The DoD SoftwareTech NEWS

WWW.SOFTWARETECHNEWS.COM

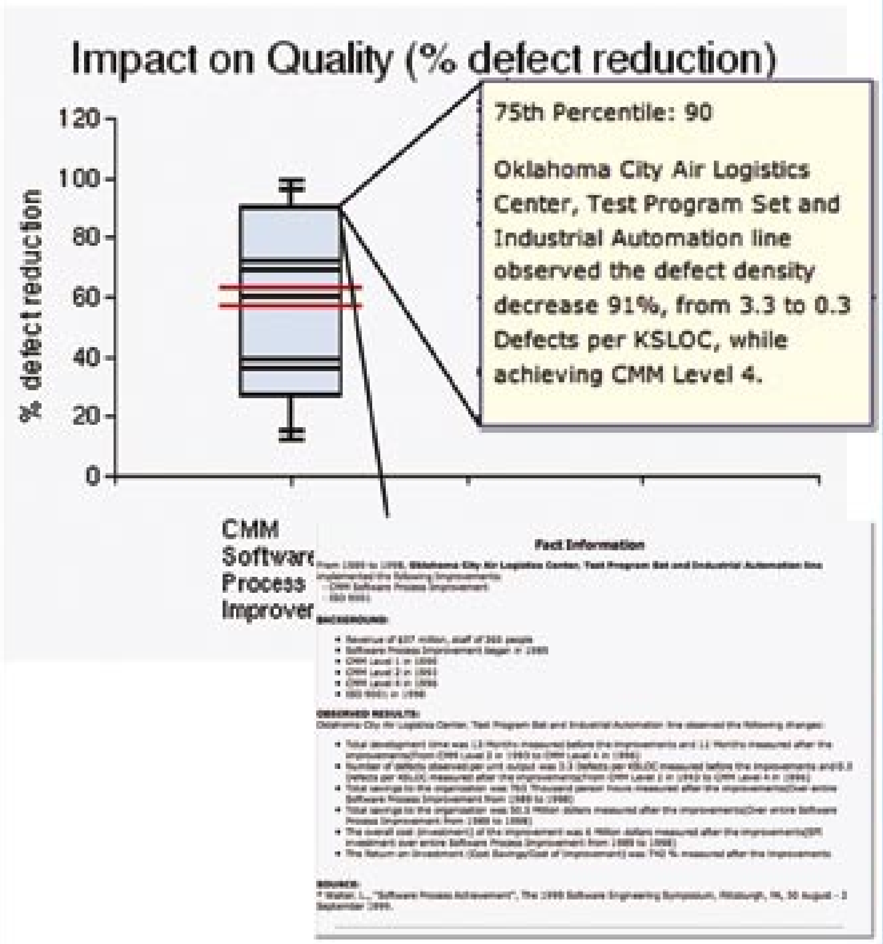
JULY 2005 Vol.8. No.2

Secure Software Engineering

http://iac.dtic.mil/dacs/

Are you getting *All* you can from your software investment?

The DACS ROI Dashboard


Impact on Quality (% defect reduction)
% defect reduction
120
100
80
60
40
20
0
CMM Software Process Improver

75th Percentile: 90

Oklahoma City Air Logistics Center, Test Program Set and Industrial Automation line observed the defect density decrease 91%, from 3.3 to 0.3 Defects per KSLOC, while achieving CMM Level 4.

Fact Information

[illegible]

BACKGROUND:

- [illegible]

OBSERVED RESULTS:

[illegible]

- [illegible]

SOURCE:

[illegible]

Technologies Covered:

- SEI CMM/CMMI
- SEI Team Software Process (TSP)
- SEI Personal Software Process (PSP)
- Inspections
- Reuse
- Cleanroom

And Many More!

Graphs Showing Impact Of Software Technologies On:

- ROI
- Productivity
- Quality

Summarizes Facts From Open Literature

Data & Analysis Center for Software
P.O.Box 1400
Rome, NY 13442-1400

http://iac.dtic.mil/dacs

Access the DACS ROI Dashboard!
http://www.thedacs.com/databases/roi/

DACS
Data & Analysis Center for Software

Developing Secure Software

Noopur Davis, Software Engineering Institute

Abstract

Most security vulnerabilities result from defects that are unintentionally introduced in the software during design and development. Therefore, to significantly reduce software vulnerabilities, the overall defect content of software must be reduced. Defect reduction is a pre-requisite for secure software development, but it is not enough. Security must also be deeply integrated into the full software development life cycle (SDLC).

Introduction

Most security vulnerabilities result from defects that are unintentionally introduced in the software during design and development. Therefore, to significantly reduce software vulnerabilities, the overall defect content of software must be reduced. Today's common software engineering practices lead to a large number of defects in released software. However, data from dozens of real-world software projects that have systematically applied improved software development practices show one to two orders of magnitude reduction in the number of defects in released software. Applying these improved practices should lead to a similar reduction in the defects that lead to vulnerabilities. Furthermore, by focusing on the specific types of defects that lead to vulnerabilities, even greater reduction in vulnerabilities could be achieved. Organizations that have applied these practices have realized additional benefits of reduced cycle times and reduced software development costs.

Along with defect reduction, Security must be deeply integrated into the full software development life cycle (SDLC). Security must be "built-in" while the product is being developed, and not just "bolted-on" after the fact.

This article begins with a discussion of why defective software is seldom secure, why defective software is not inevitable, and why reducing defects is less costly than responding to released vulnerabilities. Next, security throughout the software development life cycle will be discussed. The paper closes with a brief description of the Software Engineering Institute's (SEI's) Team Software Process^{SM} for Secure Software Development (TSP-Secure).

Defective Software Is Seldom Secure

SEI analysis of thousands of programs produced by thousands of developers show that even experienced developers inject numerous defects as they perform activities for understanding requirements, developing designs, coding, and testing software. One defect is injected for every 7 to 10 lines of new and changed code produced. Even if 99% of these defects are removed before the software is released, this leaves 1 to 1.5 defects in every thousand lines of new and changed code produced. Software benchmark studies conducted on hundreds of software projects show that the average defect content of released software varies from about 1 to 7 defects per thousand lines of new and changed code [Jones].

According to preliminary analysis done by the SEI's CERT® group, over 90% of software security vulnerabilities are caused by known software defect types. The analysis also showed that most software vulnerabilities arise from common causes: the top ten causes account for about 75% of all vulnerabilities. Another analysis of forty-five e-business applications showed that 70% of the security defects were software design defects [Jacquith]. Some problems are caused by sophisticated architectural and design issues such as inadequate authentication, invalid authorization, incorrect use of cryptography, failure to protect data, and failure to carefully partition applications. But most are caused by simple oversight that leads to defect types such as declaration errors, logic errors, loop control errors, conditional expressions errors, failure to validate input, interface specification errors, configuration errors, and failure to understand basic security issues. In a recent interview, Alan Paller, director of research at the SANS Institute, "expressed frustration with the fact that everything on the [SANS Institute Top 20 Internet Security] vulnerability list is a result of poor coding, testing and sloppy software engineering. These are not 'bleeding edge' problems, as an innocent bystander might easily assume. Technical solutions exist to them all, but they are simply not implemented."

It is clear that software development practices in common use today lead to defective software, that software defects are a principal cause of software security vulnerabilities; therefore, to reduce vulnerabilities the overall defect content of software must be reduced.

Defective Software Is Not Inevitable

When presented with the security problems caused by defective software, a common response is that software development is inherently prone to defects, and that defective software is somehow inevitable. Many people believe that trying to figure out how to build better software is "a no-win situation and just beating a dead horse" [Computer World]. However, data from dozens of real-world projects have shown that when developers follow defined, measured, and quality controlled practices, they produce products with very few overall defects. A recent study

continues on page 4

Developing Secure Software

Continued from page 3.

found that the defect content of such products can be reduced to an average of .06 defects per thousand lines of new and changed code [Davis]. This represents 10 to 100 times fewer defects when compared to industry averages of 1 to 7 defects per thousand lines of new and changed code.

The Cost of Reducing Defects

The next question usually asked is "doesn't it cost too much to reduce defects in software"? The simple answer is that software projects that produce near defect-free software also consistently meet their schedules (thus avoiding costs associated with delayed releases), and spend less time on software repair (thus improving overall productivity). For example, the average schedule error for projects using best practices was just 6%, the average time spent on software repair was just 4%, and the average increase in productivity was 78% [Davis]. Another large scale study showed a near perfect corelation between schedule and quality: the fewer the defects in the software, the lesser the schedule error [Jones].

When discussing costs, it is also fair to discuss the costs of releasing software with vulnerabilities. Producers of vulnerable software face the tangible costs of fixing and releasing patches for vulnerabilities, as well as the intangible costs of bad press, customer dissatisfaction, and threat of legal action. For consumers, the costs are even higher. A recent analysis conducted at a major corporation determined that the cost to deploy a single patch was close to half a million dollars. This cost was incurred just by the corporate infrastructure team: it did not include costs incurred by other teams such as the development teams. When these costs are multiplied by hundreds of patches that need to be applied by thousands of corporations, the overall costs to the consumers are enormous.

Secure Software Development

What can be done to reduce defects in software, and thus reduce vulnerabilities in software? Two things must be done: defects must be managed throughout the software development life cycle, and security must be addressed throughout the software development life cycle.

Managing Defects throughout the Software Development Life Cycle

Defects delivered in released software are a percentage of the total defects introduced during the software development life cycle. To reduce defects in released software, defects must be managed throughout the software development life cycle. Defect management includes both defect removal and defect measurement.

There should be multiple defect removal points in the software development life cycle. The more defect removal points there are, the closer one is to finding problems right after they are introduced. So the problems can be more easily fixed, and the root cause more easily determined and addressed.

Each time defects are removed, they should be measured. Every defect removal point becomes a measurement point. Defect measurement leads to something even more important than defect removal and prevention: it tells you where you stand against your goals now, helps you decide whether to move to the next step or to stop and take corrective action, and indicates where to fix your process to meet your goals.

The following questions must be considered when managing defects: where are the places in the software development life cycle where defects should be measured? What work products should be examined for defects? What tools and methods should be used to measure the defects? How many defects can be removed at each step? How many estimated defects remain after each removal step?

Suppose an organization has determined that it wants to produce software with less than 1 vulnerability per million lines of code. Also suppose that 25% of all software defects can lead to software vulnerabilities. Thus the quality goal for the organization is to release software with less than 4 defects per million lines of code. How will the organization know it can deliver an acceptably small number of defects to meet its quality goals? Like most organizations, suppose the first time this organization measures defects in the software development life cycle is during test. If testing exposes 100 defects per million lines of code, and like most organizations, testing in this organization is 50% effective, 100 defects per million line of code would remain in the software after testing, and would be released with the software (200 defects per million lines of code existed before the software entered test, 50% of these defects were found and fixed during test, while another 50% remained unfound and unfixed). Not only will the organization not meet its quality goal, but few options will be available for corrective action at this late stage in the development life cycle.

On the other hand, if this organization had several defect removal points in the software life cycle, each 50% effective, the defects in the released software would be much fewer. Each defect removal activity can be thought of as a filter that removes some percentage of defects that can lead to vulnerabilities from the software product, while others defects that can lead to vulnerabilities escape the filter and remain in the software (see Figure 1). The more defect removal filters there are in the software development life cycle, the fewer defects that can lead to vulnerabilities will remain in the software product when it is released. More importantly, be-

continues on page 5

Developing Secure Software

Continued from page 4.

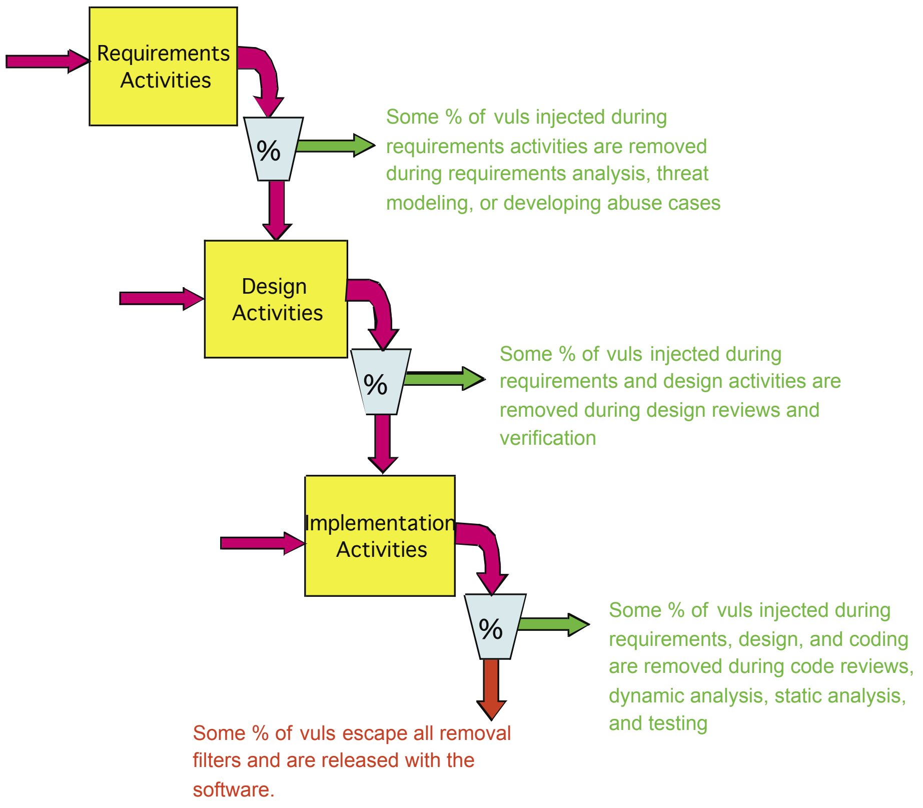

Requirements Activities
%
Some % of vuls injected during requirements activities are removed during requirements analysis, threat modeling, or developing abuse cases
Design Activities
%
Some % of vuls injected during requirements and design activities are removed during design reviews and verification
Implementation Activities
%
Some % of vuls injected during requirements, design, and coding are removed during code reviews, dynamic analysis, static analysis, and testing
Some % of vuls escape all removal filters and are released with the software.

Figure 1: Vulnerability Removal Filters

cause the defects were being measured earlier, the organization would have time to take corrective action early in the software development life cycle.

Some examples of defect removal and measurement points in the software development life cycle are architectural analysis, threat modeling, design verification, design review, code review, static code analysis, unit test, penetration test, and system test.

Addressing Security throughout the Software Development Life Cycle

Although defect reduction is the key to vulnerability reduction, more is needed to produce secure software.

First, common causes of security vulnerabilities must be understood. Some common causes include buffer overflows, SQL injection, race conditions, and cross-site scripting. Understanding involves much more than reading a laundry list of causes and examples: some organizations have 700-page documents to teach developers about common causes of vulnerabilities and how to avoid them. No one should expect developers to use such a volume of information as they perform their day to day software development activities. Although an overall knowledge of security issues is important, eliminating common causes of vulnerabilities requires defining a set of operational best practices that development teams can use in their day to day work: scripts, tools, checklists, and methods that focus on the particular job the developer is doing at a particular time.

For example, consider buffer overflows, the most common and arguably the best understood cause of software vulnerabilities. Teaching developers about buffer overflows, showing them examples of code that leads to overflows, and cataloging library calls that are prone to buffer overflows are all good ways to sensitize developers to this problem. But what are some best practices that would address not only buffer overflows, but other potential defects as well? A specific design practice may be input validation via custom typed classes. A specific verification practice may be state machine verification for session management. A specific coding practice may be language specific, checklist-based security code reviews. A specific tool may be a static code analyzer that scans the code for potentially unsafe library calls. A specific testing method may be Fuzz testing. Just as important as defining best practices is deciding when in the secure software development process these practices should be used (process scripts), how they should be measured (in-process as well as predictive measures), and how their use can be ensured.

Once the best practices have been defined, they must be applied throughout the software development life cycle. Figure 2 shows some best practices that address security through different phases of a software development life cycle. No life cycle model is implied. For spiral, incremental, or iterative development, best practices will be cycled through more than once as the software product evolves.

Examples of SDLC best practices include security risk analysis, secure design principles (such as defense in depth, application partitioning, and least privilege), threat modeling, static code analysis, checklist based inspections and reviews, and testing methods such as Fuzz testing, Ballista, or penetration testing.

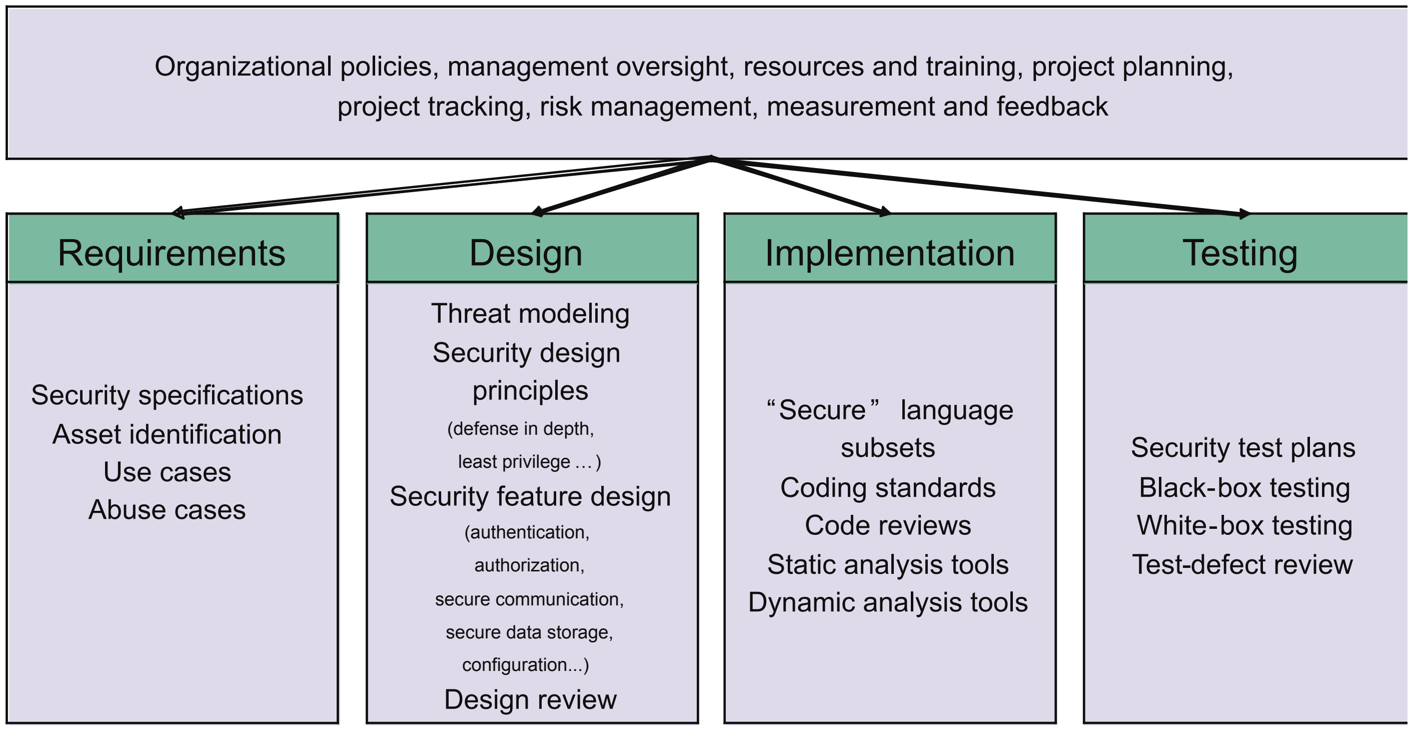
Since schedule pressures and lack of senior management sponsorship can get in the way of implementing best practices, organizational support is needed for setting security policies, providing management oversight for security activities, and for providing security training and resources. Project

continues on page 6

Developing Secure Software

Continued from page 5.

Figure 2: Addressing Security Throughout the Software Development Lifecycle


Organizational policies, management oversight, resources and training, project planning, project tracking, risk management, measurement and feedback
Requirements
Security specifications
Asset identification
Use cases
Abuse cases
Design
Threat modeling
Security design principles
(defense in depth, least privilege ...)
Security feature design
(authentication, authorization, secure communication, secure data storage, configuration...)
Design review
Implementation
"Secure" language subsets
Coding standards
Code reviews
Static analysis tools
Dynamic analysis tools
Testing
Security test plans
Black-box testing
White-box testing
Test-defect review

management is needed to ensure that security activities are planned and tracked. Risk management is needed to ensure security risks are identified, assessed, and managed.

Finally, the secure software development process should be measured to determine its effectiveness, and to determine which measures are predictive measures for latent vulnerabilities in released software.

The Team Software Process for Secure Software Development

The Software Engineering Institute developed the Team Software Process (TSP)SM as a set of defined and measured best practices for use by individual software developers and software development teams [Humphrey]. Teams using the TSP:

1) use common sense software engineering practices
2) manage defects throughout the developed life cycle
3) control the process through measurement
4) monitor the process
5) address defect prevention as well as removal
6) use predictive measures for remaining defects

Since schedule pressures and people issues get in the way of implementing best practices, the TSP helps build self-directed development teams, and then puts these teams in charge of their own work. TSP teams:

1) develop their own plans
2) make their own commitments
3) track and manage their own work
4) take corrective action when needed

The TSP includes a systematic way to train software developers and managers, to introduce the methods into an organization, and to involve management at all levels.

The Team Software Process for Secure Software Development (TSP-Secure) augments the TSP with security practices throughout the software development life cycle. The research objectives of TSP-Secure are to reduce or eliminate software vulnerabilities that result from software design and implementation defects, and to provide the capability to predict the likelihood of latent vulnerabilities in delivered software. Areas of exploration include vulnerability analysis by defect type, operational process for secure software production, predictive process metrics and checkpoints, quality management practices for secure programming, design patterns for common vulnerabilities, verification techniques, and removing vulnerabilities in legacy software.

Teams using TSP-Secure are first trained in fundamental software engineering practices. They then attend a workshop where they are introduced to

continues on page 7

Developing Secure Software

Continued from page 6.

common causes of vulnerabilities and practices they should use to address the common causes of vulnerabilities. Next, the teams plan their product development work. Along with business and feature goals, teams define the security goals for their product, and then measure and track the security goals throughout the product development life cycle. At least one team member assumes the role of Security Manager. This role is responsible for ensuring that the team is addressing security through all their product development activities.

To date, the TSP has been used by many organizations. A recent study showed that teams using the TSP produced software with an average delivered defect level of 0.06 defects per thousand lines of new and changed code, with an average schedule error of just 6%. The average productivity improvement was 78%. TSP-Secure is still under development, but an initial proof-of-concept pilot produced near defect free software with no security defects found during security audits and in several months of use.

Conclusion

Since common software defects are a leading cause of vulnerabilities, the overall defect content of software must be reduced. Next, security must be systematically addressed throughout the software development life cycle. There must be a shift in attitude from "bolting security on" after the fact, to "building security in" as the product is being developed. This requires that good software engineering practices are followed while the software is being developed, including multiple defect removal activities.

Biography of Noopur Davis

Noopur Davis is a Visiting Scientist at the Software Engineering Institute of Carnegie Mellon University, where she works with Watts Humphrey on his Team Software Process initiative. She is also Principal of Davis Systems, a company that has been providing Software Engineering Process Improvement consulting and training services for over twelve years. Noopur has been involved in the software field for over twenty years as a developer, a manager, and a consultant. Her experience ranges from real-time embedded systems software to commercial desktop products. She has launched and coached dozens of teams at major industry and government organizations.

She has authored several reports and articles on software process and software security.

Noopur has a Masters in Computer Science and a Bachelors in Electrical Engineering. She is an SEI-Authorized Team Software Process coach, an SEI-Authorized Personal Software Process instructor, program committee member for the 2003 XP/Agile Universe conference, program committee member for International Symposium on Secure Software Engineering, member of IEEE working group for draft recommended ractices for Establishing and Managing Software Development Efforts Using Agile Methods, and a member of the IEEE and the ACM.

References

[Computer World] "Congress' role in IT security debated", November 6, 2003.

http://www.computerworld.com/securitytopics/security/story/0,10801,86902,00.html?nas=PM-86902

[Davis] Davis, Noopur and Mullaney, Julia, "The Team Software Process in Practice: A Summary of Recent Results.", Technical Report CMU/SEI-2003-TR-014, September 2003.

[Kirwan] Kirwan, Mary, "The Quest For Secure Code". Global Technology, October 12, 2004. http://www.globetechnology.com/servlet/story/RTGAM.20041001.gtkirwanoct1/BNStory/Technology/

[Humphrey] Humphrey, Watts S. Winning with Software: An Executive Strategy. Reading, MA: Addison-Wesley, 2002.

[Jacquith] Jacquith, Andrew. "The Security of Applications: Not All Are Created Equal." At Stake Research.

http://www.atstake.com/research/reports/acrobat/atstake_app_unequal.pdf

[Jones] Jones, Capers. Software Assessments, Benchmarks, and Best Practices. Reading, MA: Addison-Wesley, 2000.

[Viega] Viega, Jones and McGraw, Gary Building Secure Software Building Secure Software: How to Avoid Security Problems the Right Way, Reading, MA: Addison Wesley, 2001.

The Challenge of Low Defect, Secure Software – too difficult and too expensive?

By Martin Croxford, Praxis High Integrity Systems Limited

The software industry is in crisis. A strong claim? The National Institute of Standards and Technology (NIST) reports that poor quality software costs the US economy $60 Billion per year [1]. According to the aptly named Chaos report only a quarter of software projects are judged a success [2]. Failures due to "computer glitches" are commonplace, and seem to be viewed by the public (if not by the software industry itself) as inevitable. In any other engineering discipline, or indeed any field, engineering or otherwise, this would be unacceptable. But in the safety and security sector, where reliance on correctly functioning software is increasing, and where such software is becoming ever larger and more complex, this state of affairs is unsustainable.

The challenge for the software industry has been neatly summarized by Martyn Thomas, visiting Professor of Software Engineering at Oxford University in England, as follows: "The only way to reduce costs and to keep projects within plans is to dramatically reduce the error rate at every stage in the development. If you do that, the product is not only cheaper, but higher quality: more secure, more reliable, and easier to maintain."

Thomas's emphasis on reducing errors has been backed up by recent work on behalf of the National Cyber Security Partnership, formed in 2003 in response to the White House National Strategy to Secure Cyberspace [3]. The Partnership's Secure Software Task Force reported that a primary cause of security problems is software with vulnerabilities caused by defects, or errors in software [4], and that practices which lead to low defect software are therefore to be encouraged.

One such practice cited in the report, an approach known as Correctness by Construction developed by Praxis High Integrity Systems Limited in England, has been used for over fifteen years to produce very low defect software in critical applications. As well as being low defect, such software has also proved to be highly cost-effective to develop and maintain in operation. Two examples are cited below, including a zero defect security application.

However, given that Correctness by Construction (and the other best practice approaches cited in the report) has been used successfully for a number of years, some questions arise. Why is there still so much poor quality software around? Why are these approaches not in more widespread use? Perhaps the real challenge for the software industry is to find a way of systematically applying known technology?

Before exploring these questions, it is worth summarizing the Correctness by Construction software development approach, and some examples of its results. The underlying principles are straightforward: firstly to make it difficult to introduce errors, and secondly to remove any errors as soon as possible after introduction. The key to achieving this is to introduce sufficient precision at each step of the development of the software to enable reasoning about the correctness of that step. The correctness of the software can then be demonstrated in terms of the manner in which it has been produced (the "by construction") rather than just by observing operational behavior. An analogy may be drawn with aeronautical engineering, where the demonstration of correctness during the specification, design and implementation phase is such that it is rare for a new aircraft to work incorrectly the first time operational behavior is observed!

It is the use of precision which differentiates Correctness by Construction from other approaches. While perhaps relying only on good process, many other software development approaches endure a lack of precision which makes it very easy to introduce errors, and very hard to find those errors early. Evidence for this may be found in the common tendency for development lifecycles to migrate to an often-repeating "code-test-debug" phase, which can lead to severe cost and timescale overruns. So, what kind of precision does Correctness by Construction use?

At the requirements step (a source of half of project failures [2]) a clear distinction is made between user requirements, system specifications and domain knowledge, and "satisfaction arguments" are used to show that each user requirement can be satisfied by an appropriate combination of system specification and domain knowledge. The emphasis on domain knowledge is key; half of all requirements errors are related to domain [5], yet the vast majority of requirements processes do not explicitly address issues in the domain.

At the specification and design stages, mathematical (or formal) methods and notations are used to define precisely the behavior of the software, and to model its characteristics (for example demonstrating that a multi-process design cannot deadlock). Such techniques can allow precise verification of consistency and accuracy.

At the detailed design and implementation stages, information and data flows are explicitly modeled and statically analyzed (for example, to demonstrate the separation of secure state). Where applicable, the code is written in a mathematically verifiable

continues on page 9

The Software Challenge

Continued from page 8.

programming language and statically analyzed (for example, to demonstrate the absence of run-time errors, such as buffer overflows, which are the bane of secure systems).

Correctness by Construction is cost-effective because errors are eliminated early or not introduced in the first place, dramatically reducing the amount of rework needed later in the development. The precision means that the requirements are more likely to be correct, and the system more likely to be the correct system to meet the requirements, and to work correctly in operation. Software developed using Correctness by Construction has also proved to be very cost-effective to maintain.

The results speak for themselves. Correctness by Construction was used by Praxis to develop the Certification Authority system to support the MULTOS multi-application smart card operating system developed by Mondex International (now part of Mastercard) [6]. Developed to the standards of the IT Security Evaluation Criteria (ITSEC) [7] Level E6 (roughly equivalent to Common Criteria [7] Evaluation Assurance Level (EAL) 7), the system had to meet both stringent security requirements and demanding availability requirements. The system was delivered with a warranty against defects, and had an operational defect rate of 0.04 defects/kloc (thousand lines of code), yet was developed at a productivity of almost 30 loc/day (three times typical industry figures).

In the US, Praxis used Correctness by Construction to develop a demonstrator biometrics system for the National Security Agency (NSA), aimed at showing that it is possible to produce high-quality, low defect software conforming to the Common Criteria [6] requirements for Evaluation Assurance Level (EAL) 5 and above [8]. The software was subjected to rigorous independent reliability testing which identified zero defects, and was developed at a productivity of well over 30 loc/day.

More generally, Correctness by Construction delivers software with defect rates of 0.1 defects/kloc and lower; this compares very favorably with defect rates reported by Capability Maturity Model (CMM) Level 5 organizations of 1 defect/kloc [9] (see chart Figure 1).

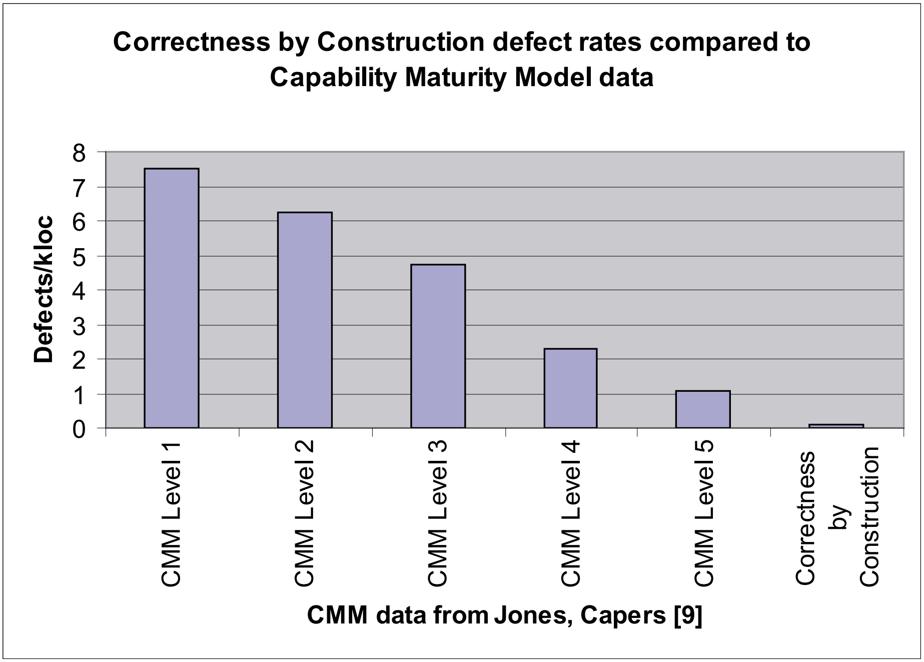


Figure 1

such as this, or papers such as those cited. Or there may be a view that such best practice "could never work here" for a combination of reasons. These reasons are likely to include perceived capability of the in-house staff, belief about applicability to the organization's product or product development approach, prevalence of legacy software which is viewed as inherently difficult and therefore expensive to maintain, or concern about the disruption and cost

So, given the apparent success of best practice approaches such as Correctness by Construction, why is there still so much poor quality software around, and why is such best practice not in more widespread use?

There seem to be two kinds of barriers to the adoption of best practice. Firstly, there is often a cultural mindset or awareness barrier. Many individuals and organizations do not, or do not want to, recognize or believe that it is possible to develop software that is low defect, secure and cost-effective. This may simply be an awareness issue, in principle readily addressed by articles

of introducing new approaches.

Secondly, where the need for improvement is acknowledged and considered achievable there are usually practical barriers to overcome, such as how to acquire the necessary capability or expertise, and how to introduce the changes necessary to make the improvements. Introducing such change may be challenging for a combination of technical, political and social reasons.

These are reasonable, common, but not insurmountable barriers, and

continues on page 10

The Software Challenge

Continued from page 9.

overcoming them requires effort from suppliers, procurers and regulators and involvement at the individual, project and organizational level. Typically, strong motivation and leadership will be required at a senior management level, where the costs to the business of poor quality (high defects, low productivity) are most likely to be experienced.

At a supplier level, a typical way forward is for organizations (and individuals within them) to take a fresh, open-minded look at what is possible by comparing current approaches to best practice and, where appropriate, adopting step-wise, prioritized improvements based on assessments of the Return On Investment. Engineering decisions on process, methods and tools need to be premised on the basis of logic and precision (for example by asking "how does this choice help me reason about my software?"), rather than on silver bullets or fashion (characterized by questions such as "how many developers already know this particular technology?").

Procurers and regulators can help by adopting an attitude of not settling for less, by demanding a warranty, by awarding contracts to organizations with the capability to deliver low-defect software, and by using contracting arrangements such as gain-share that encourage partnership and improvement.

Fundamentally, however, the main drivers for change will come from two directions.

Regulation, such as in the form of the Common Criteria [7], at least at EAL 5 and above, already requires the adoption of techniques that provide demonstration of correctness through the way software is developed. As reliance on correctly functioning software-intensive security applications increases, and where such software is becoming ever larger and more complex, the prevalence of requirements for EAL 5 and above will increase, and the software industry will need to adapt its development approaches in order to meet these requirements. The situation is analogous to the safety-critical sector, particularly in Europe, where the key safety regulatory requirements now require such approaches. This is the "stick" incentive, and there is a view that if the industry persists in producing insecure software then regulation will increase.

But there is also the "carrot" incentive. There is plenty of evidence from a range of sectors that shows that best practice software engineering produces high quality software cost-effectively. When organizations recognize that low defect software really can have through-life cost benefit (even taking into account the costs of the time and effort to acquire the capability to deliver it) then the business driver will take over – the $60 billion reported by NIST [1] is a big prize!

Perhaps the real challenge for the software industry is to recognize and eat the "carrot" before being beaten by the "stick".

About the author

Martin is Associate Director for security with Praxis High Integrity Systems Limited, a UK-based systems engineering company specializing in mission-critical systems. Martin Croxford is a chartered engineer with 15 years experience in the software industry. Martin has worked on software development projects in a range of organizations, and as a software development manager has used Correctness by Construction to successfully deliver a multi-million pound security-critical system.

Contact Details

Praxis High Integrity Systems Limited
20 Manvers Street, Bath BA1 1PX, UK
http://www.praxis-his.com/
+44 1225 466991 (switchboard)
+44 7881 516750 (cell)
martin.croxford@praxis-his.com

References

1 US NIST Report 7007.011, May 2002

2 The Chaos Report http://www.standishgroup.com

3 http://www.cyberpartnership.org/about-overview.html

4 Processes to Produce Secure Software www.cyberpartnership.org/SDLCFULL.pdf

5 Hooks and Farry, Customer Centred Products, Amacom, 2000

6 Correctness by Construction: Developing a Commercial Secure System, Anthony Hall and Roderick Chapman. Published in IEEE Software January/February 2002 pp 18-25. http://www.praxis-his.com/pdfs/c_by_c_secure_system.pdf

7 Information about ITSEC and the Common Criteria may be referenced from http://www.cesg.gov.uk/site/iacs/index.cfm

8 Fourth Annual High Confidence Software and Systems Conference Proceedings, National Security Agency, April 13-15 2004

9 Jones, Capers: Software Assessments, Benchmarks, and Best Practices. Reading, MA: Addison-Wesley, 2000

Enhancing Customer Security: Built-in versus Bolt-on

Glenn Schoonover CISSP MCSE, Microsoft Security Solutions Specialist

Introduction

Can you really bolt on security after the fact? I don't think so, at least not effectively. That is a question that software developers and security specialists have been discussing for quite some time and with the increasing number of vulnerabilities and the reduction in number of days between vulnerability and patch the best answer is to get it right the first time. At Microsoft there have been a number of significant changes in the past 3 years to address the problem of building software that is secure "out of the box" and resistant to attack even if unpatched.

What is the Problem?

One of the problems with building secure software is spelling out the requirements for the developers as early as possible. "Programmers can be taught to avoid creating buffer overflows and other well-known vulnerabilities found in commercial software," said Lawrence Hale, speaking at this year's FOSE conference on government technology. The problem is that, historically, most developers did not spend much time worrying about buffer overruns nor did they do threat modeling against applications except in very tightly controlled government environments. If they did consider the potential for a threat they were often not trained in writing secure code. An example I like to use is the URL injection exploit where a hacker can force a buffer overrun by inserting an exceptionally long character string. While I was not present those many years ago when Mosaic, Netscape, and, later, Internet Explorer were first being coded, I'm pretty sure that none of the developers ever stopped to consider what would happen if someone did insert an extra long string into their browser forcing a buffer overrun. This was new territory and people tended to trust each other when conducting business on the nascent Internet. The result was an attack path that individuals with malicious intent could use to run executable code on an unsuspecting user's machine. Writing secure code is not a challenge that is unique to Microsoft. All software vendors are faced with the challenge of building secure products, but as part of their Trustworthy Computing and Security Mobilization Initiatives Microsoft is doing something about it. The goal of our Security Mobilization is to address five key issues: 1) Build Security into our products, 2) Address Customer Pain, 3) Demonstrate Leadership, 4) Mobilize the company, and 5) Provide Security and Assurance for computer services and products that are built on Microsoft Products.

Security Philosophy: Past and Present

Until recently Microsoft's philosophy has been to build products that were easy to use and that worked seamlessly across the platform. This meant that many services were enabled by default when the operating system was installed. For example, in Windows 2000 Server, the Internet Information Services are installed by default, set to start automatically, with the Internet Printing ISAPI filter enabled. Security was often thought of in terms of "features" such as IPSEC and EFS (Encrypting File System). While this gave system administrators and home users alike the ability to run a wide range of applications with minimum intervention, it did nothing to enhance security. In the Department of Defense it took many hours of testing and evaluation to develop configuration templates that would allow organizations to meet our Certification and Accreditation requirements. System administrators would lock themselves out of the operating system because they did not understand the impact that turning off a service would have and many times the only way to recover was to reinstall the OS from scratch.

With the implementation of Trustworthy Computing, security has become the number one priority. Default installations aimed at ease of use are now not always sufficiently secure, but, going forward, security in Microsoft's products will take precedence over ease-of-use.

For instance, in Windows Server 2003, IIS6 is turned off by default. It will need to be specifically chosen for installation, and when installed will only serve up static HTML pages by default. All other functionality (ISAPI filters, Active Server pages capabilities) must be turned on by the administrator after installation. Also, the Outlook Security Patch functionality, introduced as a download for Outlook 2000, is now built-in to Outlook XP and 2003. This security patch blocks access to potentially dangerous attachments, and warns when programs try to send mail on the users' behalf.

Microsoft has committed unprecedented resources to achieving the highest level of security possible in all of our products. The goal is to become the leader in the industry both in terms of product security, and in response to security issues that arise.

Trustworthy Computing

In 2002, Bill Gates announced the Trustworthy Computing Initiative. This was the first step in a 180 degree turn in building secure products. Success with Trustworthy Computing (TWC) is not going to be an easy task. It will take several years - perhaps a decade or more, before technology is trusted.

continues on page 12

Enhancing Customer Security

Continued from page 11.

The initiative is predicated on four key pillars:

- Security: Operating systems and applications must be resilient to attack; confidentiality, integrity and availability of data and systems are protected, enabling customers to safeguard critical information.
- Privacy: Products and online services adhere to fair information principles, while protecting the individual's right to be left alone.
- Reliability: Ensuring systems and services work the way customers expect; dependable, performing and available when needed.
- Business Integrity: Open, transparent and responsive with customers, with an internal focus on excellence in our decision-making and processes.

Goal: To be everyone's trusted supplier of secure, private, and reliable computing.

Security is a core tenant and Microsoft is committed to building software and services to help better protect our customers and the industry.

Commitments

At the Worldwide Partner Conference in October 2003, Steve Ballmer announced Microsoft's commitment to "Build software and services that will help better protect our customers and the industry."

In developing and refining our approach to security over the past few years, the largest set of stakeholders that have influenced us has been our customers. Security sometimes seems too simple a term for the many aspects of business and information technology that it touches. Even just looking at security from an IT viewpoint, we want to protect networks, systems, data, processes and users. For each of those areas, people, processes and technology are necessary to manage the security business risk.

The security of our customers' computers and networks is a top priority for Microsoft. Security is an industry-wide issue and although there is no one solution, our approach to security spans across both technology and social aspects. In technology, we're focused on:

- Building greater isolation and resiliency into the computing platform.
- Providing customers with the latest and most effective advanced updating methods.
- Enabling new business scenarios through integrated authentication, authorization and access control options.
- Improving quality by enabling engineering excellence.

Progress

The first product to ship as part of the Trustworthy Computing Initiative was Windows Server 2003. We focused on making the product secure by design, by default, and in deployment. This represents huge progress on security, and the processes we use have begun to win recognition in the industry and even awards. In the area of "Secure by Design," we made a $200M Investment in security engineering covering tools, training, and the process of software development. We instituted better developer accountability - each line of code is owned by a particular developer who is responsible for ensuring security compliance. We developed pervasive threat-modeling techniques and automated code analysis to analyze the design. Another key development is shipping Windows Server 2003 in a "Secure by Default" mode. The product uses locked-down configurations so that only the features you need are enabled, reducing the attack surface to less than half of what it was in NT 4.0. IIS 6.0 is turned off by default. We implemented a stronger security policy, access control list defaults and new "low privilege" service accounts. Windows Server 2003 is also the first product to ship "Secure in Deployment." We improved the power and simplicity of Security Management Tools & Services, including software restriction policies. Secure communications (VPN/Wireless) is now easier to deploy with IEEE 802.1X protocol support, and integrated certificate services with auto-enrollment. There is greater breadth of Patch Management Solutions within and outside of the product, including Software Update Services (SUS) 2.0, and we offer much more extensive Prescriptive Guidance so system administrators can easily get information on how to deploy the product securely.

These security engineering practices have been recognized by organizations such as RSA and the SANS Institute who have given Microsoft awards on our training, tools, and product update investments. In short, with the degree of customer engagement, early production deployments across all customer segments and workloads and the measures of quality, especially security, Windows Server 2003 is a product that can be deployed today, without waiting for a service pack.

RSA Industry Innovation Award

As members of Microsoft's elite Secure Windows Initiative team, Michael Howard and David LeBlanc published "Writing Secure Code" (now in its' second edition) to provide software developers with a better understanding of the processes and practices needed to produce sound software code. Howard and LeBlanc's book is the cornerstone of the security training programs developed during the implementation of the Trustworthy Computing initiative. During the Windows security push, product development halted for more than two

continues on page 13

Enhancing Customer Security

Continued from page 12.

months as Microsoft software developers attended, and then implemented, mandatory security training, all based on the "Writing Secure Code" book.

Thousands of product developers and testers from across the company have now been trained in writing secure code as part of the Trustworthy Computing Initiative. Since being introduced internally at Microsoft, "Writing Secure Code" has become the definitive security resource for software developers and engineers at Microsoft. In addition, "Writing Secure Code" is being adapted into textbook format for university computer science courses by Addison Wesley. The success of Howard and LeBlanc's book and curriculum underscores the industry's need for secure coding guidelines and the importance of educating developers about the value of secure software in today's computing landscape.

SANS 2003 Information Security Leadership Awards

Microsoft earned recognition in three categories of SANS 2003 Information Security Leadership Awards, including automated patching and training programmers to write safer code. Red Hat also was recognized for automated patch notification.

http://www.computerworld.com/securitytopics/security/story/0,10801,79164,00.html.

Microsoft won three of the awards - demonstrating that its Trustworthy Computing Initiative is beginning to bear fruit:

- The Award for Leadership In Automated Updates for Microsoft's automated patching service (for Windows XP and Windows 2000 SP3 and above) that helps protect users who are not security experts and for the Update Server that allows security experts inside organizations to test patches and then release them for automated patching of all systems managed by the Update Server, both locally and remote.
- The Award for Leadership in Security Training of Software Developers for Microsoft's nascent program of requiring all Microsoft software developers to become familiar with common security errors made by programmers and how to avoid them.
- The Award for Leadership in Testing Software for Security Vulnerabilities for Microsoft's extensive automation of the software testing process.

What are these changes? The Security Development Lifecycle is the process that is used internally at Microsoft to build more secure software. This is a sophisticated process, with threat modeling, audit, testing and signoff stages, coupled with developer education and tools. At Microsoft, we have trained over 13,000 engineers in the rigorous process. (See Figure 1)

Security review

Once the product design is understood, the specs complete and the threat models are done, it's time to have the design reviewed. The product group should set aside a day or more for such reviews. At this meeting (taking a day at a minimum), component owners will present their architecture and the security implications, threats and countermeasures pertaining to their component. The team will provide experienced feedback and, if need be, the product group makes adjustments to the product.

Develop and Test

The purpose of the Security Days is simply to keep everyone on their toes, and to provide updated education and security analysis. In the past, many groups held such "bugbashes," but the focus should not be simply on finding bugs, it should be to educate, and attempt new attack techniques and methods on an ongoing basis. If you give people the time to do this they will find new issues.

Security Push

A security push occurs at beta time and is a team-wide stand down to focus on threat model updates, code review, testing and documentation scrub. Note that the push is not a quick fix for a process that lacks security discipline; it is simply a concerted effort to eradicate bugs before ship. Note, in the short-term, a security push is a length milestone.

Security Audit

Once the end of the project draws near, a very important question must be asked, "from a security viewpoint, are we ready to ship." The only way to answer this question is to have an end of project security audit. The process is well understood – the three main analysis points are: (1) Have the threats changed? (2) Perform a root cause analysis of incoming security vulnerabilities that require code modifications in the current code base. Why were they missed? What needs changing? (3) Penetration work; The Secure Windows Initiative (SWI) (and outside contractors and the product team) review default settings, attempt to compromise the system.

Security Response

You can only design, write and test for the security issues you know today; no matter how rigorous the process security, issues will arise simply because the threat landscape changes each week. Because of this, each team needs a group of people to handle security vulnerabilities as they are discovered after the product has shipped. The team must

continues on page 14

Enhancing Customer Security

Continued from page 13.

focus on addressing the vulnerabilities found, and also on performing a root cause analysis on each vulnerability so as to find and modify potential vulnerabilities proactively – before they are also found in the field. The team must meet common standards for response time, quality, patch packaging and release.

Challenges Ahead

At Microsoft, we are thinking about the "big picture" of security, and working to help customers in a variety of ways. First and foremost, we remain deeply committed to building software and services that will help better protect our customers and the industry. Our goal is to build the most secure software we can, while still building products that customers will want and be able to use. Beyond that, we are taking steps to help protect our customers in a world where vulnerabilities are inevitable and the threats are evolving. This means investing in new technologies; investing in training, guidance and communications to help our customers get the expertise they need; and partnering with industry leaders, customers, governments, and law enforcement to address the challenge.

Biography

Glenn Schoonover CISSP MCSE, is currently a Security Specialist with Microsoft. He is responsible for developing and executing on the Infrastructure and Security strategy for the Federal District and spends most of his time helping government customers build a secure, connected infrastructure. Prior to arriving at Microsoft he was the Director of Security for a global Internet Service Provider and Chief of Network Security for the Army at the Pentagon. A 1986 graduate of the United States Military Academy at West Point, he is an authority on network security architecture, vulnerability assessments and intrusion detection with experience using a wide variety of commercial and open source tools.

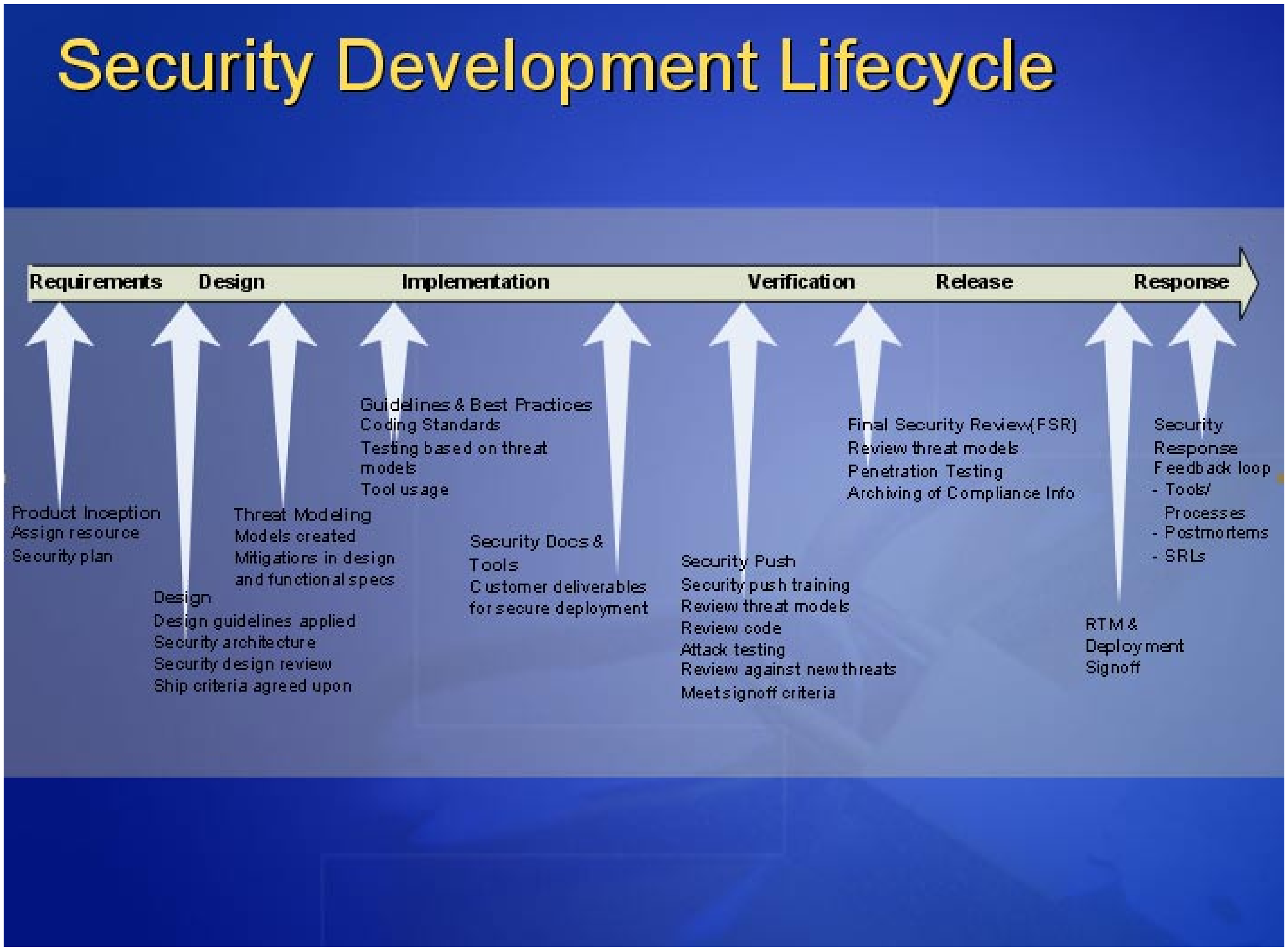


Figure 1. Security Development Life Cycle

Software Development Security: A Risk Management Perspective

By Ellen Walker, DACS Analyst

A synopsis of the Government Accountability Office (GAO), (formally the Government Accounting Office) report to congressional requesters titled "Defense Acquisition: Knowledge of Software Suppliers Needed to Manage Risks", (GAO-04-678), published in May 2004.

The Department of Defense (DOD) is concerned about the expansion of opportunities for exploiting vulnerabilities in defense weapon systems software that may result from increased reliance on prime contractors who, in turn, are outsourcing the development, implementing reuse, using COTS, and acquiring software. Additionally, contractors are growing through acquisitions, mergers, and a general trend toward globalization.

Concurrent with this software development paradigm shift, we are seeing increasing attempts by foreign entities to access U.S. technology and information, and countries and organizations hostile to the United States focusing on information warfare.

Do we know who is actually developing the software used in our weapons systems programs? Is there a significant risk resulting from the expansion of suppliers and the unknowns relating to the origins and security of the actual developers and the respective development environment? DOD thinks there is a risk that it needs to be identified and managed at the program level and that knowledge of all suppliers needs to be available for use in source selection.

Figure 1 depicts how supplier expansion is occurring and how it may encompass foreign involvement in the development process.

The spider-like image also conveys the complexity involved in identifying and tracking all suppliers and, specifically, sources of foreign involvement. The shaded oval identifies the current scope of control from the perspective of the Program Office. A solid, managed relationship exists between the prime contractor and the Program Office, but the remaining activity and information, which is primarily contractor driven, is essentially not visible to the Program Office.

continues on page 16

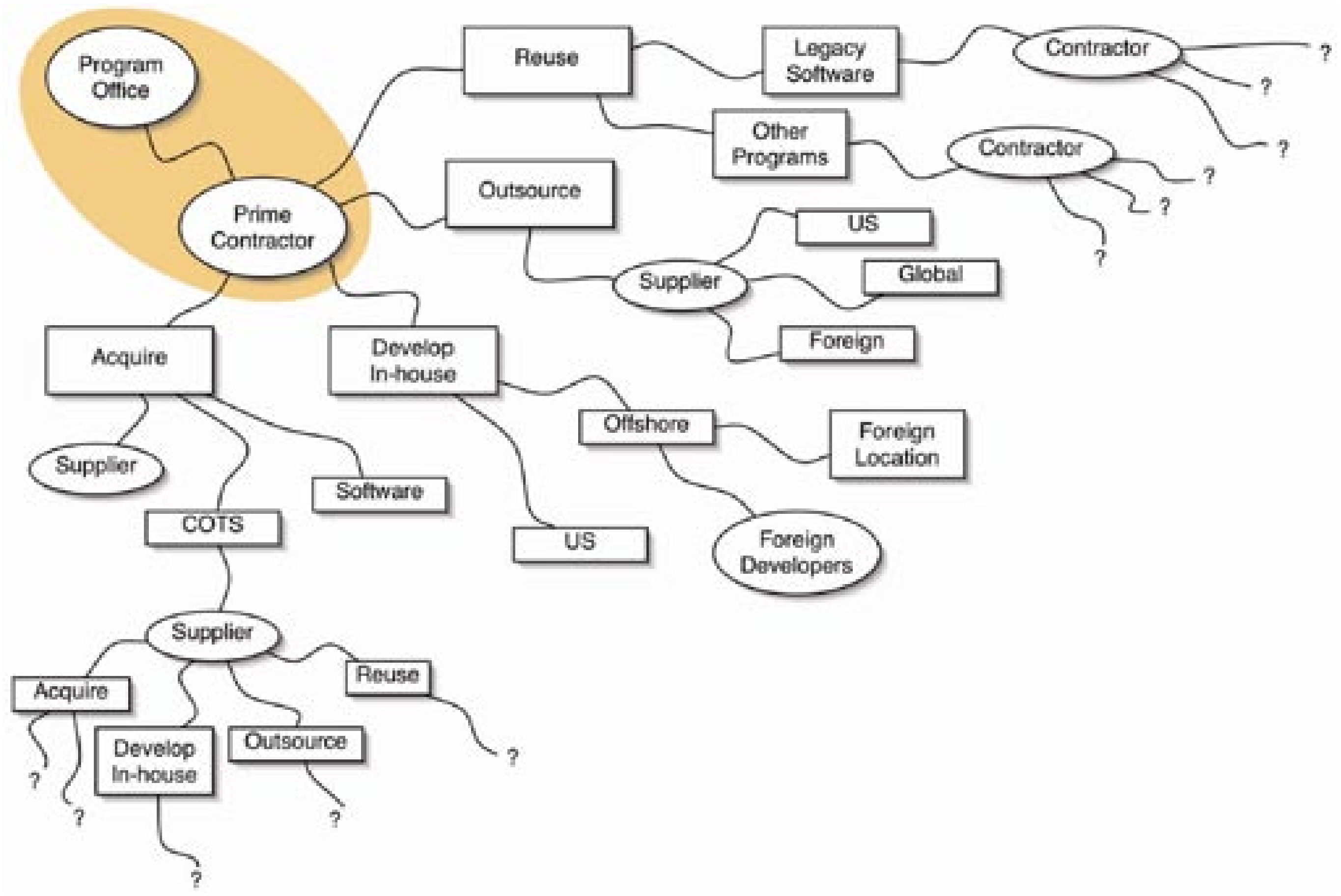


Figure 1. Scope of Supplier Expansion and Foreign Involvement

Software Development Security

Continued from page 15.

To address DOD concerns, Congress asked the GAO to examine and report on DOD's efforts to identify and manage the risks associated with foreign involvement in software development in individual weapon systems programs.

For this study, conducted from April 2003 to May 2004, GAO selected 16 weapon systems varying in age and capability; reviewed relevant DOD guidance, policies, regulations and procedures; met with experts from the SEI and the weapons engineering community; and reviewed or solicited information from the Program Offices and their respective prime contractors. Appendix I of the GAO report provides further details of the scope and methodology of this study.

Summary of GAO Findings

GAO found that software security issues in general, and the risk associated with foreign involvement in particular, are taking a back seat to the main topics of focus on weapon systems programs – performance, cost and schedule. Reasons for this tend to fit into the following categories:

- Lack of policy to address the risk of foreign involvement
- Lack of communication among organizations who possess knowledge of foreign suppliers
- Lack of prioritization of software security relative to issues of cost, schedule, and performance
- Lack of clear accountability for addressing software security related risks

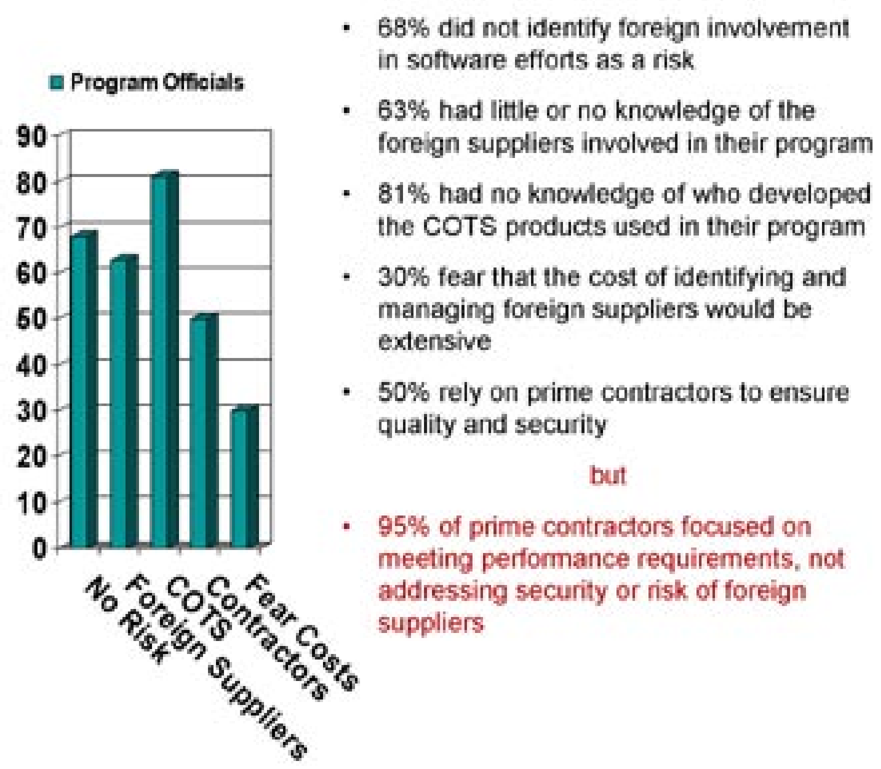
Figure 2 presents some of the quantifiable key findings with respect to the actions and viewpoints of the program officials. In general, program officials lacked awareness of foreign involvement, in either COTS, or their custom software. Consequently, they did not view any risk associated with foreign involvement as significant.

They relied on the competence of their prime contractors to ensure quality and security and make good decisions about subcontractors, but few of the programs included specific software security requirements in their contractual agreements with the prime contractor. In the absence of specific requirements to address security risks associated with foreign involvement, the contractors are not dealing with it, electing instead to focus on meeting the specified contractual requirements.

Those programs that did identify software security as a risk focused on operational threats (e.g., limiting foreign access to software development facilities and denying foreign access to software code), not insider threats that might come from foreign involvement in software development.

GAO found that DOD policy and guidance is not currently addressing the issues of software development security and adopting a risk strategy for foreign involvement. Security policies for weapon systems software focus primarily on operational threats, not insider threats such as the insertion of malicious code by software developers. Additionally, security procedures that are in place tend to be applied after the software suppliers have been selected and, thus, do not provide the manager the opportunity to evaluate whether the risks associated with using a supplier are acceptable.

Some officials noted that acceptance testing for reused and COTS software limits its focus to proving functionality and, thus, closes the door to supplier information for those products.



- 68% did not identify foreign involvement in software efforts as a risk
- 63% had little or no knowledge of the foreign suppliers involved in their program
- 81% had no knowledge of who developed the COTS products used in their program
- 30% fear that the cost of identifying and managing foreign suppliers would be extensive
- 50% rely on prime contractors to ensure quality and security

but

- 95% of prime contractors focused on meeting performance requirements, not addressing security or risk of foreign suppliers

Figure 2. GAO Key Findings

continues on page 17

Software Development Security

Continued from page 16.

GAO reported several situations in which knowledge of foreign involvement exists at some level, but is not routinely shared with the Program Office, either because there is no requirement to do so, or because the knowledge is acquired by other agencies relative to other functions, such as the export licensing process. Contractors request approval from the State Department, but the State Department does not automatically refer the application to DOD or the Program Office.

Some additional insights from the study are as follows:

- Although we know that practices like peer reviews and dedicated software testing can uncover malicious code and minimize defects, 50% of the programs made decisions about what code to test based on the risks and benefits to the functionality of the system, not on security. Experts agree that comprehensive testing (every line of code) to ensure complete security is perhaps physically impossible and would require immense resources.
- 75% of the programs reported using the Technology Assessment/Control Plan and/or the Program Protection Plan (documents that address the release of information to foreign governments through cooperative programs and military sales), but these documents do not provide specific information on suppliers who will be performing the work.
- 69% of the programs reported using the Defense Information Technology Security Certification and Accreditation Process (DITSCAP) to address general software security; however, this process does not govern contractors in cases where the requirements were not included in the original contract. In addition, the DITSCAP process bases its requirements on the program manager's assessment of risks. Therefore, unless the program manager has identified foreign software development as a risk, the process will not address it.
- Better software development practices alone (such as those represented by the SEI CMM levels of maturity) may reduce defects and improve overall software quality, but cannot be expected to address malicious software development activities intended to breach security.
- Program managers are encouraged (under the blanket of using sound systems engineering practices) to develop open software systems architectures, use COTS products, and make incremental improvements through code reuse. However, all of these practices have potential for introducing malicious code from unknown software development sources.

GAO Recommendations

The GAO concluded the DOD needs to take steps to ensure that software security is an integral element in decision-making and that program managers mitigate risks accordingly. They recommended that the Secretary of Defense take the following three actions to address risks attributable to software vulnerabilities and threats:

1. Require program managers (working with others as necessary) to define software security requirements, including identifying and managing software suppliers, and then communicate the requirements through the prime development contract to ensure that they are used in selecting suppliers
2. Require program managers to collect and maintain information on software suppliers (including software from foreign suppliers) and use the information to assess changes in supplier status and to adjust program security requirements
3. Require the Office of the Assistant Secretary of Defense for Networks and Information Integration (OASD-NII) and the Office of the Undersecretary of Defense for Acquisition Technology and Logistics (OUSD-ATL) to work with other organizations to ensure that weapons program risk assessments include attention to software development risks and threats.

DOD's concerns, in response to this report, are that the recommendations place too much responsibility on the program managers, and that insider threats are not limited to foreign suppliers. DOD believes that program managers should be able to rely on external resources to gain threat information on suppliers, and that formulation and oversight of security practices should be a collaborative function among several offices. Furthermore, a centralized information repository on software suppliers (including but not limited to foreign suppliers) is necessary because the cost of collecting and maintaining this information would require resources and assets beyond the scope of individual program managers. Perhaps identifying, tracking, and maintaining intelligence on security risks of software suppliers is best done at the DOD level so that it can be shared among the programs. Threat analysis, which drives the development of security requirements, should be carried out at the subsystem, system, and system-of-systems levels and not be limited to the scope, expertise, and resources of the individual program managers. [Source: GAO Report Appendix II, DOD Comments to the Recommendations]

Reading the actual report begs questions such as the following:

- Is the issue of malicious code potentially being inserted into a

continues on page 18

Software Development Security

Continued from page 17.

software component really a threat here and now?

- How far could someone get with this before it is discovered?
- Are foreigners the only people who could do this?
- What would it take to test for malicious code insertions (insider threats)?
- Whose job is it to verify that all the software used in a weapon system does not represent a security risk?
- What level of risk is acceptable (if any)? And at what cost?
- What do we need to know about software suppliers, or the software development environment, in order to be able to thwart such threats?
- How could we be sure that the information we collect on suppliers is, in fact, valid?

These and many more questions, collectively, hint at the complexity of the issue and its proposed solutions. Are GAO's recommendations simplistic given the realities of the issue? If implementing software security measures requires continuous tracking of all suppliers of all software products and results in enormous costs, who decides what the priority of security should be relative to overall cost, schedule and software capability requirements? Perhaps these questions have wetted your appetite for reading the report in its entirety. It is available for download at http://www.gao.gov/new.items/d04678.pdf

Author Contact Information

Ellen Walker

Data and Analysis Center for Software (DACS)

Ph: 315-334-4936

E-mail: ellen.walker@itt.com

User Comment

By Bob Ferguson,
Carnegie Mellon University

I was reading part of the July 2004 Software Tech News. I noted your criticism of "software testing as an art." I do understand why you would say that -- that certain people claim they are artists as a way of saying they do not care to have a disciplined process.

I claim that nearly every artist does have a disciplined process. I have observed my wife doing watercolor painting. Whenever she changes paper, brush or paint, she does some number of test drawings. She has to have a detailed understanding of how the materials work together and master the techniques she will use. It is important to understand the mistakes that are inherent in the process. Then it is possible to execute so that the mistakes do not compromise the end product.

If she happens to be in a rush and skips the test drawing, she inevitably has to throw away the first version -- usually before it is finished.

Michelangelo and Leonardo Da Vinci were famous for making many detailed sketches in preparation. They mixed and tested materials they would use in their frescos. I also know a couple of sculptors who like to work with new and different materials. They spend a great deal of time learning how to work with the new material before attempting a product.

Let's not demean artists by claiming we can be creative without a process. We should realize that true artists do follow a disciplined process and calibrate their work. Good testers would do likewise.

Bob Ferguson
Sr. Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University

Lessons Learned

By Thomas McGibbon, DACS Director

The four articles in this issue of Software Tech News have provided excellent guidance and a wealth of information about "Secure Software Engineering" from a development, management, supplier, and acquisition perspective. Being a software engineer myself and given the increased importance now of security and trustworthiness of software intensive systems, my perspective in reading these articles is to understand how what I perceive as "best" practices in software engineering are impacted by security issues. Here are the highlights of what I learned:

- From a development and lifecycle perspective:
  - Need to significantly reduce defects induced and improve methods for detecting software defects throughout the lifecycle. Correctness by Construction is a rigorous methodology which results in very low defect software (<0.1 defects/ksloc). TSP-Secure provides a set of defined and measured best practices for low-defect Secure software development.[1, 2]
  - Need to understand common causes of vulnerabilities and focus on defect reduction techniques for defects that lead to or cause vulnerabilities.[1, 3]
  - Security must be built-in to the software development lifecycle with appropriate checkpoints and reviews.[1, 2, 3]
  - Need to define a set of best practices that development teams can use. Correctness by Construction and TSP-Secure provides best practice approaches.[1, 2]
  - Need to sensitize designers, developers and testers to security issues through training.[1, 3]
  - Tools are available to detect some security vulnerabilities.[1, 3]
  - Best practices for developers and testers includes threat modeling, Fuzz testing, Ballista, penetration analysis static code analysis.[1, 3]
  - Developer accountability helps to ensure security compliance.[3]
  - Correctness by Construction achieves significant defect reduction through rigorous requirements analysis, use of formal design methods and information flow models for design, and verifiable code development (when needed).[2]

- From a management perspective:
  - >90% of all vulnerabilities are caused by defects resulting from inadequate, normal software engineering methods.[1]
  - In building a business case for secure software engineering, need to consider (add) costs of fixing and releasing patches from a supplier, acquirer, and consumer perspective. Not addressing security from a supplier perspective could impact customer satisfaction and result in lawsuits.[1, 2]
  - Need senior management vision, leadership, support, and oversight of implementation of security policies and best practices.[1, 2, 3]
  - Security measures need to be planned, measured, and tracked.[1]
  - Program managers should collect and maintain information on suppliers used to perform software development.[4]

- From a supplier perspective:
  - Suppliers need to ship software where default settings are secure.[3]
  - Perform a security audit prior to release.[3]

- From an acquirer's perspective:
  - Acquirers, suppliers, and program managers need to identify and manage risks associated with foreign involvement in development of software (including COTS) for weapon system programs.[4]
  - Acquirers need to communicate security requirements through prime development contracts.[3, 4]
  - Acquirers should demand software warranties, award contracts to organizations that deliver low defect software, and provide contract incentives for partnership and improvement.[2]
  - Change, in reality, will come from regulations and financial incentives.[2]

[1] See "Developing Secure Software" by Noopur Davis

[2] See "The Challenge of Low Defect, Secure Software" by Martin Croxford

[3] See "Enhancing Customer Security: Built-in versus Bolt-on" by Glenn Schoonover

[4] See "Software Development Security: A Risk Management Perspective" by Ellen Walker

The DACS Gold Practice Initiative:

- Promotes effective selection/use of software acquisition & development practices
- Defines essential activities/benefits of each practice
- Considers the environment in which each practice is used
- Addresses the timeliness of practice benefits
- Recognizes interrelationships between practices that influence success or failure
- Contains quantitative and qualitative information
- Includes discussion forums, case studies and survey opportunities
- A continually evolving resource for the DoD, Government, Industry and Academia
- Free to use/free to join

Current Gold Practices:

- Architecture-First Approach
- Binary Quality Gates at the Inch-Pebble Level
- Commercial Specifications and Standards/Open Systems
- Ensure Interoperability
- Formal Inspections
- Integrated Product and Process Development
- Model-Based Testing
- Plan for Technology Insertion
- Statistical Process Control
- Track Earned Value

Learn More About the DACS Gold Practice Initiative:
http://www.goldpractices.com

Become a Registered Member!
http://www.goldpractices.com
/registration/register_step1.php

Data & Analysis Center for Software
P.O.Box 1400
Rome, NY 13442-1400

http://iac.dtic.mil/dacs

DACS
Data&Analysis Center for Software

The first 50 people to send in a completed survey will receive a FREE DoD/IT Acronym CD from the DACS.

This valuable CD-ROM contains over 9,000 Department of Defense and Information Technology acronyms. There are hundreds of acronym lists available but none are as well done as this CD AND specifically targeted towards DoD and Information Technology. This unique-shaped CD-ROM plays in your computer's regular, hub-mounted, CD drive. You'll use this great resource over and over again. It's FREE, just for filling out our brief survey on the next page!

Fold Here

Data & Analysis Center for Software (DACS)

http://iac.dtic.mil/dacs/

Fold Here

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 120 ROME NY

POSTAGE WILL BE PAID BY ADDRESSEE

DATA AND ANALYSIS CENTER FOR SOFTWARE
ITT INDUSTRIES
775 DAEDALIAN DR
ROME NY 13449-0139

Software Tech News Subscriber Survey

1. Which volume of the Software Tech News did you receive? **STN 8:2 Secure Software**

2. When did you receive the newsletter? (month/year) ______________________

3. How satisfied were you with the CONTENT of the newsletter? (Article Quality)
☐ Very Satisfied ☐ Satisfied ☐ Neither Satisfied nor Dissatisfied ☐ Dissatisfied ☐ Very Dissatisfied

4. How satisfied were you with the APPEARANCE of the newsletter?
☐ Very Satisfied ☐ Satisfied ☐ Neither Satisfied nor Dissatisfied ☐ Dissatisfied ☐ Very Dissatisfied

5. How satisfied were you with the OVERALL QUALITY of the newsletter?
☐ Very Satisfied ☐ Satisfied ☐ Neither Satisfied nor Dissatisfied ☐ Dissatisfied ☐ Very Dissatisfied

6. How satisfied were you with the ACCURACY of the address on the newsletter?
☐ Very Satisfied ☐ Satisfied ☐ Neither Satisfied nor Dissatisfied ☐ Dissatisfied ☐ Very Dissatisfied

7. Approximately how much of the newsletter do you read?
☐ The entire issue ☐ Most of the content ☐ About half the content ☐ Briefly Skimmed ☐ Didn't Read

8. Would you read this newsletter in an E-mail newsletter format?
☐ Definitely ☐ Probably ☐ Not Sure ☐ Probably Not ☐ Definitely Not

9. How did you request the product or service?
☐ Phone Call ☐ E-mail ☐ DACS Website ☐ Subscription Form Other ______________________

10. Would you recommend the DoD Software Tech News to a colleague?
☐ Definitely ☐ Probably ☐ Not Sure ☐ Probably Not ☐ Definitely Not

11. What topics would you like to see this newsletter devoted to?

____

Comments (Optional)

____

____

Contact Information (Optional*)

Name: ______________________ Position/Title: ______________________

Organization: ______________________ Office Symbol: ______________________

Address: ______________________

City: ______________________ State: ______________________ Zip Code: ______________________

Country: ______________________ E-mail: ______________________

Telephone: ______________________ Fax: ______________________

Functional Role: ______________________

Organization Type: ☐ Air Force ☐ Army ☐ Navy ☐ Other DoD______________________
☐ Commercial ☐ Non-Profit ☐ Non-US ☐ US Government ☐ FFR&D ☐ Other ______________

*Note: You must give us your address to receive the CD.

About the DoD Software Tech News

Data & Analysis Center for Software
http://iac.dtic.mil/dacs/

DACS
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: dacs@dtic.mil
URL: http://iac.dtic.mil/dacs/

Article Reproduction

Images and information presented in these articles may be reproduced as long as the following message is noted:

"This article was originally printed in the DoD Software Tech News, Vol. 8, No. 2. Requests for copies of the referenced newsletter may be submitted to the following address:

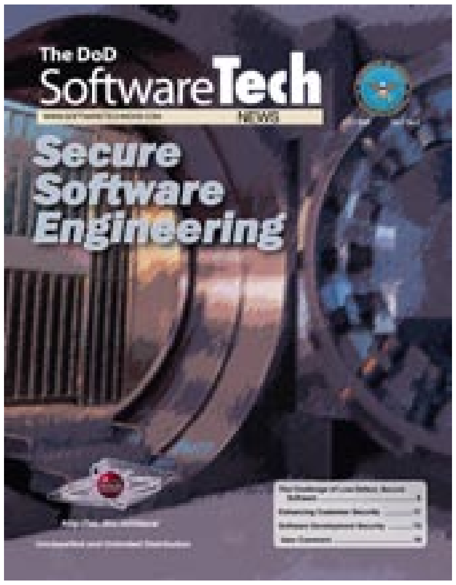
Philip King, Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil

An archive of past newsletters is available at www.SoftwareTechNews.com.

In addition to this print message, we ask that you send us three copies of any document that references any article appearing in the DoD Software Tech News.

Cover Design by Joseph Barbaccia, ITT Industries



About This Publication:

The DoD Software Tech News is published quarterly by the Data & Analysis Center for Software (DACS). The DACS is a DoD sponsored Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC). The DACS is technically managed by Air Force Research Laboratory, Rome, NY and operated by ITT Industries, Advanced Engineering and Sciences Division.

To Subscribe to this Publication Contact:

Phone: 800-214-7921
Fax: 315-334-4964

E-mail: news-editor@dacs.dtic.mil
Web: www.dacs.dtic.mil

Distribution Statement:

Unclassified and Unlimited

STN Vol. 8, No. 2

In This Issue

Advertisement

The DoD Software Tech News is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to the DoD Software Tech News in paper copy, the Tech News will also be placed on the Data & Analysis Center for Software's website (http://iac.dtic.mil/dacs/), exposing your product, organization, or service to hundreds of thousands of additional eyes.

Interested in learning more? For rates, layout information, and requirements contact:

Philip King, STN Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: (800) 214-7921
Fax: (315) 334-4964
E-mail: news-editor@dacs.dtic.mil

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Return Service Requested

PRSRT STD
U.S. Postage
P A I D
Permit #566
UTICA, NY