicable to most situations and is relatively easy and quick. To illustrate, we revisit the sources described in Figure 1 and introduce the term “Person-Name” in the alignment ontology. The subsequent assertion that ‘Name’ and ‘Full Name’ are the same as “PersonName” allows for cross model searches within the integrated store using the aligned term. Additionally, if desired, we can build a fully aligned ontology-based representation of the source data that requires transformations of data structures, semantics and values.

Main Features of Our Solution

In this paper we focused on common misconceptions and factors that must be considered when building an integration solution leveraging cloud-based technologies. Our intention was to deliberately avoid suggesting a particular data integration approach; however, we want to mention in the context of this discussion some important requirements and features that our solution is targeting (various aspects of the approach are described in detail in referenced materials). The integration approach is required to do the following:

- Accommodate data from any number of systems, each storing data about different domains in a variety of representations without changes to the semantics and structures unless desired.
- Support data accumulation, alignment of diverse semantics, and data cultivation (users and applications should be able to maintain and enrich accumulated data) in a concurrent and distributed manner.
- Maintain comprehensive metadata and provenance through the full life cycle of data.
- Support Google-style and semantic searches on data and provenance across represented data sources.
- Be scalable and performant given the volume and velocity of data.

The solution addressing all these requirements must:

- Be based on a conceptual (i.e., platform-agnostic) data and metadata representation model that has been implemented on different technologies: relational (MySQL) and Cloud (Rya).
- Support rapid data representation – the ability to represent structurally diverse data/metadata without loss or distortion with minimal human effort.
- Enable fast data utilization – the ability to query across sources in a Google-like manner immediately after data ingestion and get structured data as a result.

- Leverage arm's-length semantic alignment of data with the help of ontologies³ and, with it, the ability to meaningfully query across sources. Our representation and storage also accommodate the transformed ontological representation (when native semantics, structures, formats and values are transformed to those of the ontology) of the source data as well. Such store, however, will require time and human effort to build.
- Provide efficient and consistent comprehensive data management in a distributed and concurrent environment.

Conclusion

The paper does not describe a particular integrated solution but rather attempts to draw attention to some misconceptions about applying cloud technologies for data integration. We also tried to define with broad strokes a set of problems each integrated solution needs to address. Ignoring even one of these problems can make a solution unusable, so customers should analyze any suggested solution by considering the problems in their totality.

REFERENCES

1. Johnson, J., Folger, J., Stevens, T., Malyuta, T., Parent, K. & Brutus, J.R. (June 2016.) "Data Integration and Sharing Infrastructure on the Cloud." MSS Sensor and Information Fusion Symposium, Virginia.
2. Johnson, J., Folger, J., Stevens, T., Malyuta, T., Parent, K. (November 2014.) "The Intelligence Cloud and Distributed Management of Heterogeneous Entities." MSS Sensor and Information Fusion Symposium, Virginia.
3. Smith, B., Malyuta, T., Mandrick, W., Fu, C., Parent, K. & Patel, M. (2012.) "Horizontal Integration of Warfighter Intelligence Data: A Shared Semantic Resource for the Intelligence Community." STIDS Conference. http://stids.c4i.gmu.edu/papers/STIDS-Papers/STIDS2012_T14_SmithEtAl_HorizontalIntegrationOfWarfighterIntel.pdf.
4. Smith, B., Malyuta T., Salmen, D., Mandrick, W., Parent, K., Bardhan, S. & Johnson, J. (2012.) "Ontology for the Intelligence Analyst." Crosstalk: The Journal of Defense Software Engineering. http://www.academia.edu/2824160/Ontology_for_the_intelligence_analyst.
5. Salmen, D., Malyuta, T., Hansen, A., Cronen, S. & Smith, B. (2011.) "Integration of Intelligence Data through Semantic Enhancement." STIDS Conference. http://stids.c4i.gmu.edu/STIDS2011/papers/STIDS2011_CR_T1_SalmenEtAl.pdf.
6. Hansen, A., Salmen, D., Malyuta, T. & Antunes, N. (July 26–29, 2010.) "An Evolving Integrated DataSpace on the Cloud." MSS Sensor and Information Fusion Symposium, Las Vegas, Nevada.

NOTES

1. As we focus here on integration of data with a variety of structures and semantics that are not known a priori, we exclude from this discussion the integration solutions like traditional data warehouses where a priori known source data are transformed into specific structures, semantics, and formats that are beneficial for particular types of analytics.
2. We rely on a collection of so-called "Common Core Ontologies," based on the Basic Formal Ontology (BFO) as the upper-level ontology.

ABOUT THE AUTHORS



Mr. Jamie Johnson (corresponding author) is a software developer at EOIR Technologies. He has worked with the intelligence community for the past 13 years as a Department of Defense civilian employee and as a civilian contractor. Most recently he has worked on cloud-scale search and indexing technologies. He received a master's degree in computer engineering from Stevens Institute of Technology and a bachelor's degree in computer engineering from Rutgers University.

jjohnson@eoir.com

732-223-7413



Mr. Jacob Folger is a software developer at EOIR Technologies. He has worked as a civilian contractor for the Mission Command and intelligence communities for a combined five years. He is currently participating in the development of the Army's next-generation mission planning software, and has received his master's and bachelor's degrees in computer science from Monmouth University.

jfolger@eoir.com

732-223-7413



Mr. Trevor Stevens is a software developer at EOIR Technologies. He has spent the last five years supporting various intelligence community projects. His most recent work has focused around cloud scale search and indexing. He received his bachelor's in computer science from Rowan University.

tstevens@eoir.com

732-223-7413



Dr. Tatiana Malyuta is an associate professor of the New York College of Technology of CUNY. She is a subject matter expert in data design and data integration and has been working on data integration solutions for the Intelligence Community for the last 10 years. She received a master's degree in applied mathematics and a Ph.D. in computer science from the State Polytechnic University in Lviv, Ukraine.

tmalyuta@thedata-science.org

201-248-4332

Getting to Understand Some of the Operations and Maintenance Drivers of Cost

Jonathan W. Powell, CGFM, PMP, Security+

Abstract. The overall theme for this issue is Operations and Maintenance (O&M). This article will look at specific cost drivers, including licensing, design and documentation, replenishment planning, data collection, and impacts of the operational environment. It will then look at tools to reduce cost and improve availability. It will close with a look at the younger generation and discuss whether a tour of duty or two in O&M would help their overall career growth and effectiveness as engineers.

O&M comprises about 80 percent of system cost, yet many analyses during system acquisition ignore this phase. So if one is to look at O&M, some questions he or she might ask are:

How is the architecture influenced by license cost?

Architecture should be informed by licensing cost, in addition to other costs (such as the cost of maintenance). Over the last couple of decades, appropriate emphasis has been given to this category of cost. This was the period when Linux and the whole open-source construct came to the fore as a means to help tamp down escalating software development costs. Licensing costs pervade every aspect of architecture, from the number of tiers used for a web-based system to the makeup of the databases to transaction processes to screen displays. The rubric with respect to architecture and maintenance costs is fairly straightforward:

- 1. The more complicated it is, the higher the licensing costs.
- 2. Open source is cheaper than its alternatives.

In the federal government context, software development is most often constrained by budget. This means there's a ceiling on design costs. Since the makeup of the architecture directly impacts design costs, and licensing is an outcome of the architecture, these costs are subject to budgetary pressures. Therefore, if you're under extreme budgetary pressure, you could seek to simplify the architecture (for example, develop a single tier rather than multiple tiers).

The point is this: The right answer should always be to make the architecture as simple as possible for its intended purpose. But this is challenging and contradicts contractor self-interest, which seeks to maximize revenue by driving up complexity and, hence, costs.

Most often, when budget is an issue, the easy answer for a project manager is to default to open source components for

the architecture. Contractors invariably will argue against open source, saying it's not as robust or cannot be used for mission applications, but the government should push back. These attacks are often empty, used in an attempt to drive up revenue. They should be beaten back by the program management team.

In a perfect world, you would lay out all of the architectural options available to meet a particular set of design objectives and provide a cost-benefit assessment (CBA) for each of these options in order to apprise decision-makers of all the options and tradeoffs on the table. The CBA should be nuanced to fully account for impacts on the mission. For example, rather than using a straight tally of benefits versus costs, the benefits could be given weighted percentages of the design objectives met, ensuring all key performance parameters (KPPs) are satisfied. If you include the payback period, you can derive a matrix as shown in the following figure:

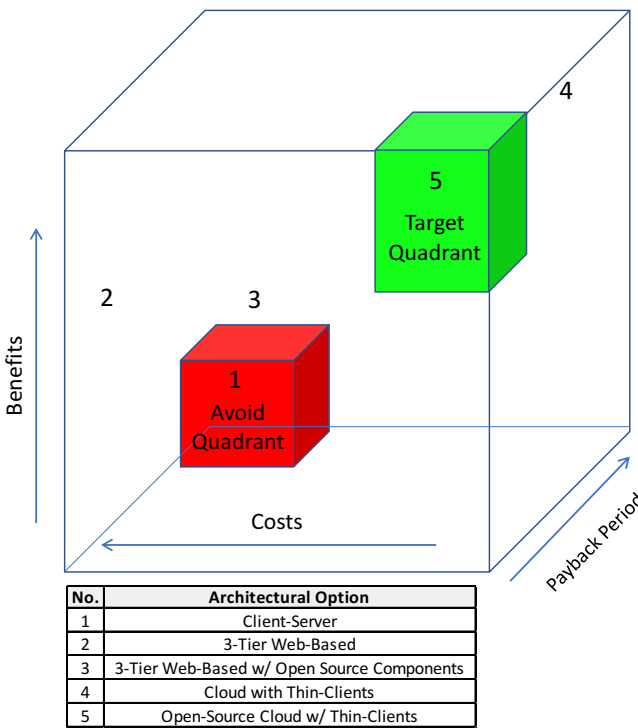


Figure 1. Example Cost-Benefit Assessment Matrix Including Payback Period Analysis

But this is rarely done, and when it is, it's often glossed over or poorly performed. Even where a CBA is required, it's often just reuse of previously submitted versions and rarely brings innovation and critical thinking to the task of sorting through the art of the possible. In the current era, when more attention is being paid to getting the requirement right the first time so the acquisition design and development are correct coming out of the chute, and when programs are positioned for success, more emphasis should be paid to this area. Or, for less complex procurements, perhaps providing a scaled-down version of the analysis for decision-makers should be considered.

How does software design and documentation influence maintenance cost?

Software design influences maintenance cost because it determines how the system will be built. A complicated design rarely leads to an easily maintainable system. In fact, overcomplicating system design is a tactic used by software developers to help drive up cost (and hence revenue) while also locking in a competitive advantage. A company can make the design of a system more complicated than it needs to be, or make it so complex that there's a learning curve built into the system such that anyone performing maintenance during O&M needs a significant period of time to learn about the system before they're productive, driving up O&M costs dramatically. I experienced this on the army software program I led for eight years. Prior to this, the program had been in existence for seven years without software being fielded. That had been enough to get the incumbent fired, but enough of a complicated design had been implemented by then. Not only were they able to lock in 40 percent work share as a subcontractor on the follow-on contract because of the inherent knowledge they had of the system's complexities, but they also forever cursed the system to a pattern of higher-than-normal maintenance costs because the byzantine architecture created could only be divined by a large number of senior and seasoned C# and PL/SQL development specialists, who were much more expensive than their more junior-level counterparts.

Most of the cost in typical systems is O&M, and therefore there is a need for documentation to help ensure the O&M is performed as effectively and efficiently as possible. This is especially true if the customer seeks to enhance their flexibility and enable vendors to compete for the maintenance work. A solid technical documentation package can help with this and help a new vendor to come up to speed quicker on the O&M of a system rather than coming in cold, thus helping reduce transition costs. The problem is that software development invariably falls behind schedule — refer to the Standish reports for data on this. And when development falls behind, documentation and training are usually the first items to go in an effort to recover schedule. The corollary to this is when documentation is overemphasized and too much investment is made in documentation. No documentation package is perfect and can fully inform staff members of all the ins and outs of performing system O&M, especially on complicated systems. In most cases, if the documentation can act as a guide and a useful reference, that will suffice. Investing in documentation beyond this is pointless as it will end up sitting on the shelf unused. The other aspect of documentation that should be carefully watched is documentation as part of O&M. How time-consuming is the paperwork regime for configuration control? Implementing bug fixes and patches? An onerous paperwork regime can have a significant adverse impact on the cost of maintaining a system, especially if every little change needs reams of paperwork to get approved and authorized. Let program managers beware.

How does the replenishment plan influence hardware cost?

Again, it depends on the mission and budgets, but in general, a replenishment plan with frequent updates will lead to increased hardware costs. For example, if the application is mission-critical and data processing intensive, the replenishment plan might call for frequent 'tech refreshes' to take advantage of increasing price performance and gains in processing power, as well as software updates designed to take advantage of these gains. However, the converse can be true as well — legacy hardware can be left in place for years without a replenishment plan and can function just fine if it meets the needs of the mission. In these cases, when the time comes to perform a major refresh of this system, developers often choose to start fresh and replace the system outright rather than attempt to upgrade the legacy infrastructure with new hardware components.

How does data collected during operations influence future service level agreements (SLAs), and sparing?

In theory, it should enhance future SLAs because the underlying detail should be available to objectively evaluate performance and implement both "carrots" and "sticks" as appropriate. Also, increased data should improve the effectiveness and efficiency of sparing operations. Assuming the right sort of data is collected, it could be analyzed to yield information that reduces duplication and helps get the right part to the right place faster, all of which leads to reduced cost. However, the underappreciated part of this equation is the cost of collecting data. The program team should assess this carefully. Collecting data for data's sake is not only costly but of minimal value. To ascertain what data needs to be collected and how this information will be used is key. A cost-benefit analysis should be done to show how the data will be transformed into useful, actionable information such that the relative benefits of the increased data collection outweigh the costs.

How do we account for the operational environment in estimating the O&M cost?

This is the whole area dealing with unknowns. You can plan all you want with respect to operations, but at the end of the day, stuff happens, and it's often stuff that hasn't been encountered before. So, there are two parts to accounting for the operational environment in estimating O&M costs. First, plan, plan and plan again. In the submarine force, we use lessons learned. In the Army, it's AARs (after action reviews). Whatever the methodology, take the collective body of evidence and build out planning based on situations that have occurred. These should be prioritized based on probability and impact, and response plans should be built and budgeted for so that the O&M team can rely on them to execute when the time comes. The impact to O&M cost for these plans includes the cost of recurring training on the plans so personnel are familiar with them and they remain fresh and can be executed effectively when needed, as well as any surge costs the plans provide for — for example, emergency personnel and equipment brought in to deal with certain scenarios. As far as the unknowns go, these should be budgeted for as well. This fund is often referred to as "management

reserve,” and the leadership should put into place some buffer they can call upon if unexpected situations are encountered in the operational environment the O&M team must respond to.

What tools should be used during operations to reduce cost, improve availability, or both?

The most important tool is a corporate culture of free-flowing communications. It's not the higher-ups who are going to know about the issues; it's the operators on the ground who are encountering them. In order to address these issues, appropriate channels need to be in place so information can be readily captured. A system must be in place so issues can be quickly adjudicated and addressed. Emphasis should be placed on ensuring the communications channels are not overly burdensome (such as having to fill out long forms to document the points). These communications channels can have the added benefit of capturing staff members' good ideas and innovations. The best way to improve availability and reduce cost are finding ways to most effectively and efficiently capture the good ideas and suggestions of those performing line operations on a daily basis. The challenge for leadership is establishing an open environment where these innovations can be shared and brought forward. The hardware or software tools for capturing these ideas are not important, as long as they make it easy for the operator to convey the information. For example, it could be as simple as a Sharepoint portal where operators can enter the information — either publicly or anonymously — and begin a workflow to get the ideas known and visible to leadership. Remember, the most important thing here is the corporate commitment to capturing good ideas and then acting upon them.

Should bright young engineers be assigned to O&M for educational purposes?

Absolutely. It's vital that young engineers understand the full life cycle of products and software. It's especially important because they'll get to see how decisions made “way back in design and development” impact the lives of maintainers during O&M. This will give them an appreciation for the effects of these decisions. It will also encourage them to be proactive in addressing downstream maintenance impacts when they lead and participate in design and development later.

ABOUT THE AUTHOR



Jonathan W. Powell, CGFM, PMP, Security+, is an adjunct professor at UMBC. A former submarine officer, Mr. Powell has led complex engagements for military, federal and intelligence agencies. His articles have also been published in “PM Network” and “Contract Management.” Mr. Powell serves on the board of the Montgomery County Revenue Authority and is a member of the State of Maryland's Cybersecurity Council. A graduate of the FBI's Citizens Academy, he holds a B.S. from USNA and an MBA from the University of Maryland.

**13205 Dodie Dr.
Gaithersburg, MD 20878
240-418-8588
jopowell@umbc.edu
201-248-4332**



Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

2017 12th International Conference on Computer Science and Education (ICCSE)

22-25 August 2017
Houston, TX
www.ieee-iccse.org

26th International Conference on Computer Communication and Networks (ICCCN)

31 July – 3 August 2017
Vancouver, BC, Canada
www.icccn.org/icccn17

2017 39th Electrical Overstress Electrostatic Discharge Symposium (EOS/ESD)

10-14 September 2017
Tucson, AZ
www.esda.org/events/symposia

MCSoc – 2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip

18 Sep 2017 – 20 Sep 2017
Seoul, South Korea
<http://www.mcsoc-forum.org/>

CT – CONNECT.TECH

20-22 Sep 2017
Atlanta, GA
<http://connect.tech>

MODELS – 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems

24-29 Sept 2017
Austin, TX
<http://www.modelsconference.org/>

28th Annual IEEE Software Technology Conference

25-28 September 2017
Gaithersburg, MD
www.ieee.stc.org

SiPS – 2017 IEEE International Workshop on Signal Processing Systems

3-5 Oct 2017
Lorient, France
<http://www.sips2017.org>

HLDVT – 2017 IEEE International High Level Design Validation and Test Workshop

6-7 Oct 2017
Santa Cruz, CA
<http://www.hldvt.org>

It's Just a Model!

If you are not familiar with the quote "It's just a model!" it comes from the movie *Monty Python and the Holy Grail*. Patsy is a character in the film played by Terry Gilliam. Serving as King Arthur's assistant, he only has a few simple tasks throughout the entire film, such as using two halves of a coconut to simulate the hoof beats of Arthur's nonexistent horse. Patsy has only one line in the film: "It's only a model," said when the Knights of the Round Table first catch sight of the castle at Camelot. And indeed, it **is** only a model. As a matter of fact, if you wish to see the dance routine *Knights of the Round Table/Camelot Song* played by Lego characters, I suggest you Google "knights of the round table Legos" and have a good laugh.

"It's just a model"? Models are a critical part of what we do for a living. From 2003 through 2009, I supported work on the Airborne Laser while I was employed working for a modeling and simulation company named AEgis Technologies. AEgis Technologies has a lot of people who really understood the modeling and simulation business. In modeling and simulation, you worry a lot about errors. Both Modeling and Simulating can produce type I and type II errors, just as any other branch of statistics. Quickly put, a type I error is when the model and simulation shows that something will not work, but it actually will work in the real world (for example, the model and simulation shows an aircraft cannot safely clear a mountain, but in the real world it could have succeeded). A type II error is the other way around, your simulation shows something will work, but it doesn't in real life (for example,

your simulation shows you can easily clear mountain on takeoff, but in reality, you crash). When you make this kind of error, especially in the avionics industry, millions of dollars may be wasted, and people may die.

There's also a type III error, although this is more in the field of verification and validation. A type III error is when you solved the wrong problem. In a type III error, for example, given the problem of "can an airplane make it over a mountain top", you instead show that a helicopter could. You have solved the problem based upon an invalid interpretation of requirements. Your model is wrong. This type III error is not really a type of modeling and simulation error, it is an error in **verification** (is the problem being solved in a correct way) and **validation** (are you solving a correct problem). You see, a model must be developed in a correct manner (correctly **abstracting** and **encapsulating** behavior from the real world) and it also must be a valid model, the model of what you're trying to use as a pattern. For example, if you're trying to model behavior of a Boeing 747, but you capture the flight characteristics of a small Cessna, it doesn't matter how good the model is, it is the wrong model and cannot be used to solve your problem. The initial assumptions or requirements are wrong, so your model will be wrong. Yeah oh okay is all this included and on.

To paraphrase Shakespeare: "Requirements? Ay, there's the rub!" It's all about requirements. That's where all our problems start, don't they?



What does it take to build a good model? I guess that depends on what you're doing. Which leads us to the real topic of this column.

#1 – Modeling is a tricky business. Unless you're comfortable tossing around words like encapsulation, abstraction, verification, validation, coupling, cohesion, inheritance, polymorphism, multiple inheritance, and overloading – you should not be developing the models. Oh sure, you can help, but you lack the tools necessary to do it right. Leave it to the experts to build a model, and you need to be content just using the model.

#2 – Using the model requires great familiarity with the model. Models and frameworks are seldom trivial. You should be reasonably smart to use the right model and you have to be very smart to use the model right.

#3 – There is no perfect model. There're lots of models that will work, and there're also a huge number that won't work. You should find the one that works best for you, and understand that if it meets 80% of your needs, it's probably good enough. You can spend all your time and budget trying to find the perfect model. It doesn't exist. This is probably related to Pareto's Principle, the "80/20 Principle". There are a lot of models out there that will not work at all. There are some that are usable. Sometimes it's better to find a usable one quickly instead of waiting for the perfect one. Remember the one that is perfect for you will probably not be perfect for anyone else – and the savings from reuse will be decreased.

#4 – Be wary of models that claim to produce executable code. Remember that the code produced by a model is only as good as the model is. You need people trained in both modeling and the use of the model-producing tool to get reliable results.

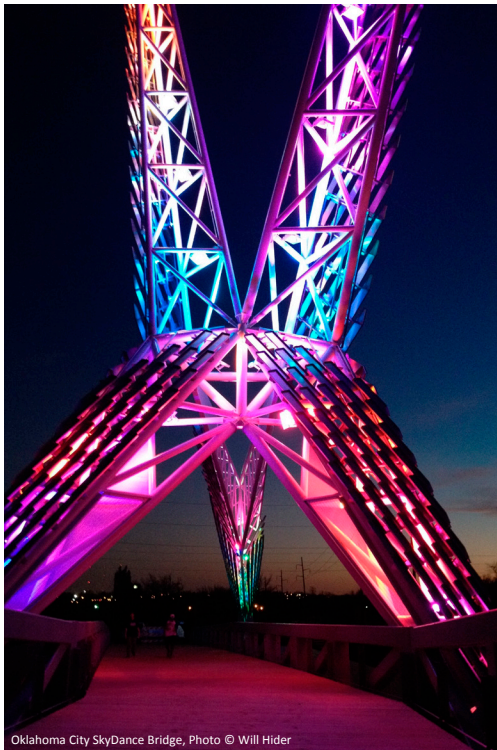
#5 – Using model-based development and model-based testing is hard. Don't expect to buy the tool in January, have a one week class in February, and have reliable test cases flying out of the tool by March.

I close with the following anecdote: Back in the 1940s, supposedly a Texas Aggie was looking for work over the summer. He was an engineer and looking to apply his talent. He asked around, and found out there was a great need for outhouses that had little odor. Using the highest quality cedar wood, and his knowledge of airflow and ventilation, he built a model of a "sweet-smelling outhouse". After he completed the model, he would travel around South Texas, showing the model and installing hundreds of his "sweet smelling outhouses". Over the summer, he made quite a lot of money. The next summer, he decided to go back into the same business. However, upon driving back to South Texas, he was confronted by many former customers who complained his outhouses did not work. He decided to investigate, and went to one of the first outhouses he ever installed. He came back out a few seconds later, and said "Well, of course it stinks! Do you have any idea what somebody did in there?"

It's all about understanding your requirements, isn't it?

David Cook, Ph.D., Professor of Computer Science
Stephen F. Austin State University
cookda@sfasu.edu





Oklahoma City SkyDance Bridge, Photo © Will Hider

WE ARE HIRING

ELECTRICAL ENGINEERS AND COMPUTER SCIENTISTS

As the largest engineering organization on Tinker Air Force Base, the 76th Software Maintenance Group provides software, hardware, and engineering support solutions on a variety of Air Force platforms and weapon systems. Join our growing team of engineers and scientists!

BENEFITS INCLUDE:

- Job security
- Potential for career growth
- Paid leave including federal holidays
- Competitive health care plans
- Matching retirement fund (401K)
- Life insurance plans
- Tuition assistance
- Paid time for fitness activities

Tinker AFB is only 15 minutes away from downtown OKC, home of the OKC Thunder, and a wide array of dining, shopping, historical, and cultural attractions.



Send resumes to:
76SMXG.Tinker.Careers@us.af.mil
US citizenship required

NAV  AIR



CROSSTALK thanks the above organizations for providing their support.