A Process-Oriented Methodology for Assessing and Improving Software Trustworthiness

Edward Amoroso, AT&T Bell Laboratories
Carol Taylor, National Security Agency
John Watson, Martin Marietta
Jonathan Weiss, AT&T Network Systems

Abstract

A high-level, technical summary of the Trusted Software Methodology (TSM) is provided in this paper. The trust principles and trust classes that comprise the TSM are presented and several engineering investigations and case studies surrounding the TSM are outlined. Appendices are included that highlight important areas of the TSM.

1 Introduction

An R&D effort has been on-going since 1989 to define the notion of software trustworthiness and to provide a means for assessing and improving the trustworthiness of both new and existing software. This effort has involved several different government and commercial organizations at various times including the Ballistic Missile Defense Organization (BMDO), the National Security Agency (NSA), GE Aerospace (now Martin Marietta), and AT&T Bell Laboratories. The result of the effort is an assessment and improvement approach that has been referred to as the *Trusted Software Methodology (TSM)*.

The initial investigations in this effort were focused primarily on the needs of the Global Protection Against Limited Strike (GPALS) system. However, as the TSM began to emerge[1,2], a number of additional groups chose to adopt its approach. For example, the Joint Integrated Avionics Working Group (JIAWG), an organization that advises a variety of avionics development and maintenance efforts, currently recommends use of the TSM on programs such as the Advanced Tactical Fighter (ATF).

In addition, a training course on the foundations of the TSM and its application to practical software development and maintenance efforts has been developed at Martin Marietta. The course has been offered during the past several years to hundreds of programmers, engineers, and managers within government and commercial software communities. This has served not only to disseminate the TSM, but also to obtain valuable feedback from practitioners.

While it is difficult to identify the primary factor contributing to the success of the TSM, it is certainly possible to list several candidate factors. For instance, unlike most existing software process standards such as the International Standards Organization (ISO) 9001 requirements[3] or the Software Engineering Institute (SEI) Capability Maturity Model (CMM)[4], the TSM includes explicit attention to security problems that may emerge during software development (see Appendix 3). This results in a more holistic protection strategy since it addresses not only inadvertent errors, but also deliberate malicious insertions.

In addition, the requirements of the TSM are organized into a collection of hierarchical classes that vary in their respective degrees of trustworthiness. While one might expect that all software development organizations would desire the highest degree of trustworthiness, practical concerns related to available funds and resources often preclude this goal. The TSM accommodates this situation by offering a range of different approaches to optimizing trustworthiness given existing cost constraints.

Another factor that has contributed to the success of the TSM is that the basic questions addressed in the design and development of the TSM requirements deal with issues that are of the utmost concern to managers of practical software development efforts. These issues are addressed by the following:

- How can the software development process optimize trustworthiness with respect to cost constraints?
- How can trustworthiness be maintained during the entire software lifecycle process?
- How is a target degree of trustworthiness determined for new software development processes?
- How is compliance with software process requirements evaluated and monitored?

Particular attention was placed on addressing these practical concerns in the TSM since it was reasoned that the degree to which satisfactory answers existed for these questions would have an enormous effect on the degree to which the TSM was actually applied in practice.

2 Definition of Software Trustworthiness

Nearly six months of extensive debate and discussions during 1989 were required before an acceptable definition of software trustworthiness could be identified. The reason for this difficulty was that many existing organizations used the term "trustworthiness" in different ways. For example, the National Computer Security Center (NCSC) had been promoting a standard for secure systems in its Trusted Computer System Evaluation Criteria (TCSEC)[5], also known as the Orange Book. As a result, many programmers, engineers, and managers had become accustomed to the notion of trustworthiness as dealing solely with security.

CCS '94- 11/94 Fairfax Va., USA
0-89791-732-4/94/0011

On the other hand, several research efforts had begun to emerge in the late 1980's and early 1990's, such as that of Parnas and others[6], that used the term "trustworthiness" in a different manner. Their focus, instead, was on the degree to which software engineering techniques such as enhanced testing, reviews, and inspections could be used to reduce the likelihood of errors in the development and maintenance lifecycle. Security was rarely, if ever, mentioned in these works.

The decision was made for the TSM to define the term in a manner that would take both security and software engineering into account. That is, the notion of trustworthiness would be focused on the avoidance of malicious insertions during development, as well as the prevention or mitigation of innocent errors. This broad scope required that the definition be generalized to the following: *The trustworthiness of software is defined as the degree of confidence that exists that it meets a set of requirements.*

This definition exhibited several characteristics that have since affected many of the technical and management decisions made with respect to the TSM. These characteristics include the following:

- Since the definition is expressed as a "degree of confidence," trustworthiness is dependent upon management and technical decisions made by individuals or groups of individuals evaluating the software.
- Since the definition is expressed with respect to "a set of requirements," trustworthiness is dependent upon the *selected* set of requirements. This may be the total set of functional requirements, it may be a critical subset of functional requirements, or it may be some set of requirements that include nonfunctional assurance requirements.

As will be discussed in the ensuing sections, the assessment and improvement approaches that are embedded in the TSM are driven by the definition of trustworthiness and its associated attributes.

3 Software Process Approach

Once the definition of software trustworthiness was agreed upon, the problem of how to assess trustworthiness had to be addressed. It became clear that two possible approaches existed: In the first approach, the emphasis would be placed on techniques for examining software products directly, and in the second approach, the emphasis would be placed on examination of the processes used to create these products. Both approaches exhibit merit and warrant discussion.

The primary benefits of examining software products directly are related to the fact that much useful information can be easily obtained using examination techniques that are familiar to most programmers, engineers, and managers. For instance, static analysis of code style, complexity, and organization using CASE tools is a common analytic technique for direct examination of software. In addition, dynamic analysis of software behavior, which includes all types of testing and reliability analysis, is another familiar technique for direct examination. Even the use of formal specification and verification is a form of direct software product examination and analysis.

A problem with the direct approach, however, is that if one's attention is restricted to direct product examination, then certain types of software flaws can be easily overlooked. As an example, recall Ken Thompson's description of a simple malicious attack method for inserting Trojan horse code into a compiler in a way that is not detectable during code inspections or reviews.[7] If such a malicious insertion were introduced to a critical software routine, then direct examination would probably not identify and remove the flaw. In addition, direct examination of software via inspection, review, or analysis does not ensure that flaws are avoided in subsequent reproduction, packaging, delivery, or maintenance activities. In fact, inspections often do not include any attention to whether a correlation exists between what is being reviewed (i.e., source code) and what is being executed (i.e., object code).

As a result, the decision was made to emphasize software process examination, rather than direct product examination. This emphasis on process caused the TSM to not focus solely on repeating certain activities in the software development lifecycle, such as tests and inspections, to assess trustworthiness. Instead, the TSM complements these activities by focusing on the manner in which they are performed during the actual development lifecycle. Such an approach places the burden of responsibility for demonstrating trustworthiness on the developers, rather than on the evaluators.

In addition, a process emphasis allows for the incorporation of reported previous experience at the Software Engineering Institute (SEI) in their process-oriented assessments into the approach being developed. It also allows for incorporation of reported experience in other process assessment and improvement approaches such as ISO 9001 and even NCSC Orange Book evaluations.

4 Trust Principles

The decision to follow a process-oriented approach led to an analysis and investigation into those characteristics of the software process that are most likely to reduce the potential for malicious or inadvertent software flaw insertion. A collection of forty-four *trust principles* was derived from familiar, generally-accepted software engineering and security principles in the available literature (see Figure 1).

These trust principles capture the best available technologies for countering innocent errors, as well as malicious insertions. They each specify a process attribute that contributes to enhanced software trustworthiness. Trust principles were selected based on several primary technical considerations. First, it was agreed that if any trust principle is ignored in a particular software process, then the resulting negative impact should not be not recoverable by other means. For instance, the damage that results from not documenting software is not recoverable by other process activities.

Second, it was agreed that each trust principle should be supported by documented experience, a sound technical foundation, and general acceptance across the software engineering and security communities. For example, a trust principle was included for basic access control to software resources. Certainly, the benefits of access control are well established, are associated with a sound technical foundation, and are generally accepted in the software community.

It is interesting to note that many students or practitioners being introduced to the TSM are often surprised that innovation in the trust principles was explicitly avoided. However, when the purpose of the TSM is revisited, it becomes clear to them why attention to established approaches is the best approach. In spite of this, the decision to avoid innovation does not preclude the future incorporation of present innovations that may ultimately become generally-accepted. However, it did preclude the introduction of many novel and interesting process suggestions to which the TSM development team was exposed (e.g., suggestions from students, research proposals in the literature), but that were deemed too immature for incorporation into a standard.

Finally, it was agreed that the trust principles should be logically separate to avoid the introduction of complex interactions between different principles. For example, if certain principles are dependent upon the degree to which other principles are met, then it becomes more difficult to utilize and apply the principles as independent building blocks. (It should be mentioned that this goal was not entirely met. Some principles, such as Auditing and Intrusion Detection, do have a dependence that could not be reasonably avoided.)

The text in Appendix 1 provides a general description of all forty-four principles. Each principle statement expresses a requirement on the software process. To ease the presentation, it is assumed that the underlying baseline process follows a familiar development life cycle as in the Military Standard for Software Development.[8] More specifically, requirements are imposed on the management issues that arise during the software process, the software development environment and tools that must be examined in the early stages of the process, and the various activities such as requirements analysis, design, development, test, and verification that arise through the entire process lifecycle.

For government and commercial software projects that are using the TSM, a more complete description of the trust principles has been made available through the Ballistic Missile Defense Organization (BMDO). These descriptions, a sample of which is presented in Appendix 2, include a more detailed set of compliance requirements that reduce the subjectivity in evaluating whether a given principle is complied with.

5 Trust Classes

Given the purpose of the trust principles, one might expect trustworthy software to be that software which is developed in accordance with all of the principle requirements. In fact, a great deal of consideration was given to this as a potential measurement approach. However, this binary view of trustworthiness is incompatible with the following observations:

- The definition of trustworthiness as a measure of confidence suggests that degrees of trustworthiness (based on degrees of confidence) are more appropriate than a strict binary definition.
- One would expect compliance with the majority of trust principles to result in more trustworthy software than compliance with only a few principles. This suggests degrees of trustworthiness.
- Finally, practical resource and schedule constraints for new software developments limit the degree to which certain trust principles (e.g., those associated with formal methods) can be applied in practice. A binary definition of trustworthiness would cause such efforts to be categorized as nontrustworthy.

As a result, a measurement scale was developed that included different degrees of trustworthiness ranging from the lowest rating T0 (unknown number of trust principles met) to the highest rating T5 (all trust principles met). Interim degrees of trustworthiness were organized into a totally ordered collection of *trust classes*. Each trust class thus represented an equivalence class of software development efforts (planned, on-going, or previous) that are designed to produce software of "equivalent" trustworthiness. The following criteria were used to combine the trust principles into trust classes:

Threats. An explicit threat model that included inadvertent and malicious threats was used as the basis for mapping trust classes to threats. The lower trust classes were designed to counter the most common and easily mitigated threats

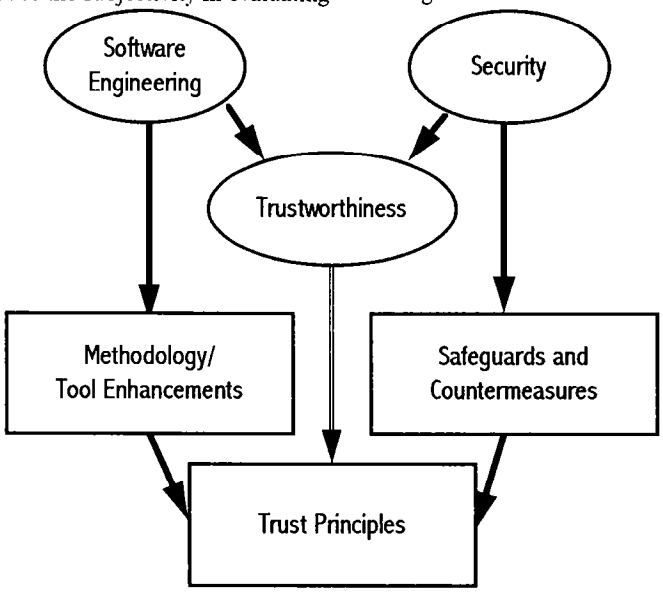

Figure 1. Rationale for the Trust Principles

such as version control errors, unauthorized login attempts, and simple object protection. The higher trust classes were designed to counter the more difficult threats such as flawed code insertions by malicious developers and viral attacks on operating system and application code.

Impact. The trust classes were designed to ensure that no class implies a smaller investment cost (e.g., short term impact on the development process activity cost) than some lower class. The reason for this is that if such a case ever existed in a set of criteria classes, then no reasonable manager would select the lower and more costly class.

State of the Practice. The trust classes were designed so that the current state of the practice was located in the "middle" of the trust class scale. This allowed for state-of-the-art process approaches and outdated, ineffective process approaches to reside on the same scale (albeit on opposite ends).

Logical Combination. In certain cases, determining which trust principles belonged in which trust classes required some subjectivity. The effects of this subjectivity were reduced by maximizing the degree to which feedback from practitioners was incorporated into the trust class organization. An extensive, on-going modification request program has been initiated to allow for suggestions on reorganization (or any other aspect of the TSM) from the software community.

The diagram in Appendix 4 describes the full set of six ordered trust classes which are denoted from T0 (lowest trust class) to T5 (highest trust class).

6 Selecting A Trust Class

If the trust classes are to be used to enhance the trustworthiness of software that has yet to be developed, then during the planning stages of the associated development effort, a suitable target trust class must be selected. This is not an easy task, because it requires that one consider all of the following factors:

Threats. If certain serious software development threats are identified for a particular software process, then a higher trust class may represent the best available mitigation approach, even in the presence of a limited budget. Malicious integrity threats during development, in particular, are best mitigated by selecting a higher trust class (as opposed to other mitigation approached not related to the TSM).

Cost. The only reason the highest trust class is not selected for every software development effort is that an up-front investment is required in most cases to meet the higher class requirements. For instance, higher classes require that security enhancements be made to the software development environment (e.g., auditing and intrusion detection of development activity). As a result, a trade-off must be made between increased trustworthiness and required cost investments. The most familiar scenario reported by users of the TSM is that insufficient funds usually force the selection of a lower trust class than is really needed.

Criticality. A commonly encountered heuristic in allocating trustworthiness requirements to any software component is that more critical software components should be associated with higher trustworthiness requirements. A typical definition of critical component is a component whose removal or malfunction would seriously jeopardize the success of the system mission or purpose. Unfortunately, since criticality is application specific, general procedures do not exist for determining criticality. In fact, engineering consensus is often the best approach. However, to reduce the subjectivity of total reliance on engineering consensus, certain factors can be considered in assessing criticality:

- Software that directly contributes to the functionality of a well-defined system critical path is generally viewed as critical.
- Software that controls single points of failure is generally viewed as critical.
- Software associated with unusually stringent requirements in the areas of security, reliability, availability, or similar attributes is generally viewed as critical.

An issue that emerges when different trust classes are selected for subsystems of larger scale systems is that dependencies may be introduced between components at different trust classes. An explicit risk mitigation should be performed to ensure that any potentially suspect dependency (e.g., more trustworthy components depending upon less trustworthy components for information or services) will not introduce unacceptable risk.

For example, suppose that a component is developed to meet the T3 class requirements using existing software library routines that were not developed in accordance with certain T3 class requirements. While the use of such routines introduces risk because it does not maintain a uniform process for all components of the software, it would be impractical to expect any software process manager to avoid software reuse for this reason. As a result, risk mitigation approaches such as the introduction of software architectural mechanisms to restrict information flow between components with different trustworthiness, could be employed to deal with this type of problem. It is worth mentioning, in addition, that a software reuse trust principle is also included to provide guidance on selecting software for reuse.

7 Trustworthiness Evaluation

A practical concern that emerges when trust classes are used as process requirements for software development is whether a given developer has complied with (or should be expected to comply with) the requirements for the selected trust class. For example, trust classes often include subjective requirements such as the use of "qualified" individuals for certain activities or the provision of "suitable" documentation in certain areas. Determination of which individuals are properly qualified, which documents are suitable, and similar types of subjective considerations led to the development of a trustworthiness *evaluation* role as part of the TSM.

The purpose of evaluation in the context of the TSM is to collect and document evidence related to the compliance or noncompliance of a software development contractor with the requirements of a selected trust class. In this sense, trustworthiness evaluation is reminiscent of the Orange Book evaluation efforts at the National Computer Security Center (NCSC) in which evidence is supplied by a development organization to determine compliance with the functional and assurance requirements in the Orange Book. This similarity between the TSM and Orange Book evaluation approaches allowed for the creation of evaluation goals for TSM that address well-known advantages and disadvantages of the Orange Book evaluation approach. Specifically, the trustworthiness evaluation goals for the TSM are as follows:

- To minimize the effect of trustworthiness evaluation on existing software development cost including resource, staff, and scheduling concerns.
- To reduce the incidence of arbitrary decisions related to process requirements compliance.
- To maximize the value added by trustworthiness evaluation in the areas of quality, security, and lifecycle cost.

These goals were accomplished via an evaluation process that starts with an extensive review and assessment of the Software Development Plan (SDP) by an explicit evaluator. Ideally, this SDP assessment and review should be completed before the software development activities described in the SDP can even begin. This SDP assessment and review is followed by a close monitoring of software development activities by the evaluator. Key characteristics of the evaluation approach in support of the goals listed above include the following:

- Early Decision-Making. By emphasizing up-front evaluation of the SDP, many of the key decisions related to development and assessment can be worked out before development proceeds. This reduces "on-the-fly" evaluation issues that arise during later stages of development that can greatly increase development costs.
- Assessment Based on Evidence. To minimize the subjectivity of evaluation, trust principle compliance or noncompliance is established based exclusively on available evidence. For GPALS projects, a comprehensive evaluator's guide has been developed that provides examples of suitable compliance and noncompliance evidence for each trust principle.
- Independent Evaluation. The TSM incorporates the recommendation that trust evaluation be performed as part of existing Quality Assurance (QA) or Independent Verification and Validation (IV&V)activities to reduce cost. However, nothing precludes a member of the development team or customer from performing the evaluation.

8 Case Studies

The TSM has been applied in a variety of software development efforts. In this section, we briefly summarize two such efforts, one performed at AT&T Bell Laboratories during the summer of 1991, and the other performed at GE Aerospace during most of 1991. Other assessments with respect to the TSM have been performed with varying goals (see, for example, the assessment of the UNIX System V/MLS development[2]). The primary goal of the two case study efforts described below was to demonstrate feasible application of the trust principles in practical software development settings, as well as to improve the trustworthiness of the resultant software.

8.1 Case Study: STAT Development

The AT&T Bell Laboratories case study involved the prototype development of a tool called STAT, which was designed to allow software development managers to perform "what-if" scenarios with models of their software development process. An on-line questionnaire is used to build a model and the tool allows managers to assess which trust principles are met in their model and how they might adjust the model toward a target trust class in the most cost-effective manner.

It was decided that the STAT development would provide an opportunity to examine some of the trust principles that were in higher trust classes and, as such, were less likely to be present in typical development approaches. These trust principles specifically included (but were not limited to) the following:

- Formal Methods Approach (Rigorous, mathematical representations of requirements and design specifications were used).
- Auditing (A secure UNIX environment provided on-line auditing).
- Identification and Authentication (Machine-generated passwords and multilevel secure password storage were employed).
- Intrusion Detection (Audit records were processed using an intrusion detection system).
- Mediation (Mandatory and discretionary access control mechanisms were provided by the environment).
- Least Privilege (Privileges to make software changes were allocated based on need).
- Multi-Person Control (Individuals could not initiate software repository changes alone).
- Shared Knowledge (Every module was thoroughly understood by multiple individuals—a subjective consideration).

The development involved three full-time programmers for a period of approximately four months. The 7K NCSL (noncommented source lines) of code was written in Ada using the Verdix Ada Development System (VADS) on an AT&T 3B2-500 running the UNIX System V/MLS secure operating system. Formal specifications were written in the Ina Jo specification language. The following results and observations were obtained from the case study development:

- Meeting the security-oriented principles was facilitated by performing the development on a secure UNIX System V/MLS platform that provided all of the required functionality. Since this was an existing development platform being used for normal government and commercial software development, no additional training or procurement was necessary for the case study.
- The use of formal methods did not greatly impact the development schedule and did not require inordinate amounts of training. One member of the development team had been introduced to formal methods in graduate school and the others were shown the syntax and semantics of Ina Jo. The formal requirements and design specifications were created manually and used as design and documentation aids. Automated theorem proving was not performed. One unexpected benefit of the requirements specification was that the test plan seemed to follow naturally from the formal specification.
- Those requirements associated with the traditional notion of separation of duty (least privilege, shared knowledge, and multi-person control) were also easily met. The team found that the shared knowledge principle, in particular, was of great use. The methods in support of this principle came in handy during reviews and when certain individuals were not available to answer questions.

8.2 Case Study: AIM Development

The GE Aerospace case study involved a development effort in accordance with the Military Standard for Software Development (i.e., DoD-STD-2167A). The primary goal of this development was to demonstrate the feasibility of the T3 trust class requirements in a practical development effort. The application selected for development was the redevelopment of an Advanced Interactive Monitor (AIM) for an ASCII terminal that had been developed earlier at Texas Instruments. Development statistics were made available on this earlier development so that comparisons could be made.

The 10K NCSL of code was developed in Ada using the Verdix Ada Development System (VADS) on a Sun 300 platform running SunOS. The development team consisted of a full-time manager, five full-time programmers, one part-time tester, one part-time configuration manager, and one part-time administrator. In addition, AT&T Bell Laboratories provided a part-time evaluator who monitored development with respect to the trust principles. The following results and observations were obtained with respect to the AIM case study development:

- The T3 class did not introduce severe investment costs although it did result in more time (approximately double) in requirements analysis than would have been planned in the absence of the trust principles. However, this was balanced by a much shorter time for coding and testing. In fact, during the subsystem testing, only two minor errors were found (which is far less than any of the testers had expected or had observed in previous, similar efforts).
- In general, the development team expressed their belief that they had performed everything required at T3 on previous separate projects, but that they had never done them all together on the same project before.
- Comparisons with the original Texas Instruments statistics were less revealing than had been originally expected. It turned out that the Texas Instruments development had been stretched out over a period of three years by a team of programmers who did not spend full-time on the development. Thus, differences in schedule and resources were less meaningful than had been hoped.
- Evaluation activities were performed at AT&T Bell Laboratories and only minor violations occurred (e.g., certain password protections violated a compliance requirement in one trust principle) and all compliance evidence was documented.

9 Cost/Benefit Analysis

The most frequent issue that arises during any discussion of the TSM is the estimated impact the process requirements will have on one's existing or proposed development approach. Making accurate estimates of such impact is complicated by the fact that different developers have different baseline approaches and no single unit of cost and benefit can be easily established to determine net impact. For example, additional resources spent during requirements analysis may be viewed as a net increase in development cost. However, if this results in reduced test time and increased code quality, then these benefits must be taken into account as well.

In spite of these difficulties, we performed an extensive cost and benefit analysis that included both qualitative and quantitative activities. The qualitative activities included comprehensive literature searches to identify reported instances of cost or benefit experience with any of the trust principles. For example, the benefits of design and code reviews are well-known. All such information, including the experiences of the programmers involved in the case studies, was collected into a cost and benefit database for analysis.

The quantitative activities included management of statistics on the case study efforts. In the GE Aerospace case study experiment, programmers were asked to keep track of their specific activities and progress during development. This information was collected and compared with cost model predictions of costs for a nontrustworthy software development effort. Since no cost models included direct references to trust principles or classes, we had to estimate the impacts of the trust principles and classes in the input domain of the cost model. This allowed us to compare results for development efforts with and without the trust principles.

Based on the qualitative and quantitative analysis, the following general results were identified:

- As one increases the trustworthiness of a development effort, one generally front-loads the development process toward requirements analysis and design and shortens subsequent coding, assurance, and maintenance activities.
- Identifying specific percentages that would apply to all development projects would only be as accurate as the development projects are similar. For the GE Aerospace case study experiment, it was estimated (using the Checkpoint model and statistics from the case study) that compliance with the T3 class would reduce subsequent maintenance over nontrustworthy approaches by as much as 40 percent.
- Estimating cost is much easier than estimating the corresponding benefit of a particular trust principle or class.
- Lifecycle benefits from the trust principles and classes increase with the size and complexity of a project.

10 Concluding Remarks

Experience with the Trusted Software Methodology confirms the benefits of a process oriented approach to trustworthy software. Perhaps the strongest evidence of this lies in the adoption of the TSM by organizations that have not been required or pressured to do so. Instead, these organizations have determined that the process-oriented guidance offered in the TSM will help them to create trustworthy software in a cost effective manner.

One of the greatest challenges that must be addressed in current TSM-related research and development is that other existing process models and standards such as the CMM and ISO 9001 must be integrated with the trust principles and trust classes in a way that allows software development managers to make reasonable decisions that have positive impacts on their development approaches.

Acknowledgments

Significant technical contributions to this work have been made by the following individuals: Cheri Dowell, Dan Goddard, Richard Iliff, Sara Hadley, Dave Harris, Howard Israel, Pete Lapiska, Mike Molloy, Thu Nguyen, Linda Pyfrom, Phil Sikora, Lina So, Terry

Starr, Jim Sweeder, and John Wilson. Additional contributions and reviews have been made by the following: Julian Center, Bill Everett, George Hoover, Eric Leighninger, and John Perkins.

References

1. J. Watson and E. Amoroso, "A Trusted Software Development Methodology," *Proc. 13th Natl. Computer Security Conf.*, Oct. 1990, pp. 717-727.
2. E. Amoroso et al., "Toward An Approach to Measuring Software Trust," *Proc. IEEE Symp. Research in Security and Privacy*, May 1991, pp. 198-218.
3. International Standards Organization (ISO), *ISO 9000*, Second Edition, 1987.
4. W.Humphrey and W. Sweet, "A Method for Assuring the Software Engineering Capabilities of Contractors," CMU/SEI-91-TR-25, 1987.
5. Department of Defense, National Computer Security Center, *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD, 1985.
6. D. Parnas et al., "Evaluation of Safety-Critical Software," *CACM*, Vol. 33, No. 6, June 1990, pp.636-648.
7. K. Thompson, "Reflections on Trusting Trust," *CACM*, Vol. 27, No. 8, August, 1984, pp.761-763..
8. Department of Defense, *Military Standard for Software Development*, DoD-STD-2167A.

Appendix 1: Trust Principles

For GPALS software development, specific trust principle statements and associated sets of detailed compliance requirements have been developed. These statements are shown below, but for lack of space, the compliance requirements have been omitted (full trust principle compliance requirements documentation runs to over one-hundred pages). Each trust principle is intended as a requirement on the development process that should increase the trustworthiness of any developed software. Readers interested in the full set of compliance requirements should contact the authors.

A few of the concepts mentioned in the statements below may be unfamiliar to some readers. For example, several references are made to terms such as Computer Software Configuration Items (CSCIs), Computer Software Component (CSC), and Computer Software Unit (CSU). These terms refer to standard software components in the DoD-STD-2167A Military Standard for Software Development. The documents mentioned (e.g., SDD, IDD) are also standard in DoD-STD-2167A. A concept mentioned repeatedly below is the "identified software lifecycle activity". This refers to the software development or maintenance activity relevant to the principle at hand (defined more specifically in the compliance requirements documentation).

Access Control: Identified software lifecycle activity shall be automatically controlled by the software engineering environment with respect to an explicitly defined security policy.

Administration: The software engineering environment, software tools, and the developed software shall be maintained according to explicit administration documentation by qualified individuals.

Auditing: A record of identified software lifecycle activity shall be automatically logged and stored by the software engineering environment in a protected repository.

Configuration Management: A configuration management system shall be established and shall include mechanisms and explicit procedures for configuration identification, configuration accounting, configuration control, and configuration auditing. All configuration items shall be stored in a protected repository that maintains all software versions, software modification requests, and software changes.

Design Documentation: In addition to the Software Design Document (SDD) and Interface Design Document (IDD), the characteristics of the design activity, critical design alternatives considered, and critical design rationales shall be documented.

Design Review: Design peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the software design.

Design Tools: Design Computer-Aided Software Engineering (CASE) tools shall be employed to maintain design/requirements traceability mappings and to generate design documentation.

Design Traceability: All aspects of the design shall be shown to be traceable to the requirements and all requirements shall be shown to be traceable to the design.

Environment and Tool Selection: The software engineering environment and all software tools shall be selected according to an explicit selection policy that considers the trust rating, maturity, documentation, and source code availability of that software.

Environment Integrity: An explicit procedure shall be available for identifying changes in the software engineering environment components and, if required, to restore the integrity associated with that software.

Formal Design Specification: In addition to any informal design specification, the design shall also be specified in a formal specification framework.

Formal Design Verification: A formal verification shall be performed to prove that the formal design specification correctly meets its requirements.

Formal Methods Approach: All formal specification and verification activities shall follow an approach which includes the use of a formal specification and verification toolset, documentation, peer reviews, and traceability mappings.

Formal Requirements Specification: In addition to informal requirements specifications, functional requirements shall also be specified in a formal specification framework.

Formal Source Code Verification: A formal verification shall be performed to prove that a low-level formal specification of the source code correctly meets its requirements.

Identification and Authentication: The initiation of an identified software lifecycle activity shall be done only be an individual that has been identified and authenticated by the software engineering environment.

Intrusion Detection: Audit trail data shall be used to perform periodic and random intrusion detection analysis on the software engineering environment.

Least Privilege: Privileges to perform identified software lifecycle activity shall be allocated and maintained so that a privilege is only given to individuals who require that privilege.

Multi-Person Control: Identified software development activity shall not be completed without the active endorsement and involvement of at least two qualified software developers.

Planning: Detailed plans for all software development activity (including trust principle compliances) shall be described in a Software Development Plan (SDP) and the management of the software development shall follow the approach described in the SDP.

Prototyping Approach: All prototyping that is performed as part of the risk mitigation strategy shall be performed according to an explicitly defined prototype plan that describes the way in which the prototype is designed, developed, tested, documented, and protected.

Prototype Software Reuse: When prototype software is reused in the developed software, then the prototype software shall be sufficiently documented, reviewed, and tested to ensure that the level of trust is commensurate with the developed software.

Requirements Analysis Documentation: In addition to the Software Requirements Specification (SRS) and Interface Requirements Specification (IRS), information useful in understanding the software requirements analysis process and a rationale for all critical requirements analysis decisions shall be documented.

Requirements Analysis Review: Requirements analysis peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the software requirements.

Requirements Analysis Tools: Requirements analysis Computer-Aided Software Engineering (CASE) tools shall be employed that provide for requirements specification, consistency checking, and documentation generation.

Requirements Traceability: All software requirements shall be shown to be traceable to an explicit system requirement or

customer source and all system requirements allocated to a Computer Software Configuration item (CSCI) shall be shown to be traceable to a software requirement.

Reliability Measurement: Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) test and field results shall be used to reduce observed software failure rates to acceptable levels.

Risk Mitigation: All potential risks associated with the software development activity shall be explicitly identified and a risk mitigation strategy shall be documented and complied with throughout the software lifecycle.

Security Policy: All software development activity shall be performed in accordance with an explicitly defined and enforced security policy with respect to all software developers and software development resources.

Shared Knowledge: Each identifiable component of the software development activity, including all aspects of requirements, source code, design, tests, proofs, software tools, methodologies employed and support activity, shall be associated with at least two individuals who are thoroughly familiar with the details, implications, and alternatives considered for that component.

Software Reuse: All reused software shall be subject to an explicit selection policy that considers the trust rating, maturity, documentation, and source code availability of the software.

Source Code Standards: An explicitly defined source code standard that enforces modular, structured programming shall be complied with throughout the coding activity.

Source Code Analysis: All developed code shall be subjected to code analysis using tools and procedures that measure complexity and style.

Source Code Review: Source code peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the source code and Computer Software Unit (CSU) tests.

Source Code Documentation: The source code and characteristics of the software coding activity shall be documented.

Source Code Traceability: All source code shall be shown to be traceable to the design and Computer Software Unit (CSU) tests. The design shall be shown to be traceable to the source code.

Test Documentation: In addition to the Software Test Plan (STP), Software Test Description (STD), and Software Test Report (STR), the characteristics of the Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) test activity shall be documented.

Test Responsibility: The responsibility for Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) testing shall be placed with an independent group or organization not involved with coding or design of the software being tested.

Test Review: Test peer reviews shall be conducted by a peer review team to ensure the completeness, consistency, and correctness of the tests.

Test Strategies: All Computer Software Unit (CSU), Computer Software Component (CSC), and Computer Software Configuration Item (CSCI) test and integration tasks shall include provisions for various testing strategies.

Test Tools: The software engineering environment shall include a testbed for creating, executing, documenting, and analyzing the completeness of all tests.

Test Traceability: All Computer Software Component (CSC) and Computer Software Configuration Item (CSCI) tests shall be shown to be traceable to the software requirements. Both the source code and the software requirements shall be shown to be traceable to the CSC and CSCI tests.

Trusted Distribution: All software shall be transferred from its source to its destination in a way that ensures that the integrity has not been compromised during the transfer.

Trusted Path: An explicit mechanism shall be included in the software engineering environment to ensure that identified software lifecycle activity cannot be intercepted by unauthorized means.

Appendix 2: Sample Compliance Requirements Description

Since space does not permit inclusion of the compliance requirements descriptions for all of the trust principles, only one is included here as a representative sampling of the exact text from the documentation provided for users of the TSM. All of the trust principles follow an organization similar to the one shown below for Trusted Path (including the set of references depicted).

Trusted Path Principle

An explicit mechanism shall be included in the software engineering environment to ensure identified software lifecycle activity cannot be intercepted by unauthorized means.

Rationale

Many successful malicious attacks on computer and network systems rely on the use of spoof techniques that imitate some aspect of the software engineering environment in order to intercept or block some critical information interchange. Ensuring that all communication paths between developers and the software engineering environment are trusted, reduces the potential for such attacks. For example, the well-known "login spoof" technique aimed at stealing password information is countered by a login trusted path.

Compliance Requirements

Additional Detail

a. (T4, T5) Identified software lifecycle activity shall include the identification and authentication sequence in which user identity is established and then validated. (See the Identification and Authentication Principle.)

b. (T4, T5) Identified software lifecycle activity shall include the execution of any commands or utilities which require a user to enter a password (e.g., remote login activity, changing of the user's password, or remote file transfers).

c. (T5) Identified software lifecycle activity shall be extended to include constrained software development operations (CSDOs) which shall be associated with a trusted path when they are invoked.

d (T4, T5) A mechanism shall be included that will reliably assure authorized users of the success or failure of a requested software lifecycle activity.

Exceptions - (T4, T5) Trusted path shall only be bypassed when the software engineering environment is not in a normal run state. When this is the case, access shall only be allowed in a controlled, physically protected location by an administrator.

Administration - (T4, T5) The trusted path mechanism shall be administered in accordance with the Administration Principle.

Duration - (T4, T5) Trusted path shall be enforced throughout all software design, development, and maintenance lifecycle activities.

Environment - (T4, T5) The mechanism shall be automatic and shall be integrated into the software engineering environment.

Target - (T4, T5) The identified software lifecycle activity shall be determined by the target criteria class. (See Additional Detail above.)

Qualifications - N/A

Granularity - (T4, T5) A trusted path mechanism shall be provided for each software lifecycle activity identified above.

Protections - N/A

Standards - N/A

Additional Considerations

1. Fine-tuning a full session trusted path to a set of CSDOs in Trust Class T5 may require a customization of the software engineering environment.

General References on Trusted Path

1. Department of Defense, Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December, 1985.

2. Department of Defense National Computer Security Center, Glossary of Computer Security Terms, NCSC-TG-004, October, 1988.

Appendix 3: How Does The TSM Compare With Other Process Standards?

The recent focus on process in the software community has led to the application and use of a variety of different software process standards that enhance different software attributes. The most well-known such standards include the Software Engineering Institute (SEI) Capability Maturity Model (CMM) and the International Standards Organization (ISO) 9001 standard for software. Since the TSM provides process guidance in a manner somewhat similar to these standards, it is useful to examine the relative similarities and differences.

Similarities: The TSM, CMM, and ISO 9001 standards all focus on the enhancement of the software process as a means for enhancing the resultant software. All include provision for the assessment and improvement of those activities that comprise a typical development and maintenance process including reviews, documentation, and management policy. Thus, enhancements directed by any of the standards are likely to promote compliance with the other standards.

Differences: Unlike the TSM, the CMM and ISO 9001 standards do not include provision for dealing with malicious developers during the software process. Also, unlike the TSM and CMM, ISO 9001 does not include degrees of compliance designed to allow for incremental evolution toward the highest rating. Finally, unlike the CMM and ISO 9001, the TSM focuses on a specific development process, rather than the characteristics of the development organizations.

Appendix 4: Trust Class Organization

Note that the diagrammatic approach in the figure below is taken directly from the Orange Book. Each row corresponds to a trust principle compliance requirement and each row corresponds to a trust class.