A Common Sense Way to Make the Business Case for Software Assurance

Antonio Drommi
Dan Shoemaker
Jeff Ingalsbe
John Bailey
Nancy Mead

ABSTRACT: This article demonstrates how a true cost/benefit for secure software can be derived using three generic practice areas: (1) threat/risk understanding, (2) implementation of security requirements, and (3) operational security testing. Having an accurate cost for these aspects of the software assurance process would allow decision makers to make intelligent decisions about the level of investment they wish to make.

WHY WE NEED TO DISTINGUISH SOFTWARE DEVELOPMENT FROM SOFTWARE ASSURANCE

The aim of this article is to demonstrate how a common valuation model can be used to make a dollars and cents business case for software assurance. However, in order to do that, it is first necessary to talk about why the elements of software assurance cost have to be differentiated from those of traditional software development.

A precise delineation of the cost elements of secure software assurance is required because the total cost of anything is the sum of the costs of its parts. And unfortunately, there is no commonly agreed-on line of demarcation between the activities that constitute software assurance and those associated with producing a correct product.

It should be apparent that the cost of producing the product should be different from the cost required to make sure that the product is secure. Yet when it comes time to assign the actual cost associated with each process, the distinction between product quality assurance and product security gets lost.

Profit margins drive most business decisions. That is why it is so dangerous to over-inflate the price of secure software. Price inflation happens because businesses tend to confuse the costs required to ensure against exploitation with the much greater costs of producing a correct product. The fact that defects are a given in software does not change the ethical obligation of the maker to produce

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

functionally correct products. So the practices associated with delivering satisfactory and bug free code need to be differentiated from the security practices.

Formal practices to ensure that the code was built correctly date back at least forty years [Royce 1970]. So if the aim of those practices was not to ensure that the code was secure, what were we trying to ensure? The answer is the quality of the software product. And in service to that aim, practices to ensure product correctness have become an integral part of the software process. Likewise, because of the long history of SQA, there is a seemingly endless list of standard quality assurance practices, which have always been considered essential to the software life cycle but not part of security. Activities such as design practices (1972), inspections (1976), change management (1979), incremental integration (1979), branch coverage testing (1979), and source code control (1980), which predate today's interest in security by decades, are often included in the steps for developing secure code [McConnell 2005].

These activities represent a cost of producing software that satisfies the professional requirements of a properly developed product. So from a costing standpoint, the marginal increase in cost for secure software assurance is not the assurance of completeness and correctness. Instead the cost of more secure software is the additional cost incurred to guarantee the code's freedom from exploitation and harm by any adversary.

Given this critical distinction, the costs that are a traditional responsibility of software development have to be partitioned from the costs of software assurance in order to obtain the true cost of the assurance part. Moreover, if that partitioning is not done, costs that should be inherent to the delivery of a quality product will serve to ratchet up the apparent cost of software assurance to a level that will be hard to justify in a business sense.

It might seem cynical to suggest that businesses would make decisions about their security based on a profit motive. But since cyber disasters are abstract occurrences—until they actually happen—and every business wants to maximize its profit margins, it makes sense to insist that the investment in security should be justified by the business case. That is why getting a cost for secure software assurance that is not inflated by inappropriate cost factors is so important to the overall argument for national security.

Software Assurance: A Definition

The reason it is so important to identify the right set of common software assurance practices is because the costing process is always built on assumptions. And from the standpoint of the business case, a properly executed software engineering process should always produce correct and properly functioning code. Consequently, the requirement to be a little more careful in the specification, design, coding, and testing of the software should rightly be part of the business case for developing the software in the first place, not the assurance case.

Similarly, any cost to rework defects incurred as a result of the build should also not be charged to the assurance case. The fact that defects are a given in commercial software does not change the ethical obligation of the maker to produce bug-free code. This reasoning, therefore, eliminates all of the traditional quality assurance practices from the costing equation for software assurance. The question then, is "What is left"?

According to the U. S. Committee on National Security Systems' National Information Assurance (IA) Glossary, software assurance is "the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner" [CNSS 2006, pg. 56]. That definition suggests that software assurance involves the execution of a well-defined process to ensure that the code is not exploitable and that everyday software development activity follows good security practice. In this paper, we will focus on the cost model for the activities needed to assure that software is free from vulnerabilities, along with the certification activities, such as those specified in the Common Criteria. We will not focus on the ordinary verification activities that are needed to ensure correctness.

Another definition suggests that one of the aims of the assurance process is to build software-based systems that "limit the damage resulting from any failures caused by attack-triggered faults, ensuring that the effects of any attack are not propagated, and recover as quickly as possible from those failures as well as ensuring that the software will continue to operate correctly in the presence of most attacks by resisting the exploitation of weaknesses in the software by the attacker, or by tolerating the failures that result from such exploits" [Redwine 2007].

A report out of the USC Center for Systems and Software Engineering (CSSE) Costing Secure Systems Workshop says that the true cost of software assurance amounts to the cost of additional concrete functionality to secure the product [Colbert 2006]. That includes the complete set of additional development activi-

ties from specification through design and on to code that are aimed at ensuring security functionality.

Along with the cost of additional functionality, any additional cost to certify and accredit the software as secure under the Common Criteria is also a cost of software assurance [Colbert 2006]. So if you adopt the CSSE definitions, the additional cost of secure software assurance is the cost associated with the definition and implementation of security functional requirements and security assurance requirements and the effort involved in additional documentation and activities such as the definition of security roles and certification [Colbert 2006].

Finally and more importantly, ISO/IEC-JTC1 [Moore 2007] is in the process of revising ISO 15026, Systems and Software Integrity Levels [ISO 1998], to encompass a better definition of the software assurance process. Rather than being a single standard, the outcome will be a multipart standard that will describe terminology, assurance case planning, system integrity levels (from the 1998 version), and standard best practices for life cycle assurance. The final part (4) has relevance to our discussion here because it relates three specific analytic activities to the assurance case: requirements analysis, architectural design, and risk management. The latter two activities are specifically driven by the integrity level requirements.

THE ROLE OF THE SOFTWARE ASSURANCE PROCESS

The outcome of these initiatives would seem to indicate that an important role of the software assurance process is to field software-based systems that limit the damage resulting from any failures caused by attack-triggered faults. The objective is to ensure that defects that represent vulnerabilities are identified, that the effects of any subsequent attack are not propagated, and that the system recovers as quickly as possible from those failures. In effect the system continues to operate correctly in the presence of most attacks by either resisting the exploitation of weaknesses in the software by the attacker or by tolerating the failures that result from such exploits.

If that is the case, then the cost elements associated with limiting vulnerabilities in assured software are just those additional processes and activities that seek to identify and remediate any identifiable means of exploitation, as well as the additional cost needed to build real-time survivability and recovery into the product.

As a consequence, the business case associated with limiting vulnerabilities in assured software should be primarily built around costs associated with identifying exploitable vulnerabilities that result from improper specification, design, and coding of software, as well as the development of specific security functionality that will actively work to minimize damage from any unforeseen attack and ensure a systematic recovery.

Generic Activities that Might Be on the List

In order to cost properly, it is necessary to have a concrete list of activities to base the actual outlay of resources on. Given the prior discussion, we have preliminarily identified three generic types of activity:

- threat/risk identification and understanding, which is part of the proposed 15026 revisions but not part of the USC model
- security functional requirements/security assurance requirements, which are part of both 15026 and the USC model
- operational security testing throughout the life cycle, which is also part of both the 15206 and USC approaches

We have added threat/risk assessment to the set of recommended practices for secure software because it is clear that risk analysis and management will be a significant element of the international standard. That addition makes sense, since in practical terms there is no security without a thorough understanding of the security risks. Therefore, in effect the shape of the functional and procedural response is always structured to meet the exact form of the potential risk. Without an adequate understanding of the conceivable ways that a given piece of code can be compromised, it is impossible to develop a secure architecture.

Also, a proper understanding of the risks allows the developer to concentrate the security functionality and assurance on just those risks that are known to be viable. The classic means of identifying and characterizing risks to software is through risk assessment. There are a number of risk assessment approaches in use; however, the Microsoft threat modeling approach is popular in industry. At a practical minimum, risk/threat understanding encompasses the identification and evaluation of the actual risks/threats identified by the risk assessment/threat model.

From a process standpoint, the specification of software assurance requirements only requires knowledge of the risks involved. However, design requires a more complete understanding of the likelihood and impact of each risk. Thus, risk understanding applies in two places in the life cycle, in the development of the security requirements and in the detailed design process. There are a number of

requirements engineering processes that can be applied to develop security requirements. The understanding of risk at that level can also be driven by architectural risk analysis results, abuse and misuse cases, and attack patterns [McGraw 2004].

Once the risks are sufficiently understood, the next logical step involves the development and validation of the functional and non-functional security requirements. Ideally these flow logically and directly from the understanding gained by the risk assessment exercise. The actual software requirements specification always encompasses a much larger set of requirements. From a costing perspective, however, the only costs that are assignable to assurance are those associated with building functions necessary to ensure the general robustness and security of the software.

Operational security testing takes place after the software is built and tested for conventional correctness. It is, in effect, a form of pre-release testing. It primarily involves ethical hacking, but it can also involve formally scheduled penetration tests. The former activity should identify all electronic vulnerabilities, while the latter will identify vulnerabilities arising from other factors such as human-based threats. The aim is to identify points of vulnerability that have either not been identified in the risk/threat understanding part of the process, or flaws that might have been injected in the construction process. The idea of this activity is to maintain the software assurance function as a living entity, and the cost of doing that is directly assignable to assurance. The life cycle model that this produces is shown in Figure 1.

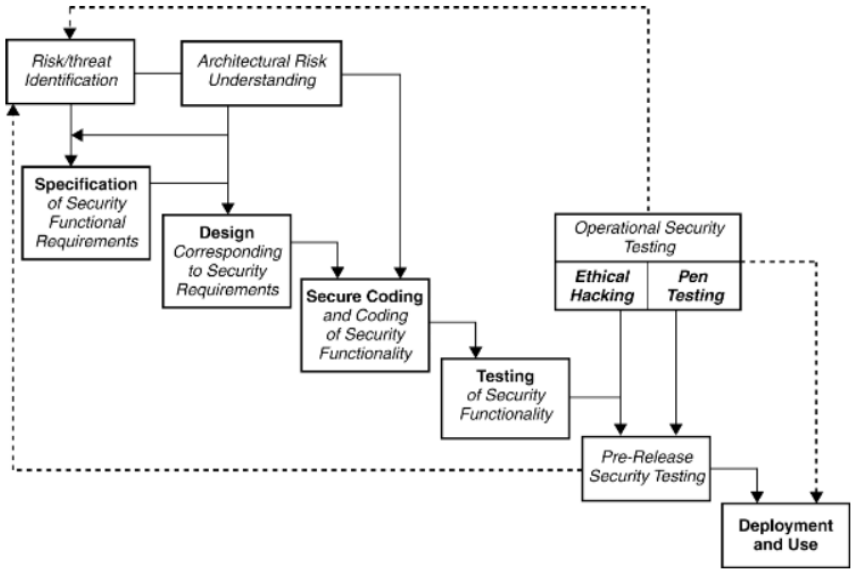

Figure 1: Life Cycle Model

The software assurance activities are in italics. These comprise the only aspects of the development process that are assignable to assurance under our assumptions. They function together to ensure that (a) all potential risks are identified and the proper response incorporated into the solution at the beginning of the process and (b) the robustness of the software is proven at the end of the process.

Finally, there is also a general management cost component that is spread over all projects throughout the life of the software assurance process. For example, identifying risks to assets and understanding the likelihood and consequences of the realization of those risks requires that the organization already have in place an agreed-on method for assessing likelihood and quantifying consequences. This must be established before the organization runs the first project through the assurance process.

Furthermore, the organization has to establish a formal decision-making structure that defines who should make the call as to whether to accept, transfer, or mitigate risks based on those specific estimates of likelihood and consequence. Finally, there needs to be an auditing function that ensures that the "formulas" for likelihood and consequences are applied correctly across all projects.

Clearly, the establishment of the software assurance management infrastructure is a cost that is a precondition to the execution of the rest of the process. It is a sunk cost, which enables the overall software assurance operation. To obtain a true cost of the process, it has to be assigned somewhere. Yet common sense would dictate that the first project to the table should not bear all of the cost of

the place settings that subsequently will be used by everybody else. Therefore there has to be a formula built into the overall costing method that apportions the cost of creating and maintaining the management function into the overall cost assignment process.

EXAMPLE: BUILDING A BUSINESS CASE FOR SOFTWARE ASSURANCE

In our initial article, Models for Assessing the Cost and Value of Software Assurance, we identified 14 commonly accepted models for valuation of IT assets:

Investment Oriented Models

1. Total Value of Opportunity (TVO) – Gartner
2. Total Economic Impact (TEI) – Forrester
3. Rapid Economic Justification (REJ) – Microsoft

Cost Oriented Models

1. Economic Value Added (EVA) – Stern Stewart & Co
2. Economic Value Sourced (EVS) – Cawly – Meta Group
3. Total Cost of Ownership (TCO) – Gartner

Environmental/Contextual Models

1. Balanced Scorecard – Kaplan and Norton
2. Customer index – Andersen Consulting
3. Information Economics (IE) – The Beta Group
4. Portfolio Management – Rubin (Meta Group)
5. IT Scorecard - Bitterman – IT Performance Management Group

Quantitative Estimation Models

1. Real Options Valuation (ROV)
2. Applied Information Economics (AIE) – Hubbard
3. CoCoMo II and Security Extensions – Center for Software Engineering

One of the most widely used of these is the balanced scorecard [Kaplan 1992]. We will therefore use that approach to illustrate how a business case might be built for software assurance under the assumptions we have just talked about.

The balanced scorecard is particularly applicable to this kind of costing because it is designed to let companies assess the performance of a target process against

factors that are outside of the scope of traditional time and effort costing. At its core, the scorecard seeks to equate the standard financial indicators of performance with more fluid indicators such as customer relationship, operational excellence, and the organization's ability to learn and improve [Kaplan 1992]. Table 1 summarizes these categories and gives some examples of ways they can be measured.

Table 1: Categories of measures for four perspectives of the balanced score card

Financial Perspective Measures	**Customer Perspective Measures**
• Net Income • Operating Margin • Economic Value Added • Revenue Growth	• Customer Satisfaction, External • Customer Satisfaction, Internal • Customer Loyalty
Operational Perspective Measures	**Learn/Grow Perspective Measures**
• Safety • Process Enhancement • Operational Efficiency • Productivity	• Employee Personal Development • Employee Satisfaction • Organizational Enhancement

We could envision additional measures, such as customer trust; however, Table 1 illustrates the balanced scorecard approach as described by the original authors. The balanced scorecard requires the use of metrics that have been specifically customized to an individual organization's particular environment. Generally there are three types of metrics involved. The first are the metrics used to describe internal technical functions. Examples of these are reliability and defect rate. These measures are not particularly useful to non-technical managers and upper level policy makers, but they are objective and easy to aggregate into information elements that can help technical managers assign value to the security aspect of the IT function.

The second category contains measures that form the ingredients of comparisons or "report cards." These are intended for use by senior executives. For example, if software assurance is designated a cost center, the goal is to show how those costs have changed over time, or to assess how they compare with costs in similar companies. Examples of measures in this area include operation costs broken out on a per-user basis.

The final category includes metrics that are intended for use by the business side. These can include such things as utilization analyses and cost and budget projections. These measures allow an organization to estimate the business impact of a given activity. This evaluation is absolutely essential if it wants be able to prioritize the level of resources it is willing to commit to the assurance process for each item of its operational software.

Table 2 contains a non-exhaustive list of potential measures that might apply to a costing process for software assurance for each scorecard category. Using a tailored scorecard composed of standard metrics, the organization can then assign a quantitative value for each of its software assurance activities. And it can track the effectiveness of those activities using data obtained through one (or all) of these categories.

Table 2: Samples set of measures for assigning value to software assurance

Financial Perspective Measures	**Customer Perspective Measures**
Measures of Technical Value • Risk assessment/threat model activity cost • Operational test activity cost • Operational test defect removal cost • Risk assessment/threat model defect removal cost Measures of Comparative Value • TCO for software assurance activities • Net overall income increase over time • Net overall revenue per unit Measures of Business Value • Economic value added for SwA • Increase/decrease in cost per unit • Cash flow expressed as P&L	Measures of Technical Value • Customer initiated change • User and focus group feedback • Problem resolution reports Measures of Comparative Value • Risk assessment/threat model action items • Pen test benchmarking data • CC capability certifications Measures of Business Value • Cost of customer initiated change • Cost to address unmet requirements • Cost of customer initiated work
Operational Perspective Measures	**Learn/Grow Perspective Measures**
Measures of Technical Value • Exploitable defect identification rate • Exploitable defect removal percent • Exploitable defect density • Percent security requirements satisfied • Change requests tied to incidents • Operational resilience Measures of Comparative Value • Performance against budget • Productivity • Mean time to incident Measures of Business Value • Marginal increase/decrease in incidents • Marginal change in cost of incident • Risk assessment cost estimates • Growth in personnel capability	Measures of Technical Value • Assessed employee security capability • SwA training/development activities • Personal certifications (e.g., CISSP) Measures of Comparative Value • Growth in satisfaction with process • Growth in assessed security process • Growth in security/safety performance Measures of Business Value • Training performance indices • Employee performance indices • Security certifications per unit

It is possible to construct a meaningful business case for security from data elements such as these. For example, the organization could begin collection of data

on all four perspectives, which it could equate to relevant indices. A sample set of these follows.

Financial Perspective Measures

Measures of Technical Value

- Risk assessment/threat model activity cost versus exploitable defect identification rate
- Risk assessment/threat model activity cost versus percent security requirements satisfied
- Operational test defect removal cost versus mean time to incident
- Risk assessment/threat model defect removal cost versus customer initiated change cost
- Exploitable defect removal percent versus productivity (e.g., in LOC)
- Exploitable defect density versus marginal increase/decrease in incidents
- Change requests tied to incidents versus performance against budget

Measures of Comparative Value

- TCO for software assurance activities versus net overall income increase over time
- Economic value added for SwA versus net overall revenue per unit
- Risk assessment/threat model action items versus cost to address unmet requirements
- Pen test benchmarking data versus cost of customer initiated change
- Assessed employee security capability versus growth in satisfaction with process
- CC capability certifications as a measure of growth in assessed security process

Measures of Business Value

- Operational test activity cost versus increase/decrease in cost per unit
- Growth in personnel capability versus marginal change in cost of incident
- Training performance indices versus cost of customer initiated rework
- SwA training/development activities versus personal certifications (e.g., CISSP)
- Security certifications per unit growth in unit security/safety performance

The indices cited are not a recommendation about the specific ones that an individual business might adopt. Instead this list is an attempt to illustrate two things. First, it is possible to collect quantitative data that can be used to prove, or even disprove, a business case for software assurance. Second, it demonstrates

that data can be turned into meaningful information that can be easily understood by executive decision makers. The latter feature might be more important than the former, since any initiative such as this has to be valued and supported by upper management in order to succeed.

CONCLUSIONS

The aim of this article was to do two things. First, we wanted to identify only those common activities that were directly assignable to the cost of software assurance. Second, we wanted to provide a business case example from the 13 possible valuation approaches that we discussed in our prior article.

The indices presented in the balanced scorecard example all have a quantitative basis that can provide the data points for any form of economic analysis. The same would hold true for all of the other approaches. For instance, cost oriented models such as Total Cost of Ownership (TCO) and Economic Value Added (EVA) both are driven by quantitative operational data. Needless to say, quantitative estimation models such as Real Options Valuation (ROV) also depend for accuracy on hard data derived from actual practice.

This article has demonstrated how costs and benefits of software assurance can be derived from a standard set of practices, which can be reliably related through the current literature to the assurance case and the assurance case alone. The ongoing operational information that this would generate, using whatever economic valuation approach that an organization might deem appropriate, would establish an exact business value for the assurance process.

Moreover, that information can then be used by decision makers to make intelligent decisions about the amount of investment that they wish to make in securing their software and information assets. The confidence that an organization can gain from this insight will allow them to optimize their software assurance investment and thereby improve the overall safety and security of society as a whole.

REFERENCES

[CNSS 2006]

Committee on National Security Systems. National Information Assurance (IA) Glossary, Instruction No. 4009. Ft. Meade, MD: CNSS Secretariat, June 2006. http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf

[Colbert 2006]

Colbert, Edward & Yu, Dani. "Costing Secure Systems Workshop Report." 21st International Forum on COCOMO and Software Cost Modeling, Herndon, VA, Oct. 29-Nov. 2, 2006. Los Angeles, CA: Center for Systems and Software Engineering, 2006.

[ISO 1998]

ISO/IEC 15026, Information Technology -- System and Software Integrity Levels, International Standards Organization, 1998.

[Kaplan 1992]

Kaplan, Robert S. & Norton, David P. "The Balanced Scorecard: Measures That Drive Performance." Harvard Business Review 70, 1 (Jan.-Feb. 1992).

[McConnell 2005]

McConnell, Steve. "The Business Case for Software Development." Construx Software Builders Inc., 2005. http://www.igda.org/qol/IGDA_2005_QoLSummit_Business-Case.pdf

[McGraw 2004]

McGraw, Gary & Potter, Bruce. "Software Security Testing." IEEE Security and Privacy 2, 5 (Sept.-Oct. 2004): 81-85.

[Moore 2007]

Moore, James. "Report on Standards Activities ISO/IEC JTC 1/SC 7 and SC 22 and Associated IEEE Activities." Presentation to Task Lead Strategy Session, Department of Homeland Security, December, 2007.

[Redwine 2007]

Redwine, S. T.; Baldwin, R. O.; Polydys, M. L.; Shoemaker, D. P.; Ingalsbe, J. A.; Wagoner, & L. D. Software Assurance: A Curriculum Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software. Washington, DC: Department of Homeland Security, October 2007.

[Royce 1970]

Royce, W. W. “Managing the Development of Large Software Systems: Concepts and Techniques.” Proceedings WESCON. Los Alamitos, CA: IEEE Computer Society Press, 1970.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120