CrossTalk

Nov / Dec 2014 The Journal of Defense Software Engineering Vol. 27 No. 6

SOFTWARE ENGINEERING TOOLS

AND THE PROCESSES THEY SUPPORT

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
DEC 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk, The Journal of Defense Software Engineering. Volume 27, Number 6. Nov/Dec 2014	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS/MXDED,CrossTalk,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

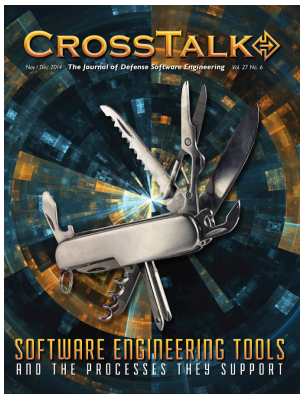
14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**44**	

TABLE OF CONTENTS

Departments



Cover Design by Kent Bingham

Software Engineering Tools and the Processes They Support

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Article Coordinator Heather Giacalone
Managing Director David Erickson
Technical Program Lead Thayne M. Hill
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-777-9828
E-mail Crosstalk.Articles@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: Office of Cybersecurity and Communications in the National Protection and Programs Directorate.

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

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

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

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

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

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

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

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

CrossTalk would like to thank NAVAIR for sponsoring this issue.

Tools and processes can be very powerful enablers of Software Engineering. The right tools in the hands of capable people who make effective use of those tools can make the difference between project success and failure. Well thought-out, effective processes can help avoid mistakes and costly rework. A good fiddle is important, but it takes a good fiddler to make the best music.

Some of the challenges we face when selecting and using tools and processes are:

- Whether to centralize or distribute their use. For example, code analyzers can be expensive to procure, require time and training to use effectively, and produce a lot of false positives. Having a small, highly-skilled team to perform such code analysis could actually be advantageous to each program office and software support activity doing this on their own.
- Balancing strict adherence to process with fostering innovation and creativity and doing things expeditiously. While we certainly want to do things right the first time, we also want to enable such things as Rapid Deployment Capability (RDC) to get the product our Warfighters need as quickly as possible.
- Keeping our tools and processes up to date and relevant. We work in a very dynamic environment, so should always be looking critically at our tools and processes to ensure they appropriately complement our programs.

As with any resource, tools and processes can be used effectively, used ineffectively or even misused. A sharp saw, as Dr. Stephen Covey pointed out, is easier to use and cuts better than a dull saw. A chain saw is more effective than a hand saw, but also has the potential to do more damage and cause injury. Adherence to process can assure the job is done right but also cause a bottleneck. It is up to us as software practitioners to select the appropriate tools and processes, and use them effectively to get the most benefit.

I hope as you read the articles about tools and processes in this issue, you think about how best to adapt them to your programs.

Al Kaniss
Software Engineering Branch Head
NAVAIR

Through These Fields of Destruction: The Tools Versus Process Wars

Jitka West, Natural SPI, Inc.
Michael West, Natural SPI, Inc.

Abstract. Most certainly, the scenario plays out a hundred times a day in organizations world-wide: People want to get work done more efficiently and effectively and then, almost inevitably, the tools camp and the process camp suit up, arm themselves as best they can with their respective "facts," and then go do battle against one another, fighting for precious little budget and resources.

When left to fight the tools-versus-process war in perpetuity, and usually on a field lacking any real leadership, not only does neither side win; both sides lose. The two camps have mutually exhausted the money which – at some level in the enterprise – comes from the same pot. The tools or technology and the processes are implemented independently of each other and, thus, do not align with or support each other. The parts of the process which are reliant on using the technology, and the parts of the technology which are reliant on following process, are all "black box" to the user. The lack of process and technology integration leaves the users inefficient and ineffective, and maybe even confused and angry, perhaps even more so than before. The battlefield is scorched … trust is broken, careers are damaged, morale is diminished, and the seeds of retribution for future battles are deeply sewn.

In this article, we – business performance consultants Jitka West and Michael West, two people who have often had to wear the blue peace-keeper helmets in such conflicts – describe a better way. Technology and process are not opponents, they are allies which, when joined under a shared vision and mission, can significantly help workers do their jobs better and faster. Now that we've described the disastrous results of the tools-versus-process wars – which many of you can relate to – in this article we'll explore the root causes for this common situation and describe changes you can make in your organization to bring greater peace and performance to the organization.

How Did We End Up Here?

Think back on all the times in your career when you've been either scripted as a soldier in process-versus-tools battles … or ended up as collateral damage. What did those battles have in common? How did I – a process person – end up disliking and distrusting my techno-geek office mate? Or why did I – the cool tools guy – stop going to lunch with that process woman down the hall?

In a very real sense, the on-going animus between process and technology is symptomatic of the way our modern organizations are structured. We are organized in departments and teams within which we exercise knowledge, skills, and experience that are homogenous with the others in our department or team. We love what we know, and we don't love what we don't know, so we tend to get along well with those in our department. For the sake of efficiency, organizations divide their people into sub-cultures such as software developers, system engineers, executives, human resources, information technologists, and project managers. The cultural boundaries between our team or department and other teams are as real as the cubicle walls and doors between our team and the others. Our organizations are not organized by mission or vision. Even the integrated team (e.g., the IPT) is an artificially imposed construct, and its members are rarely truly integrated with everyone working toward a common purpose.

Thus those of us who live in technology (e.g., the IT department) love tools, and we think that technology is always the solution to every problem. The process people, who usually live in operations or quality management, love process and think that process is always the solution. We love what we know.

As CMMI® Institute-Certified Lead Appraisers, we see a lot of interesting things when we lead appraisals, including process-oriented people hold onto a certain horribly inefficient way of doing things beyond all rationale. We've see quality assurance auditors use a physical (piece of paper) checklist to conduct an audit (with a pen or pencil), after which they often inaccurately transcribed the audit results into a Word document. The word document looked like a form, but didn't have any functionality of a Word form. After all of this, the QA person then manually transferred defect data (e.g., counts, types, etc.) into a spreadsheet for analysis and charting. When we suggested all of that work could not only be made more efficient but also more accurate if all of the work was done in a single Excel file, the response we received was, "well, this is how I do it." The person had started his career using a form from a typewriter and a pencil. The technology (MS Office products) to make the individual both more efficient and effective had been pervasive in the work place for years, but he only loved what he knew.

Another dynamic that contributes to the tools-versus-process paradigm is – hmmm … how do we say this? – technology is sexy and process is boring! It is as true as it is hard to admit. Let's face it – our whole modern existence is full of glossy, shiny, sexy technology, from our i-things to our home entertainment systems to our automobiles – which are, of course, thinly disguised rolling entertainment systems. But when was the last time you saw a Super Bowl commercial selling you a process?

Also, tool development and implementation looks cheaper than process development, even though it rarely is. An executive can wrap her head around the finite price of a SharePoint server, Team Foundation Server (TFS), an enterprise project portfolio management tool, or a corporate metrics tool. The real total cost of ownership (TCO) is rarely investigated or considered because that requires the hard work of consideration. The cost of the tool out of the box is, if nothing else, a number that we and accounting can deal with. On the other hand, the cost of process development, process improvement, or process management is ambiguous, and most managers and leaders will

retreat from ambiguity to seek safety and comfort in numbers … even inaccurate numbers.

Both tools and processes are forms of codified human knowledge, but because codifying human knowledge in software and systems gets all the press, and because it's easier to touch and feel technology, managers and leaders are just more comfortable accepting technology as the "solution."

A Better Way

So are we just to accept our lot and resign ourselves to the endless waste resulting from the tools-versus-process wars? Can we improve the business of performance improvement in a way that everybody wins, including our shareholders and the tax payer? Yes we can. This section provides you with an approach that we have partially implemented in both Government and commercial sector organizations. Although we cannot claim dramatic success just yet, we have at least gotten the two camps to call a truce.

Get Organizational Performance Improvement Out of the Silos

In my (M. West) 2013 book, Return On Process (ROP): Getting Real Performance Results from Process Improvement [1], I observe that the modern organization is really a system of systems, made up of three sub-systems: 1) people, the social system, 2) technology systems, and 3) process systems. I also make the experience-based argument that improvement or change to any one of the three sub-systems will effect collateral changes to the other two sub-systems, whether or not we plan those changes, and whether or not we observe or measure those collateral changes.

When we change technology, whether the initiative is insertion of wholly new technology, a platform migration, a systems integration, or simply an upgrade to existing tools, the change will affect process. Process often defines how people use technology to get work done, so if changes to technology affect those interactions and work practices, then the processes will need to be changed to accommodate the technology changes.

Changes to technology also effect collateral change on the people sub-system. Ironically, technology changes sometimes do not yield the intended performance improvement because workers are not trained to use the technology effectively and efficiently. Improvement in technology almost always requires a correlating change to worker knowledge and skills.

Changes to processes – presumably to improve process performance – also do not occur in a vacuum, no matter how much the initiative is pursued within an organizational silo. The performance of defined processes more often than not incorporates the use of technology. (Think configuration management processes, requirements management processes, testing processes, to name just a few.) Thus, improvements to processes often require concomitant changes to the tools and systems – primarily the human interface aspects of the tools – to support performance of the changed processes.

Yet even when the collateral effects of changes and improvements to one sub-system within the organization are understood, improvement initiatives are rarely planned and executed in an integrated approach. The CIO or CTO executes a technology change and then, sometime later, the process people realize that the defined processes no longer work with the tools, and have to execute a catch-up "improvement." Or the COO leads a process improvement initiative only later to have people complain that the way they use technology doesn't fit with the defined processes. The naïve process developer (which I once was) will proudly proclaim that he wrote his processes to be "tool agnostic." The learned and experienced process developer knows that is wrong, and just the opposite is desirable: Develop the processes such that they inextricably integrate and work with the use of the tools and systems.

Improving technology and improving process in silos will separately have less positive results on organizational performance than if the two improvements are planned and executed as one integrated improvement initiative. This won't happen accidentally. Executives and leaders in the organization need to step outside of their domain and work with leaders of the other silos. The unifying mission cannot be process improvement alone or technology improvement alone, rather it must be the higher calling of organizational performance improvement.

Perhaps you – the reader – are not operating at the executive level or in a leadership role. Yet you were not hired to just blindly follow orders. (Remember! If all you do is say "yes" to your boss, one of you is redundant!) No matter what level of the hierarchy you're operating in, when you find yourself getting involved in some kind of change initiative such as a process improvement, it is in your long-term self-interest to reach out to your colleagues in the other silo and say, "hey, this initiative I'm involved in is going to affect your work too … will you help me elevate the planning of this change to our bosses?"

Understand How the Tool and the Process Contribute to Performance

This topic is too easily and too often unjustly treated – and dismissed – by tired clichés: "There's no silver bullet;" There is no magic wand." However, quotidian quips only mask and suppress the questions and the conversations that should be embraced. Maybe that tool isn't a silver bullet, but how much fire-power does it bring to performance improvement? The reengineered process certainly possesses no magic, but what will it contribute to improved performance, or will it even be predictably repeatable? More importantly, how can the use of the tool and the process together help people perform more effectively and efficiently in their jobs?

The answers to those two questions are intuitively obvious in certain situations. For example, most modern day software workers would find it unthinkable to control the configuration of software source code manually, thus the pervasive use of software configuration management tools. Sure, some process discipline may still be required of the people using such tools, but the tools do the "heavy lifting" in managing and controlling the integrity of the software. In other cases, a tool – even if it existed – would be either useless or a hindrance. Imagine a tool that tries to facilitate and control the interactions of people in a meeting … how well would that work?

However, in most engineering and management activities, the answer to the question – tool or process? - which brings the most value to the activity requires more in-depth analysis. Such analysis should never start with a foregone conclusion lest the analysis is biased.

Peer Review Activity	Process Contribution	Tool(s) Contribution
Planning/scheduling the peer review	Defined processes can include stakeholder involvement plans that identify the personnel involved in the peer review of a particular product, eliminating the need to guess or replan for a particular peer review item.	MS Outlook provides lots of functionality such as being able to view the schedules of the planned reviewers to select their common availability, or to view and reserve a conference room, or an on-line conference.
Conducting the peer review	A peer review process or procedure is useful in defining peer reviewers' functional roles, and the steps for conducting the peer review. Checklists based on standards for the items being reviewed make reviews more effective in terms of identifying and capturing defects	Technology, such as SharePoint work-flow or Google Docs, can facilitate the on-line entry of reviewer input without anyone having to leave their desk for a meeting. Databases with web-enabled front-end checklist forms can make collecting peer review input, and the aggregation of that input, more efficient and more accurate. Online conferencing and document sharing can reduce the cost and burden associated with people having to physically attend a peer review.
Collecting and compiling peer review results	Defined defect categories and severity levels bring deeper understanding to the defects being identified than a simple count of defects.	Tools, such as databases or even Excel workbooks, can make the aggregation of quantifiable data (e.g., numbers and types of defects) far more efficient and accurate than people doing manual counting and calculations.
Disposition of peer review defects and comments	Defined standards (for the work product being reviewed) make it easier for both the author/owner of the peer reviewed item and the reviewer to reach objective agreement on the disposition of the reviewers input.	
Analyzing and reporting peer review results	A process can include measurement specifications and their representations to enable the consistent and accurate reporting of peer review information and analysis.	Tools such as Excel make the representation of data relatively easy, efficient, and accurate compared against humans manually deriving statistics and their graphical representations.

Table 1. Example of evaluating the contributions of tools and process to peer reviews

To demonstrate how such an analysis might be conducted, we'll use the example of planning and conducting peer reviews and collecting and analyzing data from peer reviews. These are all activities which are commonly performed in many organizations in both the defense and commercial sectors. Table 1 illustrates how to evaluate the contributions, and strengths and weaknesses, of both tools and processes to peer reviews.

As you can see in the simple analysis in Table 1, there are some peer review activities, such as the human interaction to disposition peer review input, for which having defined standards and processes contributes more to the performance of the process than do tools. In other activities, such as conducting a peer review, tools are the relatively stronger contributors in terms of performance efficiency and efficacy. Yet in other activities such as analyzing the results of peer reviews, defined processes and standards and technology are equal partners in terms of the value they each bring to the performance of the process.

In some Agile development environments the relative separation between tools and processes is so seamless as to be almost subconscious to process performers. If you ask an Agile software developer what she does in a daily stand-up meeting, she'll describe the process: "I share what I completed yesterday, what I will complete today, and any barriers to completing my tasks." If you then ask her to describe an image of the daily stand-up meeting, she'll probably describe the stand-up board, a tool that depicts task burn-down chart, risks, etc. The act of writing an epic or a user story is a human mental process, but most Agile teams wouldn't think of trying to plan and track their development tasks to implement the user stories without a tool to automate the process such as Team Foundation Server (TFS). In these environments, it is difficult to articulate the relative contributions of the tool versus the process, and it is not useful to do so when the developers perceive that they cannot do their work effectively and efficiently without either.

Performance Improvement is for the Performer

In terms of balancing the investment in tools or processes, another important – yet often overlooked – consideration is who will be performing the process and using the tool? You can spend many hours developing a well-articulated process description, but if the performer is a tool lover, he will find fault with the defined process, always having a bias for using a tool. Tool-oriented people will always posit the challenge, "Why do we need a defined process … our tool does that?"

Conversely, the intended user or performer may have an aversion to technology, preferring instead to perform work by following written instructions or a process. In these cases, it will be a constant challenge to convert that person simply by assuring them that the tool enforces performance of the process.

Given that different individuals could be performing the same or similar work, those of us involved in process development and management work can ill afford to alienate one group of people to make another group happy. When developing or redeveloping process, our challenge is to abandon our preconceived notions and predilection, and use the powers of inquiry and active listening to elicit from our end users – the performers – not only what they need, but also what they want.

In a current Government contract, our company is supporting a software development team in a process definition project. For years this team has ostensibly been performing the organization's defined standard processes, which takes the traditional form of a lengthy, text-based narrative document, and which is based on a traditional RUP-based waterfall life cycle model.

However, in practice the team has not been performing the organization's defined standard processes. This software team is comprised of very tech-savvy people, and they have adopted Agile/Scrum methodologies for software development. Prior to our working with this team, there was a history of dissonance at best and conflict at worst between the software team and the team respon-

sible for process and process improvement in the IPT. Our contract included supporting both the process team and the software team, and serving both sets of customers with candor and integrity.

We approached this somewhat delicate situation with two vectors. First, we helped the process team come to the realization that they simply did not have – nor would ever likely have – sufficient resources to continue developing or updating all the standard processes that had been deployed over the years, and that the various groups within the IPT needed to take ownership of their own processes and the care and feeding of those processes. This part of the solution was relatively easy since process group was already realizing that they played a more strategic role in the development of the IPT.

In working with the software team, as consultants we knew from the start that we had to play on their home turf. Culture trumps strategy … every time. We knew that they would reject us outright if we proposed that the process development project be executed in a waterfall approach, and we knew that we had to apply Agile methods to process development just as they do with software development. This has not been easy because there is not a plethora of published information about applying Agile to process development. As of this article, it is still premature to claim success, but we are already seeing results in terms of getting tool-oriented people accepting standard processes on their own terms.

One of the ways we accomplished this was how we approached Sprint 0. In Sprint 0, we took the approach of identifying functional requirements ("Rnnn") for the process system that they – the software developers – both need and want. Each functional requirement is then broken down into epics, ("Ennnn") which would later be further broken down into user stories and finally into development tasks. Figure 1 shows a partial example of the Sprint 0 functional requirements and epics for the group's defined processes.

In our consulting practice, we have also worked with numerous clients to develop their process systems using a traditional waterfall approach. The process is the product, and we employ most if not all of the proven project management and engineering practices that you would use to develop a product or a system. We facilitate meetings to elicit and define requirements for the process, develop a process architecture and design, develop the process system and verify that it meets the requirements, and then test or validate the process.

Requirement.Epic	Description	Complete	Comments
R001	Defined process user stories that will be used to design a process representation that it is adoptable and adaptable by SEA process users.	5/2/14	
R001.E0001	Process user can access the process via any web-enabled device.	4/22/14	
R001.E0002	Defined process should be published as a web application; users can navigate through the defined process using typical (standard) web navigation techniques.	5/2/14	
R001.E0003	Process user can access process assets (forms, templates, checklists, guides, repository, etc.) from the task or step being performed, and for which those assets are needed.	4/22/14	
R001.E0004	Process user can access/contribute/share "best practices," example work products and lessons-learned related to the task or step being performed.	4/22/14	For example, process user can contribute to a wiki about performance of a process, sub-process, task, or asset, or can upload sample "best" work products.
R001.E0005	Process users can easily and intuitively transition between performing the defined process and accessing the tools (e.g., TFS) and infrastructure that enable process performance.	4/22/14	
R001.E0006	Process users will be able to verify that they are producing the outputs (documents, information) that are intended to be produced by performing a task or step in the process.	4/22/14	
R001.E0007	Process users will be able to clearly know the range or the extent to which their performance of the defined process can be tailored. Process users will know the boundaries for deviating from the defined process.	5/2/14	
R001.E0008	Process users will be able to perform the defined process more effectively and efficiently, and produce higher quality work products and products, than that which was enable by previously defined processes (e.g., the SEA Handbook).	5/2/14	
R001.E0009	The defined process will enable process performance measurement.	5/2/14	

Figure 1. Sprint 0 requirements and epics/user stories for a process system

✓	**Watch for dark clouds**. Recognize the early signs of potential tools-process conflict before fighting breaks out, and take steps to prevent conflict. Listen for hallway talk about the purchase of a new system or the start of a process initiative. Go talk to the proponents of the tool or the process and, through inquiry, convince them to involve ALL the relevant stakeholders.
✓	**Be the change you want to see**. If you truly seek long-term value for the organization, don't get caught up in the arguments about tools versus processes. Stay neutral and be everyone's favorite arbitrator.
✓	**Practice inquiry over advocacy**. Your point will rarely be accepted because it's YOUR point. Ask questions of people who are advocating a tool or a process so that they start to question their own motives and why they feel so strongly and single-minded about their point of view. Even in passion there is a place for reason and critical thinking.
✓	**Use facts and question assumptions**. When someone starts advocating a tool or a process with the phrase, "Ninety percent of people …," politely ask, "What's your source." In business, it's human nature to want to quantize our beliefs to make what we're saying sound factual, even when what we're saying has no basis in fact. In tools versus process debates, try to steer people down the road of seeking more real facts for their positions. Question assumptions and factually sounding statements that are not really factual.
✓	**Love learning**. Whenever I start thinking that we know a lot about a topic, I realize that it's just because I don't get out enough! Never assume that you have all the facts, all the points of view, or the total picture. Ask yourself, "What do I not know about this topic, and how do I find out what I don't know?"

Table 2. Checklist for the tools-process peacekeeper

A Checklist for a Tools-Process Accord

One of our favorite tools is the process asset called a checklist. Checklists – no matter what their form or format – are a powerful tool for codifying things that you should do or should have done without trying to retain that do-list in your head. So we think it's appropriate to provide you – the reader – with a checklist (Table 2) for becoming an effective peace-keeper in bringing the tools and the process camps closer together for the benefit of everyone in the organization.

Disclaimer:

CMMI® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHORS

Jitka West is a CMMI Institute-Certified Lead Appraiser and a Lean Six Sigma Black Belt. After working many years in information technology, data warehousing, and business intelligence, Jitka joined Natural SPI and has served as an industry thought-leader in process system design and development. When not developing advanced processes that help people reach higher levels of performance, she enjoys knitting with kitties in her lap and traveling with her husband, Michael.

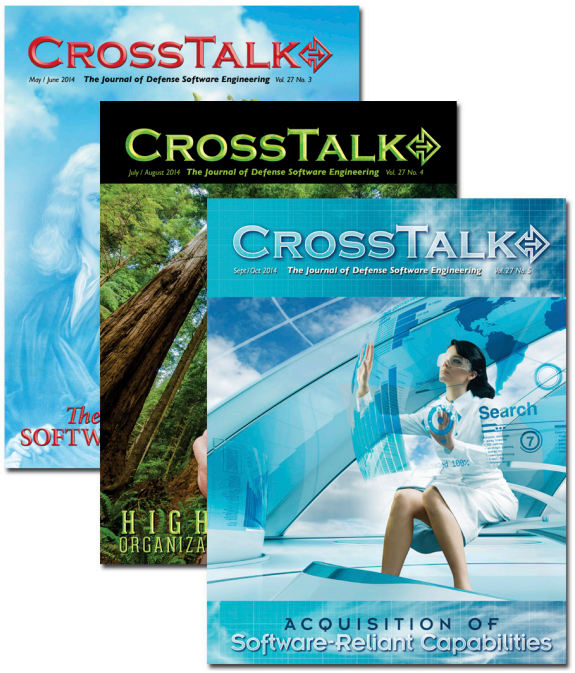
Phone: 435-729-9101
E-mail: Jitka@naturalspi.com

Michael West is a CMMI Institute-Certified Lead Appraiser and is co-founder of Natural SPI, Inc., a consultancy with 12 years of success in helping its clients achieve their performance improvement goals. Michael is the author of two books: 1) Real Process Improvement Using the CMMI (2004, CRC Press); and 2) Return On Process (ROP): Getting Real Performance Results from Process Improvement (2013, CRC Press). Mr. West is a novice beekeeper and enjoys travel with his wife, Jitka.

Phone: 435-901-4295
E-mail: michael@naturalspi.com

REFERENCES

1. West, Michael. Return On Process (ROP): Getting Real Performance Results from Process Improvement. New York: CRC Press, 2014. Chapter 1, "Real Performance Improvement."



CALL FOR ARTICLES

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

What Is Software Engineering?
May/Jun 2015 Issue
Submission Deadline: Dec 1, 2014

Data Mining and Metrics
Jul/Aug 2015 Issue
Submission Deadline: Feb 1, 2015

Supply Chain Assurance
Sep/Oct 2015 Issue
Submission Deadline: Apr 1, 2015

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

Evolving Postmortems as Teams Evolve Through TxP

Brad Hodgins, NAVAIR

Abstract. TxP was created by NAVAIR to help non-software teams define their own customized team processes, similar to the TSP process enjoyed by software teams. It takes time for a team to define a customized team process, and during this time, postmortem analysis needs to evolve as the team's customized process evolves. These postmortems will give the team the insight required to see where it has improved, and where it needs to focus its future improvement efforts on its way to standing up its customized team process.

The success that NAVAIR [1] software teams have had using Team Software Process (TSP) had led some of their parent organizations to desire to achieve the same level of performance from their non-software teams. In response to these requests, NAVAIR got together with the Software Engineering Institute (SEI) to develop an approach based upon the same fundamental principles behind TSP (i.e., plan your work, work your plan, and analyze your data), but with no specific domain discussed in the methodology. NAVAIR came away from that development effort with Team Process Integration (TPI), which requires only a day or two of classroom training [2]. This TPI training can be taught to a team, software-related or not, to get them off and running with the process as soon as possible. Please note that since the TPI training does not include all of the original, software-specific principles of Personal Software Process (PSP) [3], software teams are encouraged to take the PSP training sometime in the near future to accelerate their path towards realizing the full benefits of TSP.

All the performance data shown in the figures are from real NAVAIR teams applying TSP or TxP while producing their products or providing their services.

What is TxP?

While TSP is a process containing specific activities that a software team would follow to produce high-quality products in the domain of software, TxP is a set of generic activities (Table 1) that can be tailored to create a process a team would follow to produce high-quality products in the domain of "X." As an example, a system integration test team would use TxP to create the Team Test Process (TTP), and a requirements team would use TxP to create the Team Requirements Process (TRP).

The Path to Applying TxP

While software teams have the option to take two weeks of PSP training and immediately become familiar with how to develop software using PSP and TSP methodologies, other teams outside of software do not have this training available. Some of the metrics applied by TSP teams were identified only after the SEI had analyzed thousands of sets of process data from PSP-practicing individuals. They analyzed the patterns in the process data to discern what differentiated those process data which led to high-quality products from those that resulted in low-quality products. An example of one of these metrics is the appraisal to failure ratio (A/FR), which is calculated by dividing appraisal costs (time spent in design and code reviews) by failure costs (time spent in compile and test). The SEI found that programs with A/FRs greater than 2 have significantly fewer defects discovered in unit testing than those programs with A/FRs less than 2. This is important, since fewer defects found in unit testing usually means fewer defects in the product delivered to the customer. Instead of waiting for SEI to compile enough data to repeat this kind of analysis for the system integration test domain, a system integration test team could use TxP as a checklist for what abilities that team needs to stand up so that it can maximize its chances of doing system integration testing as well as a TSP team does software development.

Even with the TxP checklist in their hands, a team cannot simply 'fill in the blanks' on day one and stand up their Team Test Process (TTP). Some abilities, like planning on how many mistakes will be made by the team in producing a test procedure, can only be performed after the team has determined a) which mistakes count in the domain of system integration testing, b) what units to use to measure the size of a test procedure, and c) how many mistakes

TxP Planning Activities
- Project and Management Objectives
- Team Goals and Roles
- Project Strategy and Support
- Overall Plan
- Planned sizes and rates used to compute times
- Quality Preparation
- Planned Defects Injected/Removed
- Planned quality indicator values are acceptable
- Balanced Plan
- Project Risk Analysis
- Launch Report Preparation
- Management Review
- Launch Postmortem

TxP Working Activities
- Logging time
- Logging defects
- Tracking EV
- Using PROBE in Planning phase
- Entering actual sizes in Postmortem phase
- Defining Defect Types
- Using Review checklists
- Holding periodic team meetings
- Following an agenda during team meetings
- Performing/reporting on assigned roles
- Reviewing action items
- Reviewing assigned goals and risks
- Maintaining project plan and workbook

TxP Analyzing Activities
- Evaluate plan vs. actual schedule hours
- Evaluate plan vs. actual component hours
- Evaluate plan vs. actual component sizes
- Evaluate team performance vs. goals and quality plan
- Evaluate plan vs. actual quality of components
- Update planning data for schedule hours
- Update planning data for lifecycle time-in-phase %s
- Update planning data for productivity rates
- Update planning data for defect densities
- Update planning data for defect rates and yields
- Update planning data for quality indicator thresholds

Table 1. TxP Activities list

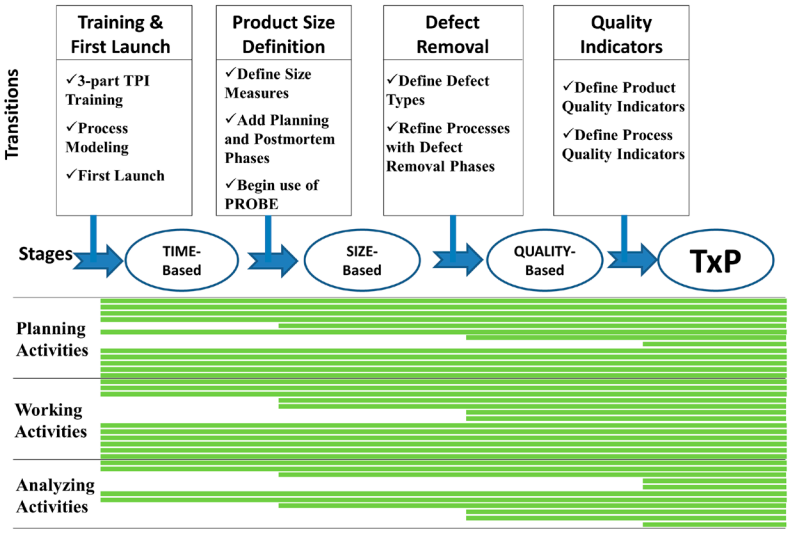


Figure 1. TxP Stages

the team makes when producing test procedures. While software teams come out of the gate with a set of orthogonal defect types identified by Watts Humphrey [4], a system integration test team will have to spend significant time logging mistakes of all kinds before being able to determine which types of mistakes should be logged. Likewise, while software teams may say with certainty that they measure the size of their software products in lines of code, the size measure for a test procedure may not be so obvious. Is it best measured in terms of the number of 'verifies' in the procedure or the number of steps in the procedure? Only after data has been logged, collected, and analyzed by the team, will the team understand which the better size measure is. Finally, only after the team has established which kinds of mistakes should be logged, and agreed on how they will measure the size of a test procedure, will they be able to measure the number of mistakes made. This kind of dependency between the activities causes a team that is starting from scratch (with only TPI training) to have to pass through a number of stages (Figure 1) before being able to completely 'fill in the blanks' and stand up its own customized TxP process.

Evolving Postmortems

Even though a team has not completely defined its customized team processes, it still has a lot of data that can be analyzed to help the team perform its job better in the next cycle. Showing a team how it has improved in some aspect is a very strong motivating factor to help the team understand how easily improvement can be achieved, and to help it embrace process improvement. Postmortems are important in sustaining a team's interest in process improvement. As a team evolves and begins to use more complex plans, more extensive postmortems become possible.

Even PSP/TSP-trained software teams have evolving postmortems. When a team is first introduced to PSP/TSP, there is a huge paradigm shift for the team members to adjust to. Even when the team is sold on TSP as the better way to develop software, there is only so much change that the team can handle and still perform its job of developing software for its customers. Then, as the team gets comfortable and more consistent at performing the primary activities of TSP (e.g., launching, logging time, logging defects, tracking progress, analyzing their data), they begin to look at executing these fundamental activities in more effective ways. The AV-8B Joint System/Software Support Activity declared TSP as their organizational standard for developing software in 2002 and were still evolving its launches and postmortems to become more efficient and effective in 2008 [5].

NAVAIR has developed a standard set of postmortem topics that can be piecewise-introduced as a team gains experience and progresses through the TxP stages on its way to defining its customized team processes. This allows a team to show management that the team is improving its performance in some aspect, regardless of whether the team is just starting with TPI, or well on its way to standing up its customized team processes. These postmortem topics are domain-independent and can be used across an organization as a standard reporting mechanism. This standardization allows organizational managers to become more familiar with the charts presented in the postmortem out-briefs by the organization's various teams, which, in turn, empowers the managers to question the teams more thoroughly on why the charts show what they show.

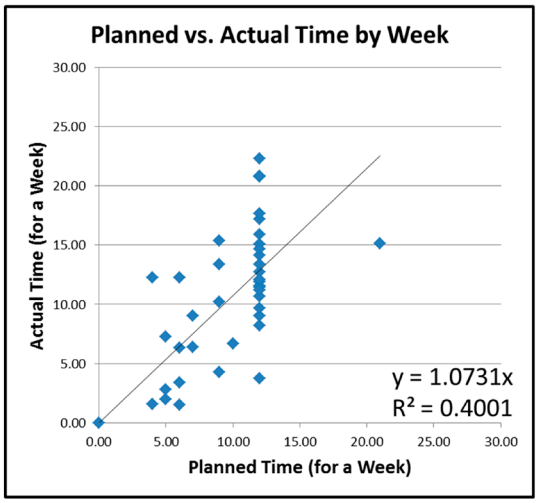
Time-Based Postmortems

The team's most consistent data at first will be time log entries. These entries contain who worked on what task, for how long, and on what date: e.g., Jaime worked on the design of the database user interface for 15 minutes on Thursday. A time log entry is made every time any individual logs any time to any task: e.g., Jaime may have half a dozen entries logged to the database user interface design task that reflect her stolen moments during the week to make progress on that design. With this information, a team can analyze the planned versus actual weekly time spent by individuals on the project (a.k.a. task time). As seen in Figure 2, the analysis can even focus on determining the average actual time logged on only fulltime planned weeks. In this example, 12 planned hours was fulltime so that weeks that contain holidays and leave days do not drive the average actual time per week down artificially. With this insight, individuals will be more attune to their personal weekly task time and will be better prepared to understand how much work they should be able to sign up for during the next cycle's planning session. The scatter chart also provides a quick check to ensure that the team member is maintaining their scheduled hours so that they do not show themselves logging time on a week they planned to be gone (this would be a data point up the left side of the chart), or being gone on a week where they planned to be working: that data point would be along the bottom of the chart.

By looking at the planned versus actual time spent on components (e.g., a user interface, a section of a requirements document, a test procedure), the team can understand its accuracy at estimating the time to create those components. The left chart in Figure 3 provides the team with general and specific information about their estimates. In this case, the value of 0.5018 in the upper left of the chart indicates that the team is only using half of the time they thought they would need, and the data points that land in the upper left and lower right regions of the chart identify specific components whose planned and actual times varied greatly from each other. The team can then discuss these components to understand what made them have actual times so different from their planned times.

As an example of this discuss, concerning the case shown in Figure 3, the team found that two data points (one in the upper left red area and one in the lower right) were Flight Testing-type components. These were two from a group of 291 Flight Testing-type components and were the only two components that were significantly off in their planned times. The team concluded that these two components were simply miss-estimated, and, while the team is still striving to improve its ability to estimate Flight Testing-type components, it can understand when one in a hundred estimates is just plain wrong. The third component (in the lower right red field) belonged to Activity D, which was a very volatile topic with many changing requirements. In this case, this component became much simpler due to a changing requirement but its estimate was not changed to reflect that shift.

By studying the planned versus actual time by type of component, the right chart in Figure 3, the team will get insight into its estimating ability by the various kinds of components it works on. In Figure 3, we see that the team is doing a great job on estimating the time to provide software-in-the-loop (SIL) testing, and landing near the mark when it comes to flight testing and ground testing, but all this is lost in the weeds when combined with the massive 100% or greater errors being realized in estimating activities A, B, C, and D. By showing these categories separately, the team can recognize its accomplishments in the first three categories mentioned and identify the need to improve the way they are estimating activities A, B, C, and D.



Planned Time	Count	Avg Actual Time	% of PT
0	1	0.0	
1	0		
2	0		
3	0		
4	2	6.9	173%
5	3	4.0	80%
6	4	5.9	98%
7	2	7.7	110%
8	0		
9	4	10.8	120%
10	1	6.7	67%
11	0		
12	21	13.5	113%
13	0		
14	0		
15	0		
16	0		
17	0		
18	0		
19	0		
20	0		
21	1	15.2	72%
22	0		

Figure 2. Task Time Charts

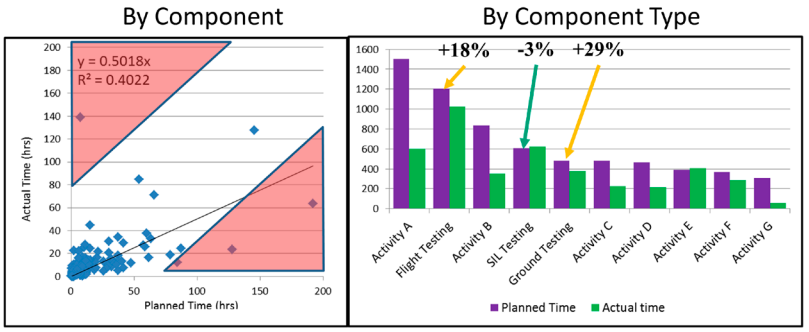


Figure 3. Time By Component Charts

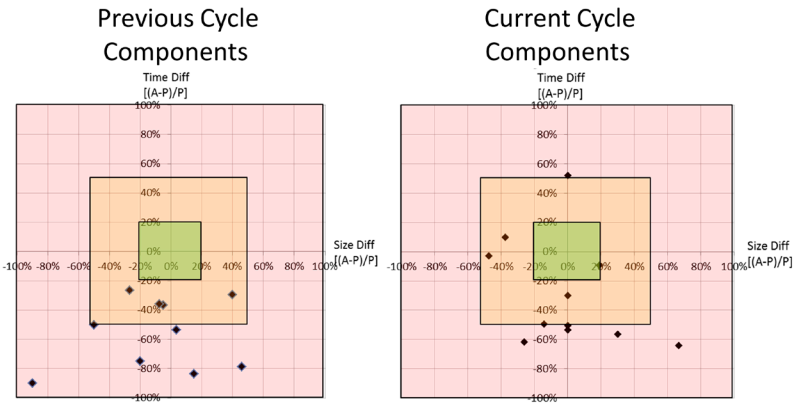


Figure 4. Size and Time Estimation Errors by Component

Size-Based Postmortems

Once the team has determined how they will measure the sizes of their various types of components and has begun to record planned and actual size data on components they have produced, then additional analysis becomes available to them. Just as TSP-trained individuals can compute their personal productivity rates, individuals on a team that logs its time and size data can compute their personal productivity rates in whatever-units-they-are-counting-with per hour. In addition, the team can utilize scatter charts, like those shown in Figure 4, to understand whether their ability to estimate times and sizes is improving from cycle to cycle.

In the example shown, all the data points are along the bottom half of the chart, showing that this team's previous cycle had a trend of consistently overestimating the time to produce a component. Along with that, the team can see that its ability to estimate sizes was mostly in the +/- 50% range, with half of those within +/- 20%. After completing the current cycle, the team can see that the improvements it made in how it estimates time have made a positive difference in that the average time estimate error has moved closer to zero. Their size estimating ability has not improved, still mostly in the +/- 50% range with half of those within +/- 20%. Now that they have made a significant improvement in their time difference errors, they can look to see if they should turn their attention to the size difference errors next, or whether they need to continue to work on improving their time difference errors.

Quality-Based Postmortems

Once consistent mistake-related data is available for analysis, e.g., mistake log entries and mistake type standards, then all kinds of metrics can be quantified concerning the quality of the product as well as the quality of the process. Metrics such as defect injection rates by phase, indicating how frequently they are making mistakes while they are logging time to a certain phase of the process, help the team to understand where the mistakes are being made in the process. With that information, the team can then discuss if there is anything they can change on how they perform that phase of the process to reduce the rate at which mistakes are being made.

Watts Humphrey injected design and code review phases into the software development process in support of the PSP principle that if the mistake is fixed as soon as possible after it is made,

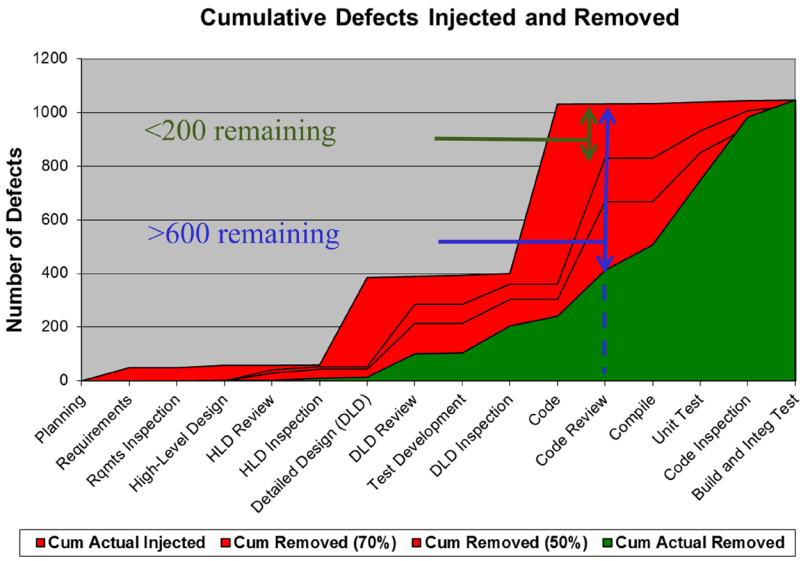


Figure 5. Defects Injected and Removed

and fixed by the person who made it, then the cost of fixing that mistake would be minimized [4]. Building a chart similar to that shown in Figure 5, the team can see where, in their process, they are making their mistakes, and where they are fixing their mistakes. In studying this type of chart, teams may recognize the need to add additional phases to their current process, or remove phases that are not providing added value, or see where they are performing better or worse than the planned performance. Comparison lines can be added to show the team where they planned on finding their mistakes to allow the team to recognize where in the process they are struggling most to meet the plan. In this example, the team needs to focus on improving its performance during code review.

TxP-Based Postmortems

Once the team has been operating at the Quality stage for some time, then analysis can start on the accumulated data to try and identify leading indicators of the quality of the product, i.e., the equivalent of an A/FR for a test procedure. This analysis can likewise be applied to find leading indicators for the quality of the process. In addition, this analysis can lead to identifying acceptable thresholds for rates and ratios which can be used to assess the quality of a plan (Figure 6 shows some of the quality rates and ratios used in TSP). Identifying these leading indicators and acceptable thresholds allows the team even more insight into ways of improving the quality of their plans, processes, and (most importantly) products. Only after the team can identify what level of process performance leads to a quality product can they then, with certainty, compare planned and actual values of these leading indicators, rates, and ratios and know which values are more desirable.

Summary

As a team's process evolves from TPI to a customized team process, the postmortem analysis of their data needs to evolve too. The focus of the analysis should be on what is value-added to the team and that analysis should help them to identify what progress they have made so far and where they need to continue to focus their attention. These analysis efforts should peak the team's interest in process improvement, but will definitely lead to improvements in planning, product quality, and communication with management.

REVIEW RATES (LOC/hr)		
Phase	Plan	Actual
DLD Review	336	829
DLD Inspection	71	136
CODE Review	147	266
CODE Inspection	60	62

RATIOS		
Phase	Plan	Actual
DLD Review / DLD Ratio	0.36	0.27
DLD / Code Ratio	0.82	0.59
Code Review / Code Ratio	0.33	0.20
Compile Defect Density (defects/KLOC)	0.00	3.15
Unit Test Defect Density (defects/KLOC)	8.86	7.81

Cost of Quality (COQ)		
Topic	Plan	Actual
% Appraisal COQ	36.8%	30.8%
% Failure COQ	19.9%	26.1%
Appraisal / Failure Ratio (AFR)	1.85	1.18

Figure 6. TSP Quality Indicators

ABOUT THE AUTHOR

Brad Hodgins is a TSP/PSP coach and instructor for NAVAIR at China Lake, California, where he coaches engineering teams in the development of high quality aviation products for the U.S. Navy and Marine Corps. He is a NAVAIR Associate Fellow and has been awarded a U.S. Navy patent for the Learning Applying Mastering Perfecting (LAMP) model for team process implementation, evaluation, and improvement. His MS in Computer Science is from Colorado Technical University.

NAVAIR
Code 414300D
STOP 6308, 1900 N. Knox Road
China Lake, CA 93555-6106
Phone: 760-939-0666
E-mail: bradley.hodgins@navy.mil

REFERENCES

1. NAVAIR is the Naval Air Systems Command. NAVAIR procures, develops, tests, and supports Naval aircraft, weapons, and related systems. For more information about NAVAIR, go to <www.navair.navy.mil>
2. Schwalb, Jeff and Brad Hodgins. Broadening the Ability to Train and Launch Effective Engineering and Service Teams. 1 September 2011. <http://www.sei.cmu.edu/tspsymposium/past-proceedings/2011/Broadening-the-Ability-to-Train.pdf>.
3. Personal Software Process (PSP) is a data-driven method of developing software that gives the software engineer insight as to where their process needs to be improved for them to produce a higher quality software product. PSP also helps to improve their ability to estimate the labor and calendar time required to produce that software product. A more thorough description of PSP can be found at <http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=5283>
4. Humphrey, Watts S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995.
5. Rickets, Chris and Brad Hodgins. How TSP Implementation Has Evolved at AV-8B. 1 May 2008. <http://www.ieee-stc.org/proceedings/2008/pdfs/BH1997.pdf>.

21st Century Cybersecurity Challenges, Comments, Solutions, and Path Forward

Dr. Bahador Ghahramani, P.E., CISM, CPE

Abstract. This scientific paper discusses the 21st Century foreseen breakthroughs in the Cloud Computing Technology and how they affect Cybertechnology and our lives.

Cloud Computing technology is moving forward constantly. Therefore, current technology may not be ready and applicable for the future. In addition to updating, the Government should also look to advanced research and development centers for new concepts and innovative principles in the field. In the future, Cloud Computing will be on-line accessed through software applications and shared information with remote server networks. This will eliminate current dependency, primarily, on tools and information located on personal computers. Cloud Computing will also become more dominant than other Internet means of communications. Simply stated, Cloud Computing will be the driving force that permits a user to communicate personal and work related activities securely through connections to servers operated by Cloud Computing sources. Cloud Computing technology is the primary driving force of social networking sites (500 million people use Facebook), webmail services like Hotmail and Yahoo mail, microblogging and blogging services such as Twitter and WordPress, video-sharing sites like YouTube, picture-sharing sites like Flickr, document and applications sites like Google Docs, social-bookmarking sites like Delicious, business sites like eBay, and ranking, rating, and commenting sites such as Yelp and TripAdvisor.

The future of Cloud Computing is bright: it will continue and effectively dominate the Information Technology (IT), Telecommunications Industry (TI), and a host of other information transactions-based technologies because it will allow users to have easy access to individualized, affordable, instantaneous, and accurate cutting-edge information. Cloud Computing will place a personal cloud in each private workstation. Cloud Computing will transform an Internet-based application that runs from smart phones instead of current software running PC. The future of Cloud Computing evolves from development of a more sophisticated desktop-cloud hybrid as a primary method of interface with information.

Background

I was asked by the United States Computer Emergency Readiness Team (US-CERT) Deputy Director Tom Baer to comment on Deputy Under Secretary for Cybersecurity Mark Weatherford's publication, "Cybersecurity Challenges and Solutions on June 5, 2012. In a formal email, Mr. Mark Weatherford states that:

"I have several significant speeches over the next two months and am developing a new narrative that includes challenging the audience (depends on audience of course) to help with the cybersecurity problem. This doesn't mean specifically help DHS, although that might be included, but big issues that young and old, hacker or non-technical, could participate in and help the global problem. Once we develop the general framework of a speech, we could use it, or portions of it, as each of us do our DHS outreach and it would go a long way to developing common themes and talking points."

In this reviewer's qualified opinion and pursuant to Mr. Tom Baer's request, the following is a list of my comments and responses to Mr. Mark Weatherford's request.

2.0 Comments

Challenge 1: The Cybersecurity threats to systems supporting Government, Industry, academic, and private citizens, on critical infrastructures, are evolving and growing.

Comments on Challenge 1: The interdependency and interconnectivity among information systems, the Internet, and other infrastructures can amplify the impact of intruders' threats that have potentially impaired operations of critical Internet-based systems, the viability of sensitive information, and the flow of commerce. Furthermore, the high-technology network's reliance on Information Technology (IT), utilizing Cloud Computing Technology (CCT), through Internet-based systems have significantly exposed the telecommunications Cybersecurity environment and resulted in unknown vulnerabilities. Unfortunately, this reliance has been severely exploited by attackers and caused potential vulnerabilities that have not yet been exploited by attackers.

Solutions to Challenge 1: As the GAO reported in January 2011, securing smart grid systems and networks presented a number of key challenges that required attention by Government and Industry. These included:

- Development of a coordinated approach to help monitor Government, Industry, and private citizens' compliance with voluntary standards. These standards should support the Federal Energy Regulatory Commission (FERC) effort that is responsible for regulating key aspects of the electric power industry, which includes adopting Cybersecurity and other standards it deems necessary to ensure smart Internet-based network functionality and interoperability.
- Development of a Cybersecurity network to help build into a high-technology Internet Cloud-based system devices. High-technology Cloud-based systems should be developed with strong security architecture and multi-layered security features. The Government should set standards, at a minimum, and provide support to Industry and private citizens when it is required.
- Development of an efficient and effective centralized information-sharing Cloud-based mechanism that is fully capable of providing Cybersecurity information to Government, Industry, and private citizens. This Cloud-based mechanism should utilize and enhance the existing Government and Industry information sharing systems and research and development centers in a safe and secure way.
- Development of a set of cost effective standards, best practices, and metrics for evaluating Cybersecurity threats, intrusions, and information sharing among Government, Industry, and private citizens.

Challenge 2: Although most of the Government and Industry Cybersecurity officials are proactively complying with the Federal Information Security Management Act (FISMA), still there is an urgent need to certify their IT security systems and encrypt their data. These officials are openly and privately stating that the security of the United States is still vulnerable to intrusions and commerce could be adversely impacted in the future.

Comments on Challenge 2: The primary reason behind Government and Industry Cybersecurity vulnerabilities is that the number of threats to the nation's infrastructures and commerce has significantly increased due to intrusions nationally and internationally. In addition, the Cybersecurity threats to our Internet-based systems have grown far more sophisticated and complex in recent years, outflanking and maneuvering traditional rigid and inflexible Government and Industry security entities.

Solutions to Challenge 2: As the Government debates its appropriate role in Federal, Industry, and private-sector Cybersecurity activities, we have to realize that it is already responsible for securing its own networks and information. This is challenging and expensive but the cost is far less than that of compromised systems. Many of the recent Cybersecurity technologies developed and utilized by Government and Industry are applicable to Federal components as well as Industry and private citizens. While our nation's defense and intelligence communities are safeguarding our infrastructures and preventing intrusions to Internet-based networks that could damage, steal, and corrupt our sensitive information, we should be vigilant in our efforts to upgrade our systems continuously. The 21st Century United States CCT relies on information technology systems and networks to control our commerce through institutional common means such as emails and data sharing. Government acknowledges that we need to take actions to adapt and upgrade modern technologies and to prevent intruders' from accessing and damaging our vital systems.

Challenge 3: The next Pearl Harbor will be a cyber attack crippling our Government components and commerce.

Comments on Challenge 3: Government should build partnerships between Federal needs, academic research, and industry solutions to protect our information and critical Internet-based infrastructures. The Government should also bring together businesses and regional research and development partners to harness the talents and creativity of subject matter experts (SMEs). In addition, it should incorporate the perspec-

tives of public and private entities to build a competitive cyber workforce that meets our 21st Century national security short-term and long-term objectives. Government, therefore, should engage all of our states' educational institutions, businesses and other invested partners in a common goal of attracting the best SMEs and training the most advanced technologists to fulfill our national Cybersecurity goals.

Solutions to Challenge 3: Perhaps the most significant changes in our culture have been through the use of Internet in our daily lives. The profound effect of Internet-based technologies has enabled us to access our bank accounts on our computers and to connect to social networks from our phones. The Internet technology proliferation and expansion has impacted all aspects of our lives and will continue influencing our decisions for decades to come. This is a paradigm shift in our way of life, culture, and thinking; however, the consequences are that serious new Cybersecurity vulnerabilities have also emerged. To safeguard our nation's intellectual property and commerce, Government needs to offer more assistance to private owners of our critical Internet-based infrastructures and .com networks.

Challenge 4: The challenge and dilemma confronting Government is how to defend critical systems while protecting our civil liberties and privacy.

Comments on Challenge 4: We need legislation to bring our nation in line with the technological realities facing us through the Internet and online communications systems. We also need to pass laws and legislations that would protect our critical infrastructures and safeguard our civil liberties because protecting our nation's infrastructures is also safeguarding personal identity and liberties of our citizens. We need to bring together Government, Industry and academic partners to harness the talent and creativity of SMEs to build a framework and then incorporate the perspectives of public and private entities to build a competitive cyber workforce that meets our national security needs. While building the Cyber workforce, Government must educate the country to define the balance of defending our critical infrastructures and protecting our civil liberties and privacy. In order to strengthen the pipeline for these efforts, Government should engage all of our states' higher-education, businesses, research and development centers, and other invested partners in a common Cybersecurity goal to keep our nation and civil liberties safe from those who intend to harm us.

Solutions to Challenge 4: Government needs to establish a technical Cybersecurity roadmap towards an enterprise services "platform." To date, the most prevalent cyber threat has been exploitation of our networks. By that, we refer to the theft of information and data from Government and commercial impacts as well as other disruptions:

- Government impact: Foreign intelligence services have exfiltrated military plans and weapons systems designs.
- Commercial impact: Valuable source codes and intellectual properties have been stolen from Industry, research and development centers, and academic institutions.
- Internet impact: Disruption of our networks by adversaries seeking to deny or degrade the use of important Government, commercial, and private networks.
- Cyber-based impact: The most dangerous Cybersecurity threat confronting our nation is cyber-based destructions, where sophisticated technologies are used to cause physical damage to our Government, Industry, and private citizens.

Challenge 5: Cybersecurity challenges, solutions, and path forward.

Comments on Challenge 5: Following is a list of the Government's proactive Cybersecurity initiatives:

- Government will formally recognize Cyberspace as a new operational domain—like land, air, sea and space. Treating Cyberspace as a domain means that the military needs to operate and defend the United States networks, which is why we have established the U.S. Cyber Command, housed at Ft. Meade.
- Government will formally equip our networks with active defenses. We have developed and now employed a more proactive and effective approach to cyber defense that operates at network speed, using sensors, software, and signatures derived from intelligence to detect and stop malicious codes before they succeed.
- Government will formally ensure that the critical infrastructures are protected on which our Federal components, Industry, and private citizens can securely depend.
- Government will formally build collective Cybersecurity defenses with our allies linked to their cyber-based defenses to provide information sharing and prevent intrusion attacks.
- Government will coordinate our nation's vast technological and human resources to ensure that we retain our preeminent capabilities in Cyberspace, as it does in other high-technology domains.

The President's Cyberspace Policy Review identifies 10 near term actions to support our Cybersecurity strategy:

1. Appoint a Cybersecurity policy official responsible for coordinating the Nation's Cybersecurity policies and activities.
2. Prepare for the President's approval an updated national strategy to secure the information and communications infrastructure.
3. Designate Cybersecurity as one of the President's key management priorities and establish performance metrics
4. Designate a privacy and civil liberties official to the NSC Cybersecurity directorate.
5. Conduct interagency-cleared legal analyses of priority Cybersecurity-related issues.
6. Initiate a national awareness and education campaign to promote Cybersecurity.
7. Develop an international Cybersecurity policy framework and strengthen our international partnerships.
8. Prepare a Cybersecurity incident response plan and initiate a dialog to enhance public-private partnerships.
9. Develop a framework for research and development strategies that focus on game-changing technologies that have the potential to enhance the security, reliability, resilience, and trustworthiness of digital infrastructure.
10. Build a Cybersecurity-based identity management vision and strategy, leveraging privacy-enhancing technologies for the Nation.

Solutions to Challenge 5: To satisfy the challenges identified above, Government will provide the following Cybersecurity solutions safeguarding its components, Industry, and private citizens' information infrastructures:

- Government will formally enforce and define its Cybersecurity policies: Prioritize risks and define requirements that govern our Industry, and private citizens. It will also enforce these policies through built-in high-technology automation and workflow.
- Government will support Industry Cybersecurity needs by proactively taking an information-centric approach to safeguard knowledge-based accessibility and accuracy. It will also initiate a content-aware approach to protecting information that is the primary solution to defining, identifying, and classifying confidential and sensitive information. This information-centric technology will identify where sensitive information resides, tracks individuals accessing it, and monitors its flow through various Internet-based systems. In addition, it identifies and records who has accessed it and how it was accessed and departed through various hardware, software, and interface vehicles. Furthermore, the information-centric system effectively encrypts various endpoints to help Cybersecurity officials minimize the consequences associated with lost devices and intrusions.
- Government will help Cybersecurity access control to identify, validate, and protect the legitimate users and prevent intruders from damaging nation's systems.
- Government will create a Cybersecurity environment and implement security procedures that distribute and enforce patch levels, automate processes to streamline efficiency, and monitor and report network systems' status.
- Government will protect its infrastructures as well as Industry and private citizen network systems by safeguarding United States systems' endpoints that include all of the Internet-based communications networks. Furthermore, it will initiate legislation, policies, and procedures that effectively protect critical internal servers, routers, and other key systems' components. It will also support the capability to store and access sensitive information and the ability to back up and recover classified and unclassified information in cases of data pollutions, damages, and intrusions.

ABOUT THE AUTHOR

Currently Dr. Ghahramani is a member of the Department of Homeland Security (DHS) National Cybersecurity and Communications Integration Center (NCCIC). Before joining the DHS, he was the Chief Technology Officer (CTO) of Eurpa Telecommunications and General Dynamics Lead Scientist before that.

Dr. Ghahramani was the Founder and Chairman of the Board of MewsTech, Inc. and BG Technical Management Solutions, Inc. Prior to starting his two successful companies, he was a distinguished member of Technical Staff (DMTS) at the AT&T Bell Laboratories and generated more than $1.9B in funded projects with various countries, industries, and governments. His extensive work also includes projects with Sandia Laboratories, Livermore National Laboratories, and Oakridge National Laboratories, incorporating sensor and telecommunications technologies. His work experience covers several years in government, academia, industry, and consulting. Dr. Ghahramani has presented and published numerous referred papers; and has been an active participant and officer in several national and international organizations and honor societies. He holds eleven patents in addition to two international patents. Dr. Ghahramani received his Ph.D. in Industrial Engineering from Louisiana Tech University; MBA in Information Systems from Louisiana State University; MS in Applied Mathematics and Computer Science from Southern University; MS in Industrial Engineering from Texas Tech University; and BS in Industrial Engineering and Management from Oklahoma State University.

Dr. Bahador Ghahramani, P.E., CISM, CPE
Department of Homeland Security
Headquarters
US-CERT Communications
Office: 703-235-3056
Mobile: 571-340-1841
Fax: 703.235.5963
E-mail: bahador.ghahramani@hq.dhs.gov
E-mail: bahador.ghahramani@us-cert.gov
Website: <http://www.us-cert.gov/>

Static Analysis Tools Pass the Quals

Yannick Moy, AdaCore

Abstract. Submitting a system to certification involves demonstrating, with a degree of confidence commensurate with the system's criticality, that it meets its requirements completely and correctly. The software life cycle process known as verification is responsible for achieving the relevant level of assurance and traditionally has relied on testing and manual reviews. Static analysis (SA) tools are starting to automate some of these verification activities. In this article we discuss what qualifies SA to be used as part of the software verification process in a certification context.

Certification and Qualification

Certification is the official process by which critical software is deemed fit-for-purpose. Depending on the domain, certification may be granted by an independent authority (civil avionics, railway, nuclear, space), the customer (military) or by an internal service within the company itself (automotive). Reference documents for certification may be international standards (civil avionics, railway, nuclear, space, automotive) or national standards (military).

In this article, we refer to the following documents:

- DO-178C (2011): civil avionics standard, known as ED-12C in Europe
- DO-333 (2011): formal methods supplement to DO-178C, known as ED-216 in Europe
- EN 50128 (2011): railway standard in Europe
- SWEHB (2013): NASA Software Engineering Handbook, guidance for implementing the NASA Software Engineering Requirements
- MISRA C (2012): coding standard for C, originally from automotive industry
- JPL C (2009): coding standard for C from NASA Jet Propulsion Laboratory

Qualification is the formal process by which software tools, for example static analysis (SA) tools, are deemed fit-for-purpose to address certification objectives. Tool vendors may provide a qualification kit that can be used by the applicant to qualify the tool in a specific certified project.

Soundness and Completeness

Soundness and completeness are usually used with different meanings in static analysis (SA) tools targeting bug finding and those targeting verification. In this article, we adopt the following definitions:

- A SA tool is sound if it never reports that a property is true when it may not be true. Reporting that an error might occur has no effect on soundness.
- A SA tool is complete if it reports all errors, either with certainty (the error occurs) or uncertainty (the error may occur).

Complete tools typically report uncertain errors where no actual error occurs. These are called false alarms or false positives. Incomplete tools typically fail to report actual errors. These omissions are called false negatives.

Static Analysis in Certification

There is no shortage of evidence that use of SA tools can increase code quality, to the point that a US military evaluator recently said that he would stop any project that did not use at least two of them. The phrase "at least two" points to repeated findings by the Static Analysis Tool Exposition (SATE) project from NIST, that different SA tools find different errors [1]. A corollary to that finding is that SA tools might not report errors in the categories that they handle. A striking example is the Heartbleed vulnerability in OpenSSL, a case of buffer overflow, which could not have been found by any SA tool before it was revealed publicly on 7th of April 2014 [2].

Most publications on SA, including previous publications in CrossTalk [3,4,5], focus on the use of SA for code quality, where the criterion of reporting only the most probable and understandable errors is justified. This is in contrast with the use of SA tools for addressing certification objectives, where the lack of soundness and completeness is inappropriate. While this use of SA is not new [6], we have seen an increasing industrial adoption in the last few years, as a cost-effective alternative to reviews or testing. In this article, we discuss what processes are used in practice to adopt SA tools in certification, based on our experience in building qualifiable SA tools at AdaCore: the coding standard checker GNATcheck, the static analyzer CodePeer, and the formal verification SPARK toolset.

As the process varies greatly depending on the category of tool, we consider separately the three categories corresponding to these tools: coding standard checking, detection of programming errors, compliance of implementation with design. These three categories are representative of the types of SA tool capabilities that are used in certification. Before considering these categories, we discuss some general aspects of SA.

Static Analysis of Source Code

Certification is concerned with getting high levels of confidence that the software behaves as it should and doesn't have undesirable behavior. SA tools used to automate verification should be sound, as expressed in DO-333: "The soundness of each formal analysis method should be justified. A sound method never asserts that a property is true when it may not be true." For SA to replace, not just complement, other activities, it should also be complete, as expressed in SWEHB: "For critical code, it is essential to use sound and complete static analyzers. Sound and complete analyzers guarantee that all errors are flagged and that no false negatives (i.e., an erroneous operation being classified as safe) are generated." This implies restrictions on programs or properties, which we discuss later, since it is impossible for any tool to be sound and complete for all conceivable properties and totally unrestricted programs.

It is important to realize that SA tools that analyze source code only analyze a model of the real executable program. Unless a programming language was primarily designed for SA (like B Method [7] or SPARK [8]), it almost certainly contains ambiguous constructs, which may be interpreted differently by a SA tool and by a compiler. In imperative programming languages such as Ada, C, C++ or Java, there are three classes of ambiguous constructs:

- Implementation-defined behavior. It is specified, possibly differently, by each compiler, for example the size of standard scalar types.
- Unspecified behavior. The set of possible effects of a construct is defined by the language but is not necessarily specified by the implementation's documentation, for example the order of evaluation in expressions.
- Undefined behavior. The effect of program execution is unpredictable, for example reading an uninitialized variable.

These can be addressed at three levels: the language, the execution platform and the compiler.

Addressing the Language Issue

For many cases of undefined behavior, the default language rules do not allow automatically separating correct from erroneous programs. For example, in C it is undefined to assign twice to the same variable inside the same sub-expression: "Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression." In the presence of calls and aliasing, no available SA tool can currently perform this check on unrestricted C programs. Instead, they can check more restrictive rules that prevent this type of undefined behavior, for example rule 19 of JPL C, which states "Do not use expressions with side effects", or rules 13.2, 13.3 and 13.4 of MISRA C, which restrict assignment operations.

Although it is conceivable to mirror all possible compiler-specified behavior in an SA tool, it is impractical because of multiple compilers with many version and switch variants. Instead, language rules (or coding standards) are defined that prevent this type of unspecified behavior. For example, the order of evaluation of parameters in a function call are not specified in Ada, C or C++, which can lead to different observable behaviors if expressions have side effects. By completely prohibiting such side effects, the rules defining the SPARK subset of Ada or rule 19 of JPL C ensure that all orders of evaluation will be equivalent.

Addressing the Execution Platform Issue

Many cases of implementation-defined behavior depend on the execution platform, for example the size of standard scalar types or the behavior of calls to run-time functions. This issue can be partially resolved through parameters to the SA tool.

Addressing the Compiler Issue

In those cases of unspecified and implementation-defined behaviors that are neither prevented by language rules nor resolved by the choice of execution platform, the SA tool needs to be able to handle all possible compiler choices. This is easier to achieve if the SA tool shares code with the compiler used to generate the executable program. This is for example the choice that we have made at AdaCore, when reengineering the CodePeer and SPARK SA tools by basing them on the GNAT Ada frontend for the GCC compiler. [9]

Coding standard checkers are probably the most widely used SA tools in certification. They automate the verification of language restrictions such as those mentioned earlier.

Coding Standard Checking

All certification standards require the definition of a coding standard, defining a set of rules that specify which language features are allowed (or prohibited). Typically, a coding standard contains rules that forbid some language features, and rules that limit code size/coupling/depth. Well-known coding standards are MISRA C [10] in automotive or the JPL C [11] at NASA.

The first objective of a coding standard is to prevent the introduction of certain types of programming errors. As SWEHB says: "The use of uniform software coding methods, standards and/or criteria ensures uniform coding practices, reduces errors through safe language subsets, and improves code readability."

The second objective of a coding standard is to facilitate detection of programming errors by testing and SA. For example, EN 50128 defines an objective "to identify programming constructs which are either error-prone or difficult to analyze, for example, using static analysis methods", with the aim that "a language subset is then defined which excludes these constructs." Similarly in DO-178C, an objective "is to ensure the Source Code does not contain statements and structures that cannot be verified". Indeed, SA tools are not sound on the entire language they target, for example when expressions have side effects or when arbitrary pointer conversions are allowed. Thus, a subset of the language should be defined in a coding standard, such that sound SA is possible. For Ada, an ISO technical report defines the compatibility between language features in Ada and classes of SA. [12] SPARK is an example of such a subset of Ada, which allows sound data-flow, information-flow, robustness and functional analyses.

Rules may be verified automatically, manually, or by a combination of both. Experience shows that manual verification, in particular when outsourced and performed only once on the final source code to decrease costs, is both error-prone and time consuming. Thus, most rules are usually verifiable automatically. For example, MISRA C distinguishes between decidable guidelines (116), where a guideline is decidable "if it possible for a program to answer with a yes or a no in every case", and undecidable ones (43). As another example, JPL C (120 rules) was applied to develop the code of the Curiosity rover, which landed on Mars in 2012, and "compliance with all risk-based rules could therefore be verified automatically with multiple independent tools on every build". [13] Only automatic tools can deal with the large number of rules in these coding standards.

As the previous quote indicates, automation also makes it possible to check the coding standard as part of the development process, catching violations as soon as they occur. Even better, most rules only require analysis of a relatively few surrounding lines of code and can therefore be analyzed very quickly on a developer's machine. There is much to gain in adapting the process to facilitate frequent analysis:

- All rule violations should be justified so that running the SA tool returns without finding any unjustified violations.
- The SA tool should be integrated in the developer's IDEs, so that it is as easy to run as it is to compile the code.
- The SA tool should be integrated in the version control system (using features for pre-commit checks) or build system. This ensures that one developer's error gets detected before it impacts others.

With the correct set of automatable rules, SA tools, and developer-centric processes, coding standard checking can be applied continuously. As SWEHB puts it: "Assuring the adherence of the developed software to the coding standards provides the greatest benefit when followed from software development inception to completion." Maximizing automation makes it possible to invest user efforts in the richer types of SA that follow.

Detection of Programming Errors

While coding standard checking is mostly concerned with decidable rules, detection of programming errors is mostly concerned with undecidable properties. Thus, a complete SA will necessarily report false alarms (possibly many), which need to be addressed in the process.

The usual main categories of programming errors detected by SA are run-time errors (buffer overflow, integer overflow, read of uninitialized data, null pointer dereference, etc.), dead code, concurrency errors (race condition, deadlock), and operating system and SQL injection vulnerabilities.

Run-time errors correspond to improper behavior according to programming language rules: a predefined exception is raised, or the behavior becomes undefined. Because there is a precise definition of where these errors occur, a complete SA is possible. But as there are many such possible errors in a program, there may be also many false alarms.

There are variations among domains on exactly which kinds of dead code, which may also be called extraneous code or unreachable code, are relevant. The two main kinds are code "which exists as a result of a software development error but cannot be executed" (DO-178C) or code "that is executed but whose removal would not affect program behavior" (MISRA C). Both are equally impossible to detect completely by SA. Instead, SA tools only report unconditionally unreachable code, with no false alarms.

If the detection of errors is incomplete, as with dead code detection, SA cannot replace testing or review activities, but can only complement them. If the detection of errors is sound and complete, which is possible for run-time error detection, SA can lower the effort on testing or review activities. Whether this can be done in practice for a given project, with a given tool, depends on the confidence that the team has in the tool. When the tool has been successfully used in practice over a long period of time, service history may provide enough confidence. Otherwise, tool qualification should be performed. Firstly, the SA tool should be used on a subset of the language on which it is sound and complete. Secondly, all unsound or incomplete heuristics meant to minimize the number of false positives in bug-finding mode should be deactivated. In our work on SA tools at AdaCore, both the language subset and the deactivating switches form part of the qualification kit that we ship to our customers. They define the qualified interface ensuring sound and complete analysis. Both subsets also ensure that tool qualification is feasible by limiting the cost of developing the tests that are needed for qualification.

The process varies depending on the time needed to run the tool, and on the precision of results:

1. The ideal situation is that developers can run the tool frequently, with few false alarms. This is typical of modular analyses. Developers are expected to commit code that passes the SA without errors, with all false alarms justified.
2. In a more common setting, the tool takes too long to be run by developers, but there are still few false alarms. Instead, developers have access to the results of a nightly run. This is typical of the global analysis performed in abstract interpretation-based SA, paired with a restricted language subset. Developers or some designated team members are expected to report/fix errors and justify false alarms on a daily basis.
3. In probably the most common setting, the run is long and produces too many false alarms to be integrated in the daily development process. This is typical of global analysis of programs with pointers, dynamic allocation, indirect calls, etc. Manual review of the analysis results is performed only at planned milestones, either by the development team or a dedicated Quality Assurance team.

While detection of programming errors mostly requires post-hoc manual work for dealing with false alarms, richer types of SA such as compliance of implementation with design require more preliminary work.

Compliance of Implementation with Design

Just as important as the absence of programming errors, the source code should implement the software design. In DO-178C, source code should be shown to be compliant with the two parts of design that are software architecture and low-level requirements (a.k.a. function specifications). Some specification languages make it possible to describe architectural constraints and low-level requirements, which allows developers to demonstrate by SA that the implementation satisfies those constraints and requirements.

This is illustrated in the languages of annotations for C programs used at Airbus in conjunction with SA tools Caveat and Frama-C [14], as described in DO-333 under the name of unit proof. First, low-level requirements are expressed as function contracts, consisting in a precondition (on entry to the function, established by the caller) and a postcondition (on exit from the function, established by the function itself). Then, code is developed to implement those requirements. Finally, SA is used to prove that the implementation satisfies the requirements. The SPARK subset of Ada similarly allows checking low-level requirements expressed as subprogram contracts, as well as architectural constraints expressed as data flows or constraints on calls. These types of SA were used on the UK military programs SHOLIS and C130J Hercules (submitted to DEFSTAN 0055 certification). [8] The B Method used in railway also allows proving high-level software requirements, at the cost of more manual proof effort. [7]

The process to apply SA depends here on the complexity of the properties to prove. Simpler properties like absence of run-time errors (see previous section) are mostly proved automatically, with the occasional addition of simple preconditions and postconditions. In our experience with SPARK, more complex properties require some tuning:

1. If contracts are executable (as in SPARK 2014 or E-ACSL contracts in Frama-C), run tests to detect most cases of mismatch between contracts and code.

2. Interact with automatic provers through the IDE to correct code or contracts and ensure the property is proved automatically.

3. If a property cannot be not proved automatically, either default to testing if the integration of proofs and tests is possible, or turn to manual proof in a theorem prover, or add a manual justification.

The key feature of these SA tools is that they analyze each function in turn, using user-written function contracts to analyze calls. This type of highly modular SA can be run easily in parallel or on a distributed farm of servers, making it possible to integrate in the daily development process.

Conclusion

We have described the processes that we have seen applied in practice to use three categories of SA tools in certification. Coding standard checking is a "must-have" of every project in certification. Detection of programming errors is a best practice, recognized for both reducing costs and improving quality. Compliance of implementation with design has been embraced by pioneers like Airbus [14, 15]. In all three categories, use of sound and complete SA tools guarantees that no error has been missed, which is essential when looking for the last error. Of course, this depends on there being a sound and complete SA tool for the property of interest and the programming language subset used.

Acknowledgements

This article benefited greatly from comments of colleagues at AdaCore, as well as reviews from Paul Black, David Mentré and Franck Sadmi, which I'd like to thank here.

ABOUT THE AUTHOR

Yannick Moy is a Senior Software Engineer at AdaCore, where he works on software source code analyzers, mostly to detect bugs or verify safety/security properties. Yannick previously worked on source analyzers for PolySpace Technologies, France Telecom R&D and Microsoft Research. Yannick holds an engineering degree from the Ecole Polytechnique, an MSc from Stanford University and a PhD from Université Paris-Sud. He is a Siebel Scholar.

AdaCore
46 Rue d'Amsterdam
Paris, France 75009
Phone: +33.1.4970.6718
Fax: +33.1.4970.0552
E-mail: moy@adacore.com

REFERENCES

1. Paul E. Black, "Static Analyzers: Seat Belts for Your Code", IEEE Security & Privacy, 2012
2. David A. Wheeler, "How to Prevent the next Heartbleed", 2014. <http://www.dwheeler.com/essays/heartbleed.html>
3. Paul E. Black, "Static Analyzers in Software Engineering", CrossTalk, March-April 2009
4. Yannick Moy, "Static Analysis Is Not Just for Finding Bugs", CrossTalk, Sep-Oct 2010
5. Piyush Jain, Ramakrishna Rao and Sathyanand Balan, "Challenges in Deploying Static Analysis Tools", CrossTalk, July-Aug 2011
6. Andy German, "Software Static Code Analysis Lessons Learned", CrossTalk, Nov 2003
7. Patrick Behm, Paul Benoit, Alain Faivre, and Jean-Marc Meynadier, "Météor: A successful application of B in a large project", FM'99
8. Ian O'Neill, "SPARK - A Language and Tool-Set for High-Integrity Software Development", chapter in "Industrial Use of Formal Methods: Formal Verification", Wiley, 2012
9. Johannes Kanig, Edmond Schonberg and Claire Dross, "Hi-Lite: the Convergence of Compiler Technology and Program Verification", HILT'12
10. MISRA, "MISRA C:2012 Guidelines for the use of the C language in critical systems". <http://www.misra-c.com/>
11. NASA Jet Propulsion Laboratory, "JPL Coding Standard for Flight Software". <http://lars-lab.jpl.nasa.gov/JPL_Coding_Standard_C.pdf>
12. ISO/IEC, "Guide for the Use of the Ada Programming Language in High Integrity Systems", 1999 <http://www.open-std.org/jtc1/sc22/wg9/n359.pdf>
13. Gerard J. Holzmann, "Mars Code", Communications of the ACM, Feb 2014
14. Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels and Benjamin Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience", IEEE Software, 2013
15. Claire Dross, Pavlos Efstathopoulos, David Lesens, David Mentré and Yannick Moy, "Rail, Space, Security: Three Case Studies for SPARK 2014", ERTS2, 2014

Locking Down the Software Development Environment

Madeline Wright, MCTD
Dr. Carl Mueller, Texas A&M

Abstract. The goal of this paper is that configuration management is a simple and cost effective method to secure the development environment without impeding innovation, creativity, or schedule. Software development is a business, and it is reasonable to assume both developers and customers want systems that are protected because there will always be attempts to gain access to software and the data/information residing in the computer systems.

Introduction

Software security is to prevent the world from harming the system, malicious or unintentional, [1] and encompasses the ability to prevent, detect, and react to malicious indicators. Many security analysts view external threat-agents as the primary source of harm. However, the greatest risk to any system is from those who are developing the system. Developers introduce security exposures either maliciously or unintentionally into the system. Mitigating these risks requires security analysis and an investigation of all vulnerabilities in the Software Development Life Cycle (SDLC) models. Investigating the vulnerabilities for all of the SDLC models is extremely labor intensive, but there are three areas common to most models: Programming personnel, Configuration Management (CM) practices, and Quality Assurance (QA) practices. Each area of the SLDC reviewed emphasizes the difference between secure coding and securing the environment by reviewing threat agents and threat exposure, assessing configuration management's role in securing the environment, and finally, discussing methods to monitor for malicious intent with lessons from Stuxnet and the Heartworm Bug. The authors look at the SDLC from a lens that strives to guide software teams to implement a comprehensive security policy to make penetration of the system's information security perimeter more difficult [2].

Programmers as Threat Agents

Microsoft reports more than 50% of the reported security defects are introduced in the design of a component [3] and this is critical to looking at where the threats exist and opportunities for threat exposure in the in the software development environment. Microsoft's finding suggests both designers and programmers are threat agents in the development environment. According to Microsoft's data, designers and programmers introduce vulnerabilities into an application; it is therefore appropriate to identify all of the software developer roles (analysts, designers, programmers, testers) as potential threat-agents. Viewing software developers as threat-agents should not imply the individuals filling these roles are careless or criminal, but they have the greatest opportunity to introduce source code compromising the systems confidentiality, integrity, or availability. Software developers can expose assets accidently, by introducing a defect or intentionally through the introduction of malicious functionality. Defects have many causes, such as oversight or lack of experience with a programming language, and are a normal part of the development process.

Development processes are the activities, constraints, resources, and techniques to produce an intended output [4]. Software developers perform the daily development activities habitually; they know their jobs, so why question whether there are flaws in the development process? The answer is that team members have their own view of the process, based on what is important to them personally. For example, someone's financial circumstances may have changed or other personal hardships could result in an angry, vulnerable, or distracted team member. Many organizations conduct background checks, credit checks, and drug tests when hiring new employees as part of the company's security policy, but the company may not perform periodic background checks. Developers need to understand that as organizations uses locks on their doors to protect their physical property; there is a need to conduct periodic security screenings to protect intellectual property and financial assets from those with the greatest access. Although these actions are intrusive, they serve to sustain a secure environment, provide stability through structure, and reduce risk.

During development of a software application, there are many opportunities to introduce security exposures. To address these exposures, many researchers recommend enhancing an organization's QA program. One frequent recommendation is expanding the inspection practice by introducing a checklist for the various exposures provided by the programming languages [3] [5]. Items added to a security inspection checklist typically include functions such as Basic's Peek () and Poke () functions, C's string copy functions, exception handling routines, and programs executing at a privileged level. Functions like Peek() and Poke() make it easier for programmers to access memory outside of the program, but a character array or table without bounds checking produces similar results. A limitation of the language specific checklist is each language used to develop the application must have a checklist. For some web applications, this could require three or more inspection checklists, and this may not provide safeguards for all the vulnerabilities. Static analyzers, such as the Software Assurance Metrics And Tool Evaluation (SAMATE) research, sponsored by the National Institute of Standards and Technology (NIST), is an approach to automating some of the objectives associated with an inspection checklist, but static analyzers have a reputation for flagging source statements that are not actually problems [6]. Using a rigorous inspection process as a safeguard identifies many defects, but does not adequately protect from exposures due to malicious functionality. An inspection occurring before the source code is placed under configuration control provides substantial exposure. In this situation, the developer simply adds the malicious functionality after the source code passes the inspection or provides the inspectors a listing without the malicious functionality.

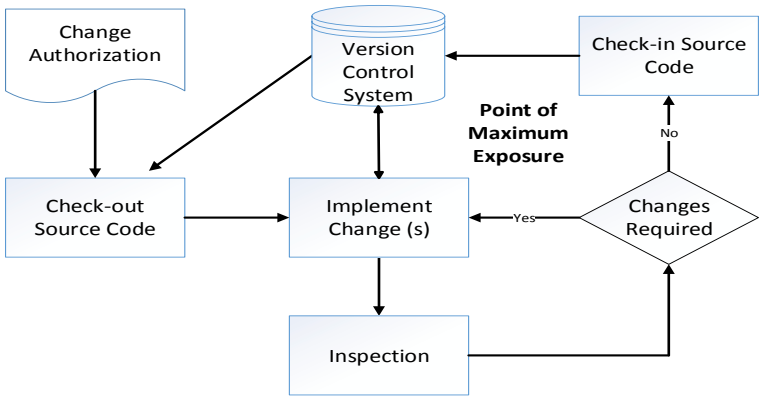


Figure 1. Traditional Unit Level Development Process

Programming languages and development processes offer a number of opportunities to expose assets; but many of the tools, such as debuggers and integrated development environments, can expose an asset to unauthorized access. Many development tools operate at the same protection level as the operating system kernel and function well as a worm to deposit a root kit or other malicious software. Another potential exposure, not related to programming languages, is testing with 'production data'. Using 'production data' may permit access to information the developers do not need to know. Now that we have addressed some areas to apply QA, we will look at the next step, addressing vulnerabilities and back doors.

Software developers address software vulnerabilities by loading the latest virus definitions and implementing all of the security guidelines [7], but if the environment is not secure, the product is still at risk for back doors or malicious intent resulting in security breaches. Acknowledging the software development environment's threat agents are often the software engineers is a major step in reviewing security processes. There is a tendency to focus on the existing vulnerabilities and threats since this provides managers and customers a sense that all security bases are covered. The development environment permits many opportunities to discover a business's processes and, in turn, their vulnerabilities, and how we protect those process leads to our next topic, Configuration Management.

Configuration Management

One tool for securing the environment is auditing through CM so the development team can focus on building functionality, not managing the change [8]. Software CM (SCM) is the traditional technique for controlling the content of deliverable components and is an essential element of a robust security policy [9]. Figure 1 illustrates a traditional unit-level development process indicating possible points of vulnerability. As illustrated in Figure 1, a developer receives a change authorization to begin the modification or implementation of a software unit. Generally, the "authorization" is verbal and the only record of the authorization appears on a developer's progress report or the supervisor's project plan. To assure another developer does not update the same source component, the developer "reserves" the necessary source modules. Next, the developer modifies the source code to have the necessary features. When all of the changes are complete, the developer informs the supervisor who assembles a review panel consisting of three to five senior developers and/or designers. The panel examines the source code to evaluate the logic and documentation in the source code. A review committee recommends the developer make major changes to the source code that requires another review, minor changes that do not require a full review, or no changes are required. It is at this point in the development process where the source code is the most vulnerable to the introduction of malicious functionality, because there are no reviews or checks before the software is "checked-in."

Another limitation of inspections is that the Agile methodology recommends formal inspections and Scrum uses pair programming and testing based on the Backlog list to determine what functionality is priority for developer resources [5]. Using inspections as the primary safeguard from development exposures limits the cost savings promised by Agile development methodologies and does not provide complete protection from a developer wishing to introduce malicious software. Of the six areas of CM, the two areas having the greatest effect on security are configuration control and configuration audits. Version control tools, such as Clearcase and CVS, provide many of the features required by configuration control. Most version-control systems permit anyone with authorized access to check source code "in" and "out" without an authorized change request and some do not even track the last access to a source module. However, in a secure environment, a version control system must integrate with the defect tracking system and record the identification of the developers who accessed a specific source module. Integrating the version control system with the defect tracking system permits only the assigned developer to make a specified change and access the related source code. It is also important for the version control system to track the developers who access the code – traceability. Frequently, developers copy source code from a tested component or investigate the approach used by another developer to address a specific issue in their work, or need access to read source modules they are not maintaining. This access provides an opportunity for insiders to research and learn how to introduce malicious functionality into another source module. By logging source module access, security personnel can monitor access to the source code. Configuration audits are the second management technique making a development organization more secure [10]. Some regulatory agencies require audits for safety critical applications/high reliability applications to provide an independent review of the delivered product. An audit in a high security environment addresses the need to assure the delivered product's software does not expose the organizational assets to risk from either defects or malicious functionality. To increase confidence that the delivered software does not contain defects or malicious functionality, auditors should assure that the test cases provided 100% coverage of the delivered source code. This is particularly important with interpreted programming languages, such as Python or other scripting languages, because a defect can permit the entry of malicious code by a remote use of the software. Auditors could adopt the approach of selectively re-testing configuration items with the unit-test data to assure the results from the re-test match those produced in the verification and validation procedure, and that all of the statements in the code are executed.

Adopting the recommendation for a stronger CM process modifies the typical unit-level development process, illustrated in Figure 1, to a more secure process illustrated in Figure 2. In the more secure process illustrated in Figure 2, a formal change authorization is generated by a defect tracking system or by the version control system's secure change authorization function. Next, a specified developer makes the changes required by the change authorization. After implementing and testing the changes, the developer checks all of the artifacts (source code, test drivers, and results) into the version control system. Checking the artifacts automatically triggers a configuration audit of the developer artifacts. Auditors may accept the developer's changes or create a new work order for additional changes. Unlike the review panel, the auditors may re-test the software to assure adequate coverage and that the test results match those checked in with the source code. Making this change to the development process significantly reduces the exposure to accidental defects or malicious functionality because it verifies the source code deployed in the final product, along with all supporting documentation.

CM should be viewed as important to keeping the development team informed of code changes, to include who made the code changes, and to indicate when those code changes were made [11]. The goal is to inform and challenge developers to merge security and process to prevent the introduction of malware by threat agents through safeguards to strengthen existing processes. CM helps keep the cost of security low-cost with the basics of version control and audit. Management will be pushing the team to deliver and move on to the next project, software developers need something practical that provides structure while not restricting innovation and creativity or impeding the schedule. CM should play a major role in securing the software development environment because it assists with the discovery and prevention of malicious intent by threat agents, whether the agents are hackers, malware coders, or insider threats. Next, we will discuss how to establish QA in the SLDC and use lessons from external threats to emphasize the importance of QA.

Quality Assurance and Lessons from External Threats

Another element of a robust development security policy is QA in the SLDC through the separation of the development and production systems. Developing software in the production environment exposes organizational assets to a number of threats, such as debugging tools or simply writing a program to gain unauthorized access to information stored on the system. A worm implanted on computers or portable flash drives that might eventually be connected to the targeted network, such as the Stuxnet worm. The level of sophistication of the Stuxnet worm could only come from insider knowledge of the computing architecture and daily operations. Another important point about the Stuxnet worm is that it targeted a development tool and the tool introduced the malicious functionality. For a secure development environment, testing must not only look at the vulnerabilities of the past, but also conduct what-if analysis for all of the tools and software being used for the project developers have to think like a hacker because hackers work to reverse engineer an application to make it perform what they want. Since Stuxnet was engineered brilliantly as described by

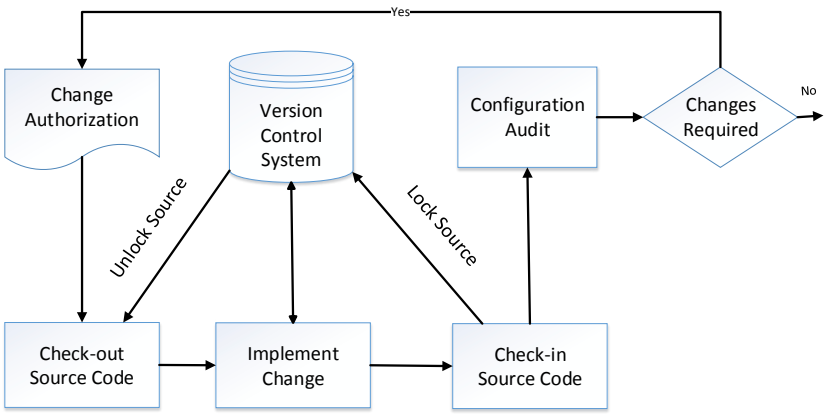


Figure 2. Secure Unit Development Process

personnel from Kaspersky Lab [12], developers must take preventive action. Code can have a time delay or malware can remain dormant, or malware may be hidden in a library or service routine, all actions an insider could insert for activation after the software is deployed. The Heartbleed Bug was a software defect developers closed in April 2014 with a patch to the OpenSSL code [13]. The OpenSSL software is, as the name implies, open source, a result of many developers coding beginning in 1998 using the C programming language to build crypto services. OpenSSL is used widely both on the Internet and in firmware [13], further delaying the ability of many organizations to implement the patch across all platforms for the HeartBleed bug. There will be damage from the HeartBleed bug because of the types of data possibly extrapolated from websites, i.e. certificates, user names and passwords, messages, emails, and documents. Using open source code does not preclude the development team from checking that the code meets QA standards. Code must perform its intended function and not introduce security vulnerabilities. Identifying all of the potential exposures and creating safeguards provides a significant challenge to the security analysts; but by analyzing the development process, it is possible to identify a number of cost effective safeguards. Configuration management, pairwise coding, and vulnerability checks are all simple security safeguards that could have prevented the HeartBleed bug. Security breaches have made the news headlines and consumed labor hours for patches and corrective actions, a result of attacks on software services by software developers who have become insider threat agents. The insider threat is real, whether it is from former employees who have maintained possession of their identification cards or contracted National Security Agency (NSA) employees having access levels exceeding their authorization, the insider can upend all of a development's team effort. Social engineering in the case of Stuxnet [14] and other worms prompted the government to ban USB drives, and these examples continue to prove that securing the environment is the only option. Give hackers a face; they are thieves with an objective to gain access into a software system for which they do not possess the credentials to access or possess. Locking down the environment means the owners of the SDLC are a step ahead of the criminals because they employed QA for each step in the process. For example, developers have to begin to

design tests beyond code issues and look at routine service calls to ensure the code does not introduce a common vulnerability for exploitation upon deployment. Consider whether Bluetooth is enabling cyber espionage. Historical data from the credit card industry has proven Bluetooth as a lucrative technology for cyber thieves and should be monitored to ensure the interfaces do not contain malware. Programs should be designed to perform cyber surveillance on the malware so programs get routine reports on data sent to servers and service routines – trends and analysis can point to unusual activity. Agile takes advantage of the cross-functional teams with both developers and testers to improve the quality of the software [15]. This teaming could be used in any type of development model to encourage collaboration and feedback. The feedback loop and reinforcement of using the QA and CM processes, manual or automated, traces changes and makes changes visible to everyone on the team [8]. Visibility and sharing of information in the Agile Process make it difficult for insiders to bypass the CM process and could even prevent someone from attempting espionage. For example, if a developer wanted to add a logic bomb and deploy it six months later, he or she might reconsider if they knew the code would be peer reviewed [16] [17]. If CM personnel compare lines of code from the previous version to a new version, the records of code submissions are traceable to an individual, and this step could curtail fraudulent activity. Waterfall and its recordkeeping rigor remove the anonymity, but testing at the end still provides the insider opportunities to hide malicious code during the rush to meet the delivery schedule. So how does the development team know the code deployed is the code the team wants to deploy? Adherence to code changes being checked in to CM, and verification, i.e. testing, comparing the number of lines of code to the tested lines of code is critical to documenting the software baseline. Teams must follow the process and not allow a programmer to check in code without an audit trail. Processes remove the human-in-loop factor, i.e. relationships between the developer and the CM Lead, ensuring processes apply to everyone. The process requires buy-in at all levels because security is the responsibility of the development team. Accountability in the software development environment maintains the integrity of the software. If developers follow the CM process, then there is traceability. Development teams must not forfeit traceability and take shortcuts – the process is there to help protect the product. Not following the process opens the software development environment to vulnerabilities. Statistics from the Computer Emergency Response Team/Coordination Center (CERT/CC) indicates well-known vulnerabilities are responsible for 75% of the security breaches; leaving 25% a secure development model will not address [12]. It sounds simple, map to known vulnerabilities to improve your design and let this become a new model to assist the testers, QA teams, and automated tools. Shirazi's proposal lists 25 common vulnerabilities, such as buffer overflow, integer overflow, command injection, SQL injection, cross-site scripting, and illegal pointer values, along with countermeasures to avoid the vulnerabilities [18]. Many developers could view this model as too prescriptive, because it assumes if most programmers followed basic secure programming principles, then the environment would be locked down. Some would also view this model as useful to train inexperienced developers or to standardize knowledge across the organization. Any list can become a useful checklist, but a model is a process that can be repeated to impose consistency on the output. Consider how software development teams moved from Waterfall to Agile; the process selected by the team depends on the environment and business need. Additionally, change has to be significant to encourage and sustain change [14]. Secure software development does not require a new model, but the phases of the existing models must address security and the system architecture in the test and QA phases. Another proposed secure software development model is to build test tools based on ratios of predicted vulnerabilities [15]. Any model relying on reported vulnerabilities or associations attempts to predict future vulnerabilities does provide indicators [19], but until the model's variables are validated, caution must be used. These proposals help to code securely, but still lack in

providing a means to locking down the software environment [14]. While past vulnerabilities do assist with inspection and testing [16], [17], this is still a reactive model. Code churn is expected during development, but sound CM practices can manage the change. Additionally, there are many secure development lifecycle models (SDL), for example, Microsoft's SDL is attempting to train new developers. This makes the development team aware of the pitfalls of functional but unsecure software [20], but this still does not secure the environment, and training does ensure the developers understand secure coding. Secure coding does not prevent malicious intent within the environment. As one technology manager stated, developers' mistakes rarely affect them directly because the defects are not discovered until the operations phase. Correcting defects once software is deployed affects the operations budget, not the development budget [21]. Some security professionals attempt to scare the developers with doom and gloom about security and privacy problems, as well as describing how best to ensure compliance to regulations through secure coding [22]. There should be an emphasis on information sharing; inform the development teams and everyone in the company about how much it costs in operations and marketing to clean up the environment if malware is not prevented [23] [24]. Steps such as training, meetings to gain buy-in, rewards to developers who embrace methods that secure the environment, as well as reprisals for those who do not adhere to doing their part to secure the environment are necessary to instill a culture change so everyone values securing the environment. Some notable examples of the threat agent's work were the use of social engineering and development tools in the case of Stuxnet and Heartbleed Bug. Both exploits resulted in havoc and further degraded the public's ability to trust the security of software. Developers must take action by viewing the software development environment as an entity within the SLDC that requires a rigorous auditing method. Development is the stage in the SLDC is when the software is most vulnerable to threat agents so there has to be a focus on gaps in the development process. The company developing software does not discharge its responsibility after its deployment; the creation, however perceived, remains a product that is company's responsibility until the software is retired.

Summary

Software development is a business, and it is reasonable to assume that customers want systems that are protected against known vulnerabilities. The insider threat is real, whether it is from former employees who have maintained possession of their identification cards or contracted National Security Agency (NSA) employees having access levels exceeding their authorization, the insider can upend all of a development's team effort. Social engineering in the case of Stuxnet [14] and other worms prompted the government to ban USB drives, and these examples continue to prove that securing the environment is the only option. Software developers can address software vulnerabilities, load the latest virus definitions, and implement all of the security guidelines [7], but if the environment is not secured, the product is still at risk for back doors and malicious intent that lead to security breaches. Security breaches result in damages to customer and developer reputation. Only a comprehensive security policy focusing on personnel, operations and configuration management can provide the safeguards necessary to secure an organization's assets from cyber risk factors [2]. Following all of these recommendations will not guarantee the security of the software development environment. There are always new vulnerabilities and vulnerabilities from social engineering. However, using reoccurring security checks, separating developers from production systems and data, controlling media, training developers, a culture of awareness, and rigorous configuration management practices should make penetration of your information security perimeter more difficult. It is also necessary to conduct a periodic review of development tools and configuration management practices, as well as a review of the security standards because threat agents will adapt to any safeguard that does not adapt to new technology. It makes good business sense to allocate resources proportionately across the development and maintenance phases to prevent malicious intent, versus stopping the current work and re-directing work to address malicious intent. Technology changes, customers' business changes, the market changes, and management changes are all drivers that will influence and determine how the customer will assess the quality of the final software product.

Although this paper does not strive to develop any new methodologies, models, processes, or test tools, each area reviewed in the SLDC emphasized the difference between secure coding and securing the environment with a focus on the effectiveness and simplicity of Configuration Management in keeping the cost of security low-cost with the basics of version control, quality assurance, and auditing. Any security policy focusing on one aspect of the process will not succeed - the software development environment is complex, encompassing people, systems, and software. Since management will be pushing the team to deliver and move on to the next project, software developers need something practical that provides structure while not restricting innovation and creativity or impeding the schedule. In that vein, the authors of this paper hope to have contributed.

ABOUT THE AUTHORS

Madeline Wright has more than 15 years of experience in software testing and evaluation of mission command systems through test planning, resource management, and collecting data to support operational tests She currently serves as senior test manager for the Mission Command Test Directorate (MCTD), US Army Operational Test Command (OTC), a test center for the US Army Test and Evaluation (ATEC) Command. Mrs. Wright gained system of system test experience at U.S. Army Communications-Electronics Command's (CECOM) Whitfill Central Technical Support Facility (CTSF), supporting the Electronic Proving Ground (EPG), to lead the interoperability testing of new software baselines for the Army's Chief Information Office/G-6. Mrs. Wright's MCTD team also has the responsibility to manage testing as system under test (SUT) headquarters for OTC during the Network Integration Evaluation (NIE) events; this includes mission command systems, network capabilities, and radios that are planned for fielding to keep the Warfighter with technology upgrades. Mrs. Wright holds a BA with a major in Computer Information Systems and an MBA from Tarleton State University – Central TX, as well as two Master Certificates from Villanova University in Six Sigma Green Belt and Lean Six Sigma (LSS). Mrs. Wright is Level III certified in Information Technology by the Defense Acquisition University (DAU) and she is a member of the Army Acquisition Corps. Mrs. Wright is working towards her DAU Test and Evaluation certification. Her publications include abstracts/presentations at the 26th and 27th NDIA Conferences. Mrs. Wright lives in Harker Heights, TX with her husband of 27 years, Charles, a retired Army NCO. The Wrights have two daughters, Charlene who serves with the Navy, and Jamilla, who lives in Dallas, TX. Mrs. Wright is pursuing a Computer Science/Software Engineering degree with TAMU-CT prior to pursuing a doctorate as part of her continuing education. Mrs. Wright enjoys family time and traveling. She supports the Killeen Food Bank with food drives and raising funds for scholarships as part of her duties as President of the Delta Mu Delta Zeta Lambda Chapter at Texas A&M University – Central Texas (TAMU-CT).

Phone: 254-286-6300 ext. 6302
E-mail: madeline.b.wright.civ@mail.mil

Carl J. Mueller, Ph.D., CISSP. Dr. Mueller is an assistant professor in the Department of Computer Information Systems, TAMU-Central Texas in Killeen. He received is initial training in computer science from Marine Corps Schools Quantico, and later earned a PhD from Illinois Institute of Technology. Dr. Mueller has over 9 years of teaching experience and more than 35 years of industrial experience specializing in developing and testing safety critical/high reliability applications (medical devices, telephony, and other applications).

Phone: 254-519-5400
E-mail: muellercj@ct.tamus.edu

REFERENCES

1. C. W. Axelrod, Engineering Safe and Secure Software Systems, Norwood, MA: Artech House, 2012, p. 349.
2. M. Mushi and J. Bakari, "Security in In-House Developed Information Systems: THe Case of Tanzania," Systemics, Cybernetics and Informatics, vol. 10, no. 2, pp. 1-5, 10 2012.
3. N. Davis, W. Humphre, S. J. G. Zibulski and G. McGraw, "Processes for Producing Secure Software: Summary of US National Cybersecurity Sumit Subgroup Report," IEEE Security and Privacy, vol. 2, pp. 18-25.
4. S. L. Pfleeger and J. M. Atlee, Software Engineering Theory and Practice, Prentice Hall, 2010.
5. M. Beedle and K. Schwaber, Agile Software Development with Scrum, Upper Saddle River, NJ: Prentice Hall, 2002, pp. 1-158.
6. "SAMATE - Software Assurance Metrics and Tool Evaluation," 2014.
7. H. Shahriar and M. Zulkernine, "Mitigating Program Security Vulnerabilities: Approaches and Challenges," ACM Computing Surveys, vol. 44, no. 3, pp. 11.1-11.46, 6 2012.
8. M. E. Moreira, Adapting Configuration Mangement for Agile Teams: Balancing Sustainability and Speed, Hoboken, NJ: Wiley, 2010, p. 303.
9. A. Leon, A guide to software configuration management, Artech House, Inc, 2000.
10. N. R. Nielsen, "Computers, security, and the audit function," in Proceedngs of the May 19-22, 1975, National Computer Conference and Exposition, Anaheim, CA, 1975.
11. H. R. Berlack, Software Configuration Management, New York: Wiley, 1992.
12. D. Kushner, "The real story of stuxnet," IEEE Spectrum, vol. 50, no. 3, pp. 48-53, 2013.
13. J. Lyne, "How Hearbleed Happened, The SNA and Proof Heartbleed Can Do Real Damage," 2014.
14. I. Porche, S. McKay and J. M. and Sollinger, A Cyberworm That Knows No Boundaries, Santa Monica: Rand Corp, 2011.
15. J. L. Cooke, Everything You Want to Know About Agile: How to Get Agile Results in a Less-than-agile Organization, Ely, Cambridgeshire: IT Governance, 2012.
16. R. L. Jones and A. Rastogi, "Secure Coding: Building Security into the Software Life Cycle," Information Systems Security, pp. 29-39, 11/12 2004.
17. V. O. Safonov, Using Aspect-Oriented Programming for Trustworthy Software Development, Hoboken, NJ: Wiley-Interscience, 2008.
18. H. M. Shirazi, "A New Model for Secure Software Development," International Journal of Intelligent Information Technology Application, vol. 2, no. 3, pp. 136-143, 2009.
19. Y. Shin, A. Meneely, L. William and J. A. Osborne, "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities," IEEE Transactions on Software Engineering, vol. 37, no. 6, pp. 772-787, 2011.
20. P. C. Jorgensen, Software Testing: A Craftsman's Approach, Boca Raton: Auerbach, 2008.
21. D. Bradbury, "Secure coding from first principles," 8 4 2008. [Online].
22. S. Stirling and J. Rainsberger, JUnit Recipes: Practical Methods for Programmer Testing, Greenwich, CT: Manning, 2005.
23. M. U. A. Khan and M. Zulkernine, "On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software," 33rd Annual IEEE International Computer Software and Applications Conference, pp. 353-358, 2009.
24. M. Dawson, D. N. Burrell, E. Rahim and S. Brewster, "Integrating Software Assurance into the Software Development Life Cylce (SDLC)," Journal of Information Systems Technology & Planning, vol. 3, no. 6, pp. 49-53, 12 2010.

Static Analysis is Not Enough:

The Role of Architecture and Design in Software Assurance

Walter Houser, NIST

"Software design errors are like space explosions. They are seldom heard and hard to spot. You do not see them coming but when they hit, they hit with more energy than they did when initiated. Once found, you spend the rest of the project dodging their debris. Spend your time and attention in the design phase. Knock out design flaws while they are small; do not wait until they are death stars. [1]"

- Gary Petersen, Shim Enterprise, Inc., CrossTalk, July 2004[1]

Abstract. Static analysis testing of software source code is necessary but not sufficient. Of the nearly 1000 CWEs, 40 percent can be introduced in the architecture and design phase of the development life cycle.[2] By their very nature, architectural and design flaws are difficult to find via static analysis. Furthermore, fixes to architectural and design errors can be complex, can inject additional defects, and can alert adversaries to the existence of these weaknesses. Moreover design flaws can obscure coding bugs that static analysis might otherwise detect, as demonstrated by the Heartbleed vulnerability.[3] This paper describes the techniques that architects and designers can employ to minimize the implementation of architectural and design flaws.

Introduction

Identification and mitigation of flaws early in the software development life cycle (SDLC) may avoid a ten to hundred-fold cost in post deployment detection and remediation.[4, 5] Yet cybersecurity reviews are often done just prior to an application's going live, typically because of a requirement for compliance.[6, 7] Architectural and design flaws found late in the SDLC can be costly to repair so they are often catalogued and not acted on until the next release (assuming the application survives that long). Moreover, by their very nature, architecture and design flaws are resistant to code patches. Fixes to these errors can further compound the problem by injecting additional defects. Moreover, patches can alert adversaries to the existence of these flaws.[8] Given that there are more than 61000 documented common vulnerabilities and exposures (CVE),[9] web application firewall rules can provide only partial mitigation against the exploitation of applications. This paper describes the techniques architects and designers can employ to minimize flaws in applications.

Definitions

The following definitions clarify concepts in this article:

Bug: A mistake introduced during the development, implementation, and sustainment phases of the SDLC.[12]

Flaw: A mistake introduced during the conceptual, architectural, or design phases of the SDLC.[13] Flawed code can be bug-free. "Microsoft reports that more than 50% of the problems the company has uncovered during its ongoing security push are architectural in nature. Cigital data show a 60/40 split in favor of flaws…"[14]

Software architecture: "…the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationship among them… In this regard, architecture is the primary determiner of modularity and thus the nature and degree to which multiple design decisions can be decoupled from each other. Thus, when there are areas of likely or potential change, whether it be in system functionality, performance, infrastructure, or other areas, architecture decisions can be made to encapsulate them and so increase the extent to which the overall engineering activity is insulated from the uncertainties associated with these localized changes."[15]

Software development life cycle (SDLC): The scope of activities associated with a system, encompassing the system's initiation, development and acquisition, implementation, operation and maintenance, and ultimately its disposal that instigates another system initiation.[16]

Vulnerability: An occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness.[17]

Weakness: A type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software. This term applies to mistakes regardless of whether they occur in implementation, design, or other phases of the SDLC.[18] For the purposes of this paper, weaknesses are categorized as being either flaws or bugs.

Common Weaknesses in Architecture and Design

In contrast to the dictionary of specific software vulnerabilities itemized in the CVE, the more general Common Weakness Enumeration (CWE)[10] is a dictionary of classes of flaws and bugs. Managed by MITRE Corporation under the sponsorship of the Department of Homeland Security (DHS), the CWE provides a publicly available, unified, measurable set of software weaknesses. As part of building the CVE, MITRE developed in 2005 a preliminary classification and categorization of vulnerabilities, attacks, faults, and other concepts to generalize the CVE into common software weaknesses. However, while sufficient for CVE, those groupings were too rough to be used to identify and categorize the functionality offered by the code security assessment tool makers. The CWE List was created to better address those additional needs.[11] CVE are akin to an inventory of traffic accidents while the CWE are analogous to the conditions (e.g. highway architecture, bridge and road design, traffic signage, driver practices, enforcement procedures) that lead to accidents. Given the large number of CVE and the process for reporting them, CVE references to architectural and design causes are intermittent and largely unverified. When generating findings

from code scans, static code analysis tools can draw upon the CWE for weakness descriptions and mitigation recommendations; identifying the relevant CVE would be difficult given their specificity and their disconnection from the code that generated them.

Thus the CWE enables more effective discussion, description, selection, and use of software security tools and services for finding weaknesses in source code and operational systems. Just as importantly, the CWE promotes a better understanding and management of software weaknesses related to architecture and design.

Although the CWE is most commonly associated with static code analysis reporting, MITRE classifies 381 of the total of 943 CWEs "as likely introduced in the Architecture and Design phase of the development life cycle."[19] Most of these CWEs are not apparent in the coding phase but can be identified during the design phase via the techniques discussed below. In other words, over 40 percent of the CWE can be addressed early in the SDLC in the architecture and design phase. But if not detected then, these weaknesses can slip past subsequent automated and manual reviews, with substantial cost and delays.

Three Perspectives of Architecture

To better understand these flaws and how to address them, we posit three perspectives for architecture: enterprise architecture, security architecture, and software architecture. The enterprise architecture perspective looks to the welfare and the effectiveness of the entire enterprise. The enterprise architect is concerned with the designing, planning and governing of strategic missions and programs. The enterprise architect seeks to optimize and harmonize services, processes, or components across the enterprise to achieve interoperability and portability. To enable the business objectives of the enterprise, the enterprise architect works with the respective software architects to leverage solutions across multiple systems and product lines. For example, the enterprise architect will consolidate individual systems into a service-oriented architecture and eliminate one-off personnel data sets, asset inventories, and expense tracking and reimbursement. The enterprise architect will promote adoption of a single technical architecture (such as Microsoft's .NET or Java Platform, Enterprise Edition) to avoid the headaches of interconnecting incompatible technologies. Portability and interoperability are the key architectural objectives.

Examples of flaws that can stem from a misunderstanding, or a misapplication of, enterprise architecture include:

- Erroneous business rules can lead, for example, to CWE-20: Improper Input Validation,
- Misunderstood user authorities and responsibilities can lead, for example, to CWE-272: Least Privilege Violation and CWE-200: Information Exposure,
- Errors in integration and module development can be linked to a deficient or missing SDLC strategy, [2] and
- Incompatible data definitions and inconsistent data management can lead to CWE-202: Exposure of Sensitive Data Through Data Queries.

The perspective of security architecture concerns itself with marshalling the controls, tools, and skills needed to protect the enterprise from external and internal threats. The security architect applies knowledge of the business goals and roles to identify and mitigate threat to the enterprise.

"… the first iteration of the analysis should take place when only the operational concept and a notional architecture is defined. Though the fidelity of the analysis may be fairly rough, this early stage is the perfect time to be considering what attacks your system could be facing, and whether there are design, architecture, physical composition choices or changes in operational concepts that could dramatically help to mitigate, manage, or control those attacks with minimal cost and schedule impact."[20]

The recent iOS Security White Paper[21] from Apple is a good example of security architecture. The White Paper covers the iPhone system hardware and software security, encryption and data protection, application security, network security, internet services, and device controls. Apple identifies the threats at each of these levels and describes how the architecture addresses them. "iOS protects not only the device and its data at rest, but the entire ecosystem, including everything users do locally, on networks, and with key Internet services."

Some flaws that should be addressed by security architecture include:

- CWE-260: Password in Configuration File
- CWE-261: Weak Cryptography for Passwords
- CWE-310 Cryptographic Issues
- CWE-330 Use of Insufficiently Random Values
- See CWE-254: Security Features for other CWE in this category.

The Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK®) asserts that software architectural design "(sometimes called high-level design) develops top-level structure and organization of the software and identifies the various components."[22] It is a top down process with various subcomponents and relationships between components. A software architect works within a business unit to identify stakeholders and their concerns, developing the architectural requirements for the systems that the business unit develops and deploys. The software architect coordinates enterprise coding standards and recommended practices with software development managers and software designers to reduce training costs and learning curves and promote portability of developers and applications across the enterprise.

The SWEBOK[23] distinguishes architectural design from detailed design which "describes the desired behavior of these components." However, this distinction will vary by organization. Some enterprises will not have software architects, instead dividing those duties between enterprise architects and system and software designers. In some cases the enterprise architecture will incorporate the higher level constructs of the security architecture, relegating the details to the developers and security engineers. The software architects could develop the security architecture, but this is not common given the special-

ized knowledge and skills of the security architect.

Some software architectural design flaws include:

- CWE-203: Information Exposure Through Discrepancy
- CWE-710: Coding Standards Violation
- CWE-289: Authentication Bypass by Alternate Name
- CWE-349: Acceptance of Extraneous Untrusted Data With Trusted Data
- CWE-280: Improper Handling of Insufficient Permissions or Privileges
- CWE-227: Improper Fulfillment of API Contract ('API Abuse')
- CWE-733: Compiler Optimization Removal or Modification of Security-critical Code
- CWE-14: Compiler Removal of Code to Clear Buffers

Maturity Model Context for Software Architecture and Design Controls

What can organizations do to mitigate or remediate architectural and design errors? Why not just fix the flaws identified in the CWE/SANS Top 25 Most Dangerous Software Errors? Before itemizing the anti-flaw arsenal, an SDLC-based framework can provide context for these controls and identify the gaps that will result from a purely tool and techniques point of view. MITRE[24] recommends that one should "treat the Top 25 as an early step in a larger effort towards achieving software security. Longer term strategic possibilities are covered in efforts such as Building Security In Maturity Model (BSIMM),[25] SAFECode,[26] OSAMM,[27] Microsoft SDL,[28] and OWASP ASVS."[29]

As an assessment of maturity models is beyond the scope of this article,[30] we will focus on BSIMM owing to its broad appeal to security practitioners seeking pragmatic and actionable content. BSIMM version V reports on the software security initiatives of sixty-seven firms drawn from twelve verticals (with some overlap): financial services (26), independent software vendors (25), cloud (16), technology firms (14), telecommunications (5), retail (4), security (4), healthcare (3), media (3), insurance (2), energy (1), and internet service provider (1).[31] Two of the four domains of the BSIMM Version 5[32] – Governance and Intelligence – list practices that are exclusively related to architecture and designs. A third domain – Secure Software Development Lifecycle (SSDL) Touchpoints – has the first of its three practices as Architecture Analysis. In all, seven of the 12 practices or categories of BSIMM activities precede the development phase of the SDLC.

BSIMM's governance domain encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates. This domain also identifies controls for compliance, setting organizational software security policy, and auditing. Also part of governance domain, training is critical because software developers and architects often start with little security knowledge. BSIMM's intelligence domain addresses organization-wide resources such as attack models to capture information used to think like an attacker (threat modeling, abuse case development and refinement, data classification, and technology-specific attack patterns). This domain also covers the creation of security patterns for major security controls, building middleware frameworks for those controls, and creating and publishing other proactive security guidance. Lastly the intelligence domain involves eliciting explicit security requirements from the organization, determining which COTS to recommend, building standards for major security controls, creating security standards for technologies in use, and creating a standards review board. BSIMM's SSDL touchpoints domain includes architecture analysis: 1) capturing the software architecture in concise diagrams, 2) applying lists of risks and threats, 3) adopting a review process, and 4) constructing an assessment and remediation plan for the organization.[33]

Security Controls for Software Architecture and Design

Given the BSIMM (or one of the other maturity models) to provide context, enterprises can integrate the following methods into their system development lifecycle process.

Threat Modeling

Threat modeling (aka threat analysis) is "a design-time conceptual exercise where a system's dataflow is analyzed to find security vulnerabilities and identify ways they may be exploited."[34] The process of identifying threats can reveal flaws in data handling, data sensitivity, authorization requirements, workflow, business logic, and hazards and other manifestations of failure[35] that can be addressed with changes in documentation rather than revisions to deployed code or re-engineered solutions. Moreover, static analysis tools lack knowledge of the operating environment and can only infer the potential threats facing the system being analyzed."[36]

Categorize data by information types[37] to establish data sensitivities, data sharing rules, and access privileges. Next, users can be grouped by privilege and their duties can be separated to prevent abuse and theft. The manner in which duties are separated may be an architectural determination derived from organizational missions; the software architect or software designer will need to ensure the appropriate stakeholders are identified and consulted before coding begins.

Data flow analysis can identify the appropriate levels of encryption and appropriate data handling, enforcement of encrypted data transfers between web browsers and servers, and encrypted backend data storage and transfer. Threat analysis can identify restrictions on the type and quantity of content a user can upload and download, as well as the file headers needed to enforce those actions.

Attack Trees

Attack trees, or threat trees,[38] provide a formal model for depicting the attacker perspective on the security of systems. The potential attacks against a system are arranged in a tree structure, with the ultimate objective as the root node. The leaves represent the various paths to achieving that objective. In Figure 1, the path to "obtaining the authentication credentials" uses network monitoring to recognize credential data. The arc labeled 'AND' indicates that the connected two leaf nodes both must

be completed before going to the parent node. Every arc that isn't labeled as an 'AND' is considered an 'OR' condition. Adding monetary values to the leaf nodes can indicate the financial cost for an attacker to accomplish the attack. With assigned monetary values, the designer can create countermeasures that increase the expense of attack routes.[39]

Misuse Cases

Misuse cases[40] are like use cases that present potential abuses of the system. Like use cases, misuse cases require understanding of the functionality to be provided by the application. A use case generally describes behavior that the system owner wants the system to implement. In contrast, misuse cases create conditions or situations that are undesirable in the view of the stakeholders. Misuse cases help organizations understand their software as the attackers would. Just as use-case models have proven quite helpful for eliciting functional requirements, misuse cases can effectively reveal security requirements. Use misuse cases to turn non-functional requirements into functional requirements.

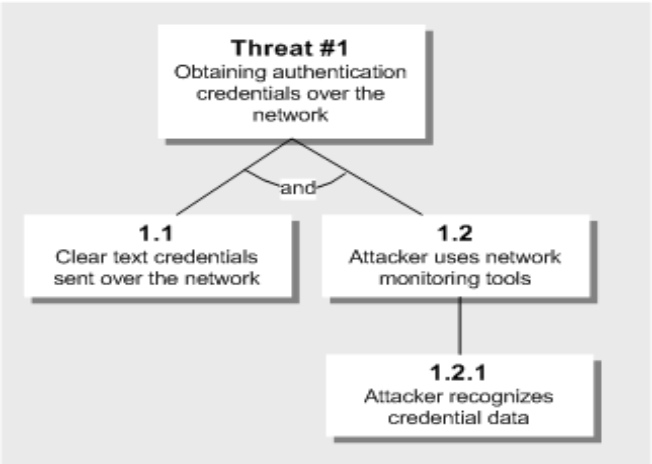


Figure 1: Attack Tree Example

Identity and Access Management

Owing to the complexity of the task, most applications use enterprise services for identity and access management (IDAM) of internal users[41]. However, an increasing number of applications are using outside social networks to authenticate external users. Although on-boarding for social networks tends to be far less rigorous than enterprise on-boarding processes, membership in a social network raises the threshold for malicious and spam accounts. However outsourcing IDAM for external users in this manner trades away user privacy to avoid the technical and management challenges in creating and maintaining accounts for external users. Furthermore such outsourced services are hard (or impossible) to scan for coding errors and review for design flaws, particularly if provided for free or in barter for customer usage data.

The software or security architect, or possibly the application designer, will need to identify and negotiate roles and responsibilities for IDAM controls. The architects and designers will need to agree on password complexity, resulting action for failed login attempts, session timeouts, user session refresh, forced re-authentication, re-authentication for privileged or sensitive data access, etc. The enterprise needs consistency for password reset and login input sanitization to prevent injection attacks. The software or security architect should ensure the IDAM interface enforces strict authentication and least privilege for administrative user access and prevention of session hijacking attacks against privileged users. The IDAM must meet authorization and accountability requirements, as well as provide logging and reviewing of administrative transactions for insider threats.

Least Privilege

The concept of least privilege is: "Every program and every user of the system should operate using the least set of privileges necessary to complete the job."[42] System designers can reduce the damage caused if a system is compromised by considering least privilege during threat modeling. A compromised application running with full privileges can perform more damage than a compromised application executing with reduced privileges. Most operating systems make little if any distinction in access privilege between a web browser and a word processer, despite the greater risks associated with the former's exposure to the internet. Sandbox applications to employ operating system security features to restrict the access available to sandboxed processes.

Formal methods

Formal methods are the incorporation of mathematically based techniques for the specification, development, and verification of software.[43] Owing to mathematical syntax and semantics, formal specification is precise when compared to non-formal and even semi-formal specifications that may be ambiguous or internally inconsistent. "Much anecdotal evidence suggests that formal verification can increase productivity, improve quality, and reduce development time by finding errors early and avoiding rework and testing delays."[44] Formality is used to achieve clarity of expression, to force early expression of precise behavior, and to allow more powerful verification and validation techniques to be applied."[45]

Formal methods add to the time needed to specify an application, yet they can improve the code quality and reduce testing and maintenance costs.[46] Formal methods should be employed in safety critical systems such as trains and nuclear reactors. Formal methods can be used in the design phase to build and refine the software's formal design specification, as well as employed in verification to prove that each step satisfies the requirements imposed by previous steps. Likewise, they can be used to improve software security, although they are not well understood and not commonly practiced in that context owing to the training and discipline required.

To compensate for a lack of experienced personnel and organizational support, formal methods can focus on algorithms,

components, properties, and other key functions in software, rather than applying them to the entire system. As for training limitations, it may be difficult to find developers with the needed expertise in formal logic, the range of appropriate formal methods for an application, or appropriate automated software development tools for implementing formal methods. Therefore, developers should employ formal methods for software likely to be reused or for critical functionality.

Secure Design Patterns

Secure design patterns[47] are descriptions or templates describing general solutions that can be applied in many different situations to eliminate or mitigate the consequences of flaws. By providing a higher level of abstraction than secure coding guidelines, secure design patterns can be applied across programming languages. Secure design patterns differ from security patterns in that the former do not describe specific security mechanisms such as access control, authentication, authorization, and logging. Whereas security patterns are focused on security-related functionality, secure design patterns can (and should) be employed broadly in a system.

Secure Session Management

The hypertext transport protocol is stateless[48]; the protocol does not maintain user state from one page to the next. Session management allows web applications to authenticate users at the beginning of the session by issuing a Session ID. This ID ensures that all actions during the session are performed by the same user (or web browser) that originally supplied their authentication information. Attackers will seek to manipulate the Session ID to steal the session from an authenticated user. Developers can defend against such attacks by requiring re-authentication when the session has timed out or when the user attempts to use sensitive functionality. The system designers must identify all functions where preservation of session state is necessary.

Conclusions

Over 40 percent of the CWEs can and need to be addressed in the architecture and design phases of the SDLC, especially as they are not usually apparent in the coding phase. They can be prevented, discovered, and mitigated via attack trees, threat models, misuse cases, and secure design patterns. Designers can use these techniques to identify sensitive information for encryption at rest and in transit. Session identifiers must be protected. Formal methods can improve software security with mathematically proven techniques. Lastly, managers should assure themselves that development teams are taking appropriate measures to prevent software weaknesses.[49] Figure 2 suggests how these solutions inter-relate and a possible sequence in which they may be applied.

Acknowledgements

Ram Sriram, Paul Black, Yan Wu, Vadim Okun, Bertrand Stivalet, and Clarence "Butch" Rappe provided useful feedback on earlier versions of this paper.

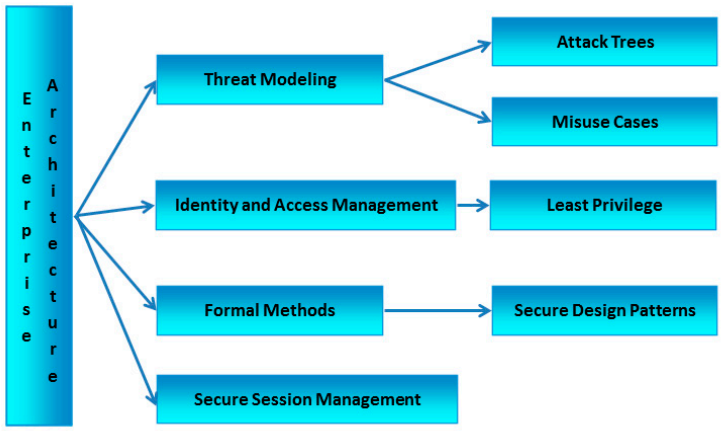


Figure 2: Security Controls for Software Architecture and Design

ABOUT THE AUTHOR

Walter Houser works in the Software and Systems Division of the National Institute of Standards and Technology (NIST) on the Software Assurance Metrics and Tool Evaluation (SAMATE) Project. There he investigates design flaws, code weaknesses, and reviews static analysis tool findings. Previously, he led information assurance projects with numerous Federal agencies and served as an IT policy officer, webmaster, applications development manager, and enterprise architect. Over 40 years ago he began his career as a COBOL and FORTRAN programmer; his work with SAMATE returns him to his professional roots to bring a coder's perspective to the challenge

Phone: 703-284-6180
E-mail: walter.houser@nist.gov

REFERENCES

1. Certain commercial entities, equipment, or materials may be identified in this article to describe an experimental procedure or concept adequately. Such identification is not intended to imply recommendation or endorsement by NIST, nor is it intended to imply that the entities, materials, or equipment are necessarily the best available for the purpose.
2. "The SSG [software security group] takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries. In addition to teaching by example, this middleware aids architecture analysis and code review because the building blocks make it easier to spot errors. " Also "Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review. Secure coding standards are necessarily specific to a programming language and can address the use of popular frameworks and libraries." Although the identification and maintenance of frameworks, libraries and coding standards is arguably a security architecture function, the decision to create an SSG and to establish and enforce these methods is an enterprise commitment. "We have observed that successful software security initiatives are typically run by a senior executive who reports to the highest levels in an organization. These executives lead an internal group that we call the Software Security Group (SSG), Software Security Group (SSG)…" McGraw, Gary. Sammy Migues, Jacob West. Building Security In Maturity Model, Version 5, October 2013, at <http://bsimm.com/>
`

NOTES

1. Shim, Gary, "Movie Physics and the Software Industry," CrossTalk The Journal of Defense Software Engineering, July 2004 <http://www.crosstalkonline.org/storage/issue-archives/2004/200407/200407-Petersen.pdf>
2. MITRE. CWE-701: Weaknesses Introduced During Design <https://cwe.mitre.org/data/lists/701.html>
3. A. Chou, On Detecting Heartbleed with Static Analysis, April 2014, at <http://security.coverity.com/blog/2014/Apr/on-detecting-heartbleed-with-static-analysis.html>
4. Kan, Stephen. Metrics and Models in Software Quality Engineering, 1995, P.171. "Defect removal at the earlier development phases is generally less expensive… Fagan (1976) contended that rework done at the I0, I1, and I2 inspection levels can be 10 to 100 times less expensive than if done in the last half of the process (formal testing after code integration)."
5. Jones, Capers and Olivier Bonsignour, The Economics of Software Quality, 2012, P.97. "The oft repeated aphorism that 'it costs 100 times as much to fix a bug after release as during design' is based on a flawed analysis that ignores fixed costs. Due to fixed costs, the following rule applies: 'Cost per defect is always cheapest where the number of defects found is highest.' … the cost-per-defect metric focuses too much attention on defect repairs and ignores the greater economic advantages of high quality in shortening schedules and lowering costs." Furthermore on page 111 … "Cost per defect penalizes quality and is always cheapest where the greatest numbers of bugs are found." And on page 112 "writing test cases and running them act like fixed costs."
6. Federal Information Security Management Act of 2002 Title III of the E-Government Act of 2002 (Pub.L. 107–347, 116 Stat. 2899).
7. Payment Card Industry Data Security Standard <https://www.pcisecuritystandards.org/security_standards/index.php>
8. Rains, Tim. Director of Trustworthy Computing, Microsoft <http://blogs.technet.com/b/security/archive/2013/08/15/the-risk-of-running-windows-xp-after-support-ends.aspx>
9. <https://cve.mitre.org/cve/cve.html>
10. MITRE. CWE–Common Weakness Enumeration. 10 Apr 2014 <http://cwe.mitre.org/>.
11. Ibid. "What is the relationship between CWE and CVE?" <http://cwe.mitre.org/about/faq.html#A.8>
12. MITRE. CWE-701: Weaknesses Introduced During Design <https://cwe.mitre.org/data/lists/701.html>
13. McGraw, Gary. Software Security. 2006. P. 14. Please note that MITRE does not make this distinction in describing the CWE
14. Ibid, p. 16.
15. Ibid, p. 18.
16. Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK®) A Project of the IEEE Computer Society, 2014, Page 7-5.
17. "National Information Assurance (IA) Glossary: CNSS Instruction No. 4009." Committee on National Security Systems, 26 April 2010, page 72. <http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf>.
18. MITRE. <https://cwe.mitre.org/documents/glossary/index.html#Vulnerability> MITRE.
19. MITRE. <https://cwe.mitre.org/documents/glossary/index.html#Weakness>
20. MITRE, Engineering for Attacks <https://cwe.mitre.org/community/swa/attacks.html>
21. <http://images.apple.com/ipad/business/docs/iOS_Security_Feb14.pdf>
22. SWEBOK® Version 3.0 IEEE Computer Society, 2014, P. 2-1
23. SWEBOK® Version 3.0 IEEE Computer Society, 2014, P. 2-1
24. <http://cwe.mitre.org/top25/index.html>
25. McGraw, Migues, and West, ibid.
26. Software Assurance Forum for Excellence in Code (SAFECode). Fundamental Practices for Secure Software Development, page 2. <http://www.safecode.org/publications/SAFECode_Dev_Practices0211.pdf> Software Assurance Forum for Excellence in Code http://www.safecode.org/publications.php
27. The Open Software Assurance Maturity Model <http://www.opensamm.org/>
28. Microsoft Security Development Lifecycle <http://www.microsoft.com/security/sdl/default.aspx>
29. Open Web Application Security Project (OWASP) Application Security Verification Standard Project (ASVS) <https://www.owasp.org/index.php/ASVS>
30. SwA Capability Benchmarking, <https://buildsecurityin.us-cert.gov/swa/forums-and-working-groups/processes-and-practices/swa-capability-benchmarking>
31. The list of 47 firms who have agreed to be identified is listed at <http://bsimm.com/community/>.
32. McGraw, Migues, and West, ibid.
33. McGraw, Migues, and West, ibid, page 19.
34. McGraw, Migues, and West, op.cit.
35. Critical Code: Software Productivity for Defense, 2010, Computer Science and Telecommunications Board, National Academy of Sciences. Page 92.
36. SAFECode, Op. cit.
37. NIST Special Publication 800-60 Rev. 1 Guide for Mapping Types of Information and Information Systems to Security Categories Aug 2008.
38. Swiderski, Frank, and Window Snyder, Threat Modeling, Microsoft Press, 2004.
39. Related to attack trees, "kill chains" can be reviewed by designers when developing attack trees. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains Eric M. Hutchins, Michael J. Clopperty, Rohan M. Amin, Ph.D. <http://papers.rohanamin.com/wp-content/uploads/papers.rohanamin.com/2011/08/iciw2011.pdf>. For an example of a kill chain analysis, see A "Kill Chain" Analysis of the 2013 Target Data Breach, Majority Staff Report, Committee on Commerce, Science, and Transportation, U.S. Senate. March 26, 2014 <http://www.commerce.senate.gov/public/?a=Files.Serve&File_id=24d3c229-4f2f-405d-b8db-a3a67f183883>
40. OWASP Comprehensive, Lightweight application Security Process (CLASP), <http://www.lulu.com/shop/owasp/owasp-clasp-v12/paperback/product-1869926.html>
41. National Strategy for Trusted Identities in Cyberspace (NSTIC), <www.nist.gov/nstic/>
42. Saltzer, Jerome and Michael Schroeder "The Protection of Information in Computer Systems.", 1975 <http://web.mit.edu/Saltzer/www/publications/protection/>.
43. Ghezzi, Carlo. Mehdi Jazayeri and Dino Mandrioli Fundamentals of Software Engineering (2nd Edition), Sep 29, 2002. 334-337 and pp 550-559. "Validation of the specification and verification planning required slightly more resources than expected,… The design activity turned out to be substantially simpler than in similar traditional projects, because of the higher quality of the specification."
44. P. Black, personal correspondence.
45. Ibid
46. The National Security Agency (NSA) asked Praxis High Integrity Systems to undertake a research project to develop part of an existing secure system (the Tokeneer System) using Praxis' Correctness by Construction development process. "Notice that the productivity during coding for the TIS core is higher than for the support software despite the core coding effort including static analysis. This is because there was very little rework of the TIS core software since the early lifecycle activities produced an unambiguous definition of the required software functionality… It should be noticed that almost half of the project costs were incurred prior to coding. This reflects the emphasis placed on correct construction of requirements, specification and design." <http://www.adacore.com/sparkpro/tokeneer/>
47. Dougherty, Chad. Kirk Sayer, Robert Seacord, David Svoboda, and Kazuya Togashi, "Secure Design Patterns, Software Engineering Institute, October 2009. <www.cert.org/archive/pdf/09tr010.pdf>. See also Architecture and Design Considerations for Secure Software, Version 2.0, 18 May 2012. <https://buildsecurityin.us-cert.gov/sites/default/files/ArchitectureAndDesign_PocketGuide_v2%200_05182012_PostOnline.pdf>.
48. MITRE. CWE-384: Session Fixation Document1<http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.htmlhttp://cwe.mitre.org/data/definitions/384.html> and CWE-613: Insufficient Session Expiration <https://cwe.mitre.org/data/definitions/613.html>
49. Department of Homeland Security, "Architecture and Design Considerations for Secure Software," Version 2.0, 18 May 2012. <https://buildsecurityin.us-cert.gov/sites/default/files/ArchitectureAndDesign_PocketGuide_v2%200_05182012_PostOnline.pdf>

A Software Requirement Tool for Capturing Implied Security Requirements

Hossein Saiedian, University of Kansas
Annette Tetmeyer, University of Kansas

Abstract. While expressing software requirements and needs, many clients, especially the non-technical ones, will indirectly imply concerns and expectations that are security related. One way to capture such implied concerns (i.e., security requirements that are not explicitly stated) is to use a parsing tool and look for terminology and keywords that indirectly (and perhaps sometime very directly) imply security requirements, constraints, or expectation. Such keywords will be tagged and further refined into formally specified software requirements and incorporated into the final requirements document. We introduce such a tool and a mini process for utilizing it. Our approach has the advantage of steadily incorporating security requirements engineering into existing software development processes with minimal disruption while adding value to the software development process.

Introduction

Scan and carefully review any software requirements specification artifact for security related terms such as "password", "encryption", "authorization", "integrity", "hacking", "accountability", "monitoring", "controlling", "event log", or even "security". Are these terms likely to be found within the artifact? Yes. Are these terms associated with a security specific requirement? Possibly. What does this imply? In many cases, security is implied within software requirements but may not be specifically considered a security requirement. In addition, the requirement that contains security terms may be vague and open to interpretation depending on the viewpoint of the reader. Implied security requirements may be creeping into software requirements due to increasing awareness of security needs from novice end-users to security experts alike. Data breaches, privacy issues and security concerns related to software are increasingly headline news and are raising the security awareness of a broad spectrum of the population. Software users increasingly expect security even if they cannot clearly define what security means. Software developers are responding by implementing security features to mitigate risk, but often this is on an ad hoc basis. Legislators at the state and federal level are also enacting regulation in response to security events. All of these responses are primarily reactionary in nature. The question is what should be done to improve the integration of security into requirements engineering from the start rather than reacting later?

Security requirements engineering (SRE) [1] is receiving more attention as a not only a valid, but increasingly necessary part of the software development process. Building security into software, much like building quality into products from the start, can improve the overall security of software and move from a reactionary to a proactive approach to secure software [2,3]. In order to build secure software, processes and tools are needed that address software security in the early stages of development as a part of the normal software engineering process. Regardless of the type of software development life cycle model (waterfall, agile, spiral, etc.) used, all include a requirements phase in the earliest stage of development. During development, functional and non-functional requirements are elicited, analyzed and developed into software requirements with the key goal being that these requirements are as final as possible. Requirements changes during development are the bane of software project managers due to the amount of effort needed to address these changes (particularly if changed during the late stages of development). It is highly desirable to develop stable requirements as early as possible to reduce the detrimental impact on cost and time as a project progresses.

One type of software requirement that can be overlooked is security requirements. Even the most casual of software users may expect a minimum level of software security even if they have a hard time defining security expectations. Defining security requirements can be difficult due to a lack of common ground among stakeholders in terms of security knowledge, skill, and even vocabulary. If stakeholders cannot define and understand security needs, then it is unlikely that security requirements will be properly elicited, captured and defined. Even if security requirements are defined, they are often seen as constraints on functional requirements and can be at odds with the goals of stakeholders. The cost of developing and implementing security requirements may also be a difficult sell to those signing off on the cost of the project. This leaves members of the software development team in a position in which they must anticipate the security goals of client and develop the appropriate security requirements as early as possible in development. Finally, security requirements must be justified based on a risk-reward analysis.

Security requirements engineering must become a prioritized part of the software development process and tools must be developed to aid in developing security requirements.

Best practices, enumerations, methodologies, models and elicitation techniques have been proposed that are intended to improve the integration of security requirements into early phases of development. A key factor in each of these is the focus on the first stage of software development or requirements development. Software developers who have previously not emphasized the development of security requirements must start including them in their software development processes. However, jump-starting an SRE initiative can be daunting and if initially unsuccessful, can be a detriment to future inclusion of security requirements. An alternative method, particularly for a small software development team, is to capture security requirements during the normal requirements process and build to a comprehensive security requirements engineering process. Stakeholders often have difficulty expressing security related needs but use terminology that implies a security need. By capturing these implied security needs, further elicitation activities can be undertaken to refine security needs into security requirements.

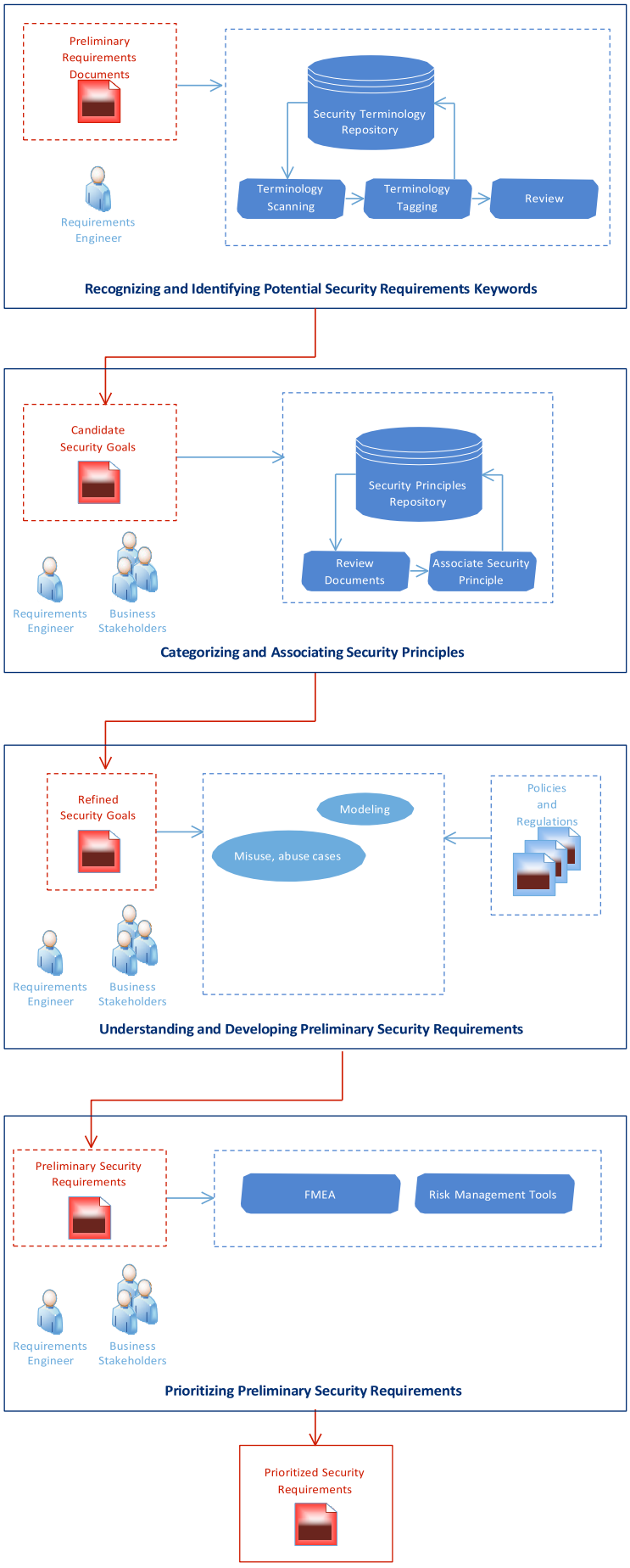


Figure 1: A Tool for Capturing Security Requirements

A Requirements Tool for Capturing Security Requirements

How can security requirements be inferred from general software requirements? One method is to parse the natural language that stakeholders use when defining requirements to extract security implied terms. During requirements elicitation, security related terms may even be included in general requirements without identifying them as security specific requirements. Identifying and extracting phrases based on these terms is the first step to understanding security goals. During elicitation activities, the requirements engineer can further extract security needs from stakeholders and lead prioritization activities in order to convert security goals into security requirements. A tool to identify, categorize, understand and prioritize security goals that is integrated into requirements elicitation activities can be the starting point for a security requirements engineering initiative. Figure 1 illustrates the software requirements artifacts input and output, stakeholder roles and the processes defined by the security requirements capturing tool.

Recognizing and Identifying Potential Security Requirements Keywords

The requirements engineer starts by identifying potential security goals based on security terms and phrases appearing in preliminary requirements artifacts. Preliminary requirements artifacts can be formal software requirements specifications, user stories, business process documents, or other documents related to requirements specifications. The type of requirements artifacts and software life cycle model used is not a constraining factor; the point is to identify implied security requirements based on terminology. The identification of these terms and location within the requirements artifacts are the starting point for security requirements elicitation.

Scanning can be manual for small artifact sets, but an automated scanning tool is desirable for larger artifact sets. An automated scanning tool can be easily implemented for common types documents (such as Word or text documents) and should identify the frequency and tag the location of security terminology passages in artifacts for further analysis. Prior to scanning, the requirements engineer would define a set of security terms or phrases in a security terminology repository. The starting set of terms can be based on the security knowledge of the requirements engineer or by using a common dictionary of security terms. Over time, the security terms and phrases would be refined by the requirements engineer for reuse on other projects. After initial scanning, the requirements engineer with review the tagged terminology to determine an initial set of candidate security goals for further requirements elicitation.

Categorizing and Associating Security Principles

Categorizing security goals and associating security common security principles will aid in gaining a deeper understanding of the stakeholders security requirements. The requirements engineer should be knowledgeable about general security and key security principles such as confidentiality, integrity, and availability principles (also referred to as CIA). A starting set of security principles will be defined and used to categorize and associate with

each candidate security goal. Working with business stakeholders, the requirements engineer will be able to extract a deeper understanding of needs while also building a common ground of security understanding with the business stakeholders. Tagged security passages should be refined from general or vague to more exacting language that clearly defines the security need. Categorizing based on a common set of security principles and including this language within the security goals will also aid in understanding. After an initial review among all stakeholders, any tagged passages containing security terms that are not deemed as candidate security requirements should be discarded from further review.

Understanding and Developing Preliminary Security Requirements

Using the refined security goals as input, the requirements engineer will continue elicitation activities with business stakeholders to develop preliminary security requirements. Structured or unstructured elicitation activities such as face-to-face meetings or review sessions will aid in communication and yield further understanding of the stakeholder's needs. Activities specifically related to developing security requirements such as modeling activities, developing misuse/abuse cases, or building attack trees can be undertaken at this time. Business artifacts such as policies and regulations should be used as input at this time. The goal of these elicitation activities is to gain a deeper understanding of the implied security goals and develop them into preliminary security requirements.

Prioritizing Preliminary Security Requirements

Next, candidate security requirements should be prioritized. There is always a trade-off to be made when determining which requirements are feasible to be implemented regardless of the type of requirement. Functional requirements are often cut from consideration if they are deemed too costly or do not meet a return on investment (ROI) threshold. Security requirements are not immune to analysis to determine if they are feasible. Attaining 100% secure software is not feasible. The software development team and business stakeholders should perform ROI or risk analysis to determine which candidate security requirements should be implemented. These activities will further enhance communication and foster familiarity with software security among all stakeholders. In the absence of a preferred analysis technique, Failure Modes and Effects Analysis (FMEA) can be utilized.

FMEA is an analysis and decision-making tool often associated with quality and Six Sigma methodologies [4]. A failure mode is the manner in which something might fail. Effects analysis is the study of the consequences of these failures. FMEA is used to identify, estimate, prioritize, and reduce the risk of failure. As a software engineering tool, FMEA is not widely used, but has advantages over other analysis tools in that it is easy to implement and can be used by a broad audience. A requirements engineer can use FMEA to elicit security related information from stakeholders, prioritize the data, and present an analysis of the risks associated. The prioritized risks allow for informed decision making to choose which actions to consider.

This approach is very useful to communicate and clarify the impact of technical materials in an easy to understand format.

Analysis requires creating severity, occurrence and detection rankings in order to determine a risk priority number (RPN). A standard scale for severity, occurrence and detection can be adopted as a starting point for FMEA analysis but experienced FMEA users may wish to develop more refined rankings scales. A standard scale ranges from a low of 1 for unlikely to a high of 10 for very high. The RPN is calculated as the product of the risk rankings:

RPN = (severity ranking)(occurrence ranking)(detection ranking)

The requirements engineer could generate a preliminary ranking of candidate security requirements and follow-up with business stakeholder or all stakeholders could be involved at the start of analysis. Rankings for severity, occurrence and detection are determined by the stakeholders and the RPN is calculated. The resulting RPN generates a prioritized list of potential security requirements. Using the FMEA results, requirements engineer and business stakeholders will refine the preliminary security requirements until a list of final security requirements has been generated.

Scenario to Demonstrate the Capturing Security Requirements Tool

The following scenario demonstrates the capturing security requirements tool. A software developer has been contracted to develop a software application for a small organization. The software developer embraces agile software development methodologies (face-to-face customer interaction, lean techniques and minimal documentation) and is accustomed to fast-paced project schedules. Preliminary requirements artifacts have been developed using standard word processing tools and the software developer's requirements document template. Scanning and tagging of the preliminary requirements artifacts have revealed an implied security need to limit the impact of "unauthorized" users. Working with business stakeholders, the elicitation activities associate the security principles of confidentiality and integrity with the use of the term "unauthorized" in the requirements artifacts. In this case, requested data needs to be protected against unauthorized disclosure (confidentiality) and as well as against unauthorized modification or destruction (integrity). The refined security goals are further understood and developed by reviewing relevant regulation as well as by developing misuse and abuse cases with the business stakeholders. Both the requirements engineer and business stakeholders are beginning to fully appreciate the need to refine these security goals into preliminary security requirements. A new set of preliminary security requirements is added to the requirements artifact document to address the impact of data requests by unauthorized users. The final step is to prioritize the preliminary security requirements use FMEA analysis. The requirements engineer and business stakeholders identify three failure modes and effects for analysis (see Table 1). Severity, occurrence and detection rankings are agreed upon and RPN's are calculated. The resulting FMEA analysis reveals that both data being viewed or stolen by an unauthorized user represents a strong risk to the business and need to be represented in the security requirements. Data corruption by an unauthorized user

Table 1: FMEA Analysis of Security Requirements

Failure	Effect	Severity	Occurrence	Detection	RPN
data request by an unauthorized user	data viewed	3	7	9	189
data request by an unauthorized user	data stolen	9	4	9	324
data request by an unauthorized user	data corrupted	5	4	4	80
Standard Impact and Rating Scale for Severity, Occurrence or Detection Very High (9-10), High (8-7), Moderate (4-6), Low (3-2), Unlikely (1)					

represents a lower risk and the business determines that this requirement does not need to be represented in the security requirements. Therefore, a new security requirement is written to address the impact of data requests by unauthorized users and is included in the final software requirements artifact.

A Brief Analysis of the Security Requirements Capturing Tool

In order to be successful, the overall approach (process and tools) must be both measurable and feasible. Measurability is a key to determining the success of any process or tool. Scanning and tagging of security terms and phrases provides the basis for benchmarking the use and importance to security requirements within existing documents. Over time, statistics can be gathered to refine the process of determining the importance of implied security terms within requirements artifacts. Risk analysis activities, such as FMEA, also provide the ability to analyze security requirements activities. Implementing an SRE process can quickly become overwhelming due to the complexity of software security, resources required and need for security expertise. All projects need to balance cost and resources in order to deliver on time and on budget. This approach combines both process and tools to feasibly meet project goals. The process of integrating security requirements into an existing requirements engineering process of capturing implied security needs by tagging security terms can be a feasible addition to existing processes. Additional resources from the development team (such as a security expert) are not compulsory and activities are integrated into existing processes. Existing requirements artifacts are used as the basis for SRE activities. An automated scanning tool to tag security terms is desirable, but can be built or acquired at minimal cost. Minor training may be required to implement FMEA or other analysis activities. In general, all of these activities can be implemented at any scale and can grow and mature with the needs of the organization. Other security frameworks, best practices and models can coexist side-by-side with these activities with minimal disruption. Therefore, the proposed approach is a feasible alternative to beginning and building a SRE initiative for any organization.

Summary

SRE is one part of a solution for secure software development. Security requirements can be captured by scanning and tagging security terminology within requirements artifacts to identify security goals. Elicitation activities further refine the goals into security requirements by associating each with security principles and developing a deeper understanding of stakeholders needs. Risk analysis prioritizes preliminary security requirements to determine a final list of security specific requirements. These newly identified security requirements are integrated into the final software requirements document. The development of security requirements is integrated into existing software development processes in order to build a SRE initiative.

Further efforts can be continually refined to build the basis for secure software development.

ABOUT THE AUTHORS

Hossein Saiedian (Ph.D., Kansas State University, 1989) is currently an associate chair, director of IT degree programs, and a professor of computing and information technology at the Department of Electrical Engineering and Computer Science at the University of Kansas (KU) and a member of the KU Information and Telecommunication Technology Center (ITTC). Professor Saiedian has over 150 publications in a variety of topics in software engineering, computer science, information security, and information technology. His research in the past has been supported by the NSF as well as other national and regional foundations.

Phone: 785-864-8812
E-mail: saiedian@ku.edu

Annette Tetmeyer is a Ph.D. candidate in computer science at the University of Kansas. Her research interests include security requirements engineering, HCI, and engineering education. In addition to experience in private industry, she has taught a variety of undergraduate and graduate engineering courses at the University of Kansas. She received her MS in Computer Science from the University of Kansas (2013) and BS in Mechanical Engineering from Iowa State University (1993).

Phone: 785-864-8812
E-mail: tetmeyer@ku.edu

REFERENCES

1. J. H. Allen, S. Barnum, R. J. Ellison, G. McGraw, and N. R. Mead, Software Security Engineering: A Guide for Project Managers: Addison-Wesley, 2008.
2. G. McGraw, S. Migues, and J. West. (2012). The Building Security In Maturity Model.
Available: <http://www.bsimm.com/>
3. G. McGraw, "The Security Lifecycle-The 7 Touchpoints of Secure Software-Just as you can't test quality into software, you can't bolt security features onto code and expect it to become hack-proof Security," Software Development, vol. 13, pp. 42-43, 2005.
4. ASQ. Failure Mode Effects Analysis (FMEA). Retrieved 11/21/13, from <http://asq.org/learn-about-quality/process-analysis-tools/overview/fmea.html>

Enhance the Team Status Meeting

David Saint-Amand, NAVAIR

Abstract. In keeping with the theme of tools and the processes they support, this submission to Open Forum highlights the team status meeting. These meetings help keep a team aware, engaged, and on-track. Their content is used to develop progress and status reports for management. In short, they are a critical tool in support of the project management process. The foundation of these meetings is the agenda which guides, focuses, and bounds the discussion. After years of practice, NAVAIR found that a few small changes to the way the agenda is presented enhanced the effectiveness of their team meetings.

The Traditional Meeting Agenda is a Dull Tool

At NAVAIR, as with other organizations, a typical team meeting probably uses the traditional one-page meeting agenda. This is usually a generalized list of meeting topics distributed either in an email message prior to the meeting, or as a paper handout at the start of the meeting. They are often short and look something like this:

Agenda
1. Call to Order
2. Management Minute
3. Project Status
4. Role Reports (customer interface, quality, etc.)
5. Individual Status ("around the table")
6. Goals and Risks Review
7. Action Items
8. Oh-by-the-ways
9. Adjourn

In practice, the PRT has found that this style of agenda is not very effective. While it may have worked for getting across a broad sense of the status of the project, it did not provide an opportunity for developing a depth of understanding. At best, only one or two people read the agenda prior to the meeting and the rest just glanced at it as the meeting began. As a result, there was usually only cursory preparation for the meeting. Overall project status was delivered either vocally, or by displaying the live version of whatever tool, if any, the team was using to track their project. Individual status reports from the team members were virtually all delivered verbally; and were usually not interesting, memorable, or useful. They tended to devolve into either one or two sentence declarations or rambling descents into minutia, and worst of all; most of the status information was not being captured in written form.

The Presentation-Driven Meeting

After coming to understand the shortcomings of the traditional approach to agendas, the PRT process improvement coaches began to make a series of changes to how their team's meeting agendas were created and presented. It was thought that modifying the process of collecting, displaying, and reviewing the team data would make the team status meetings more effective and less time consuming.

The changes worked, but what the coaches didn't expect was that the changes also made the meetings more interesting and even a little creative; engaging the team members and enhancing the overall status meeting experience. Adopting these changes can help virtually any team enhance their status meetings.

The Framework

What were these changes? They were to transcribe the agenda into a PowerPoint slide deck, give each team member control over their own slides, and place the whole assembly in a team status meeting folder on a network shared-drive that is accessible from both the individual work stations and from the room where the meetings are held (Figure 1). The PRT chose PowerPoint, but any presentation-friendly software should do.

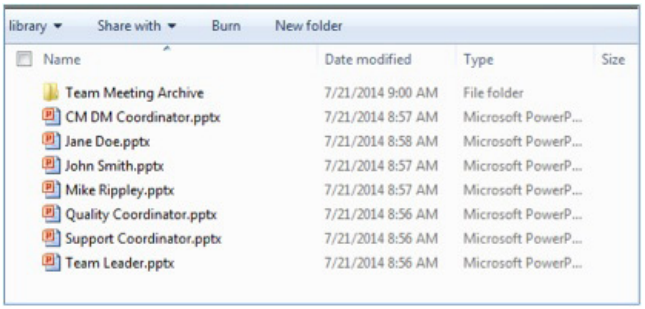


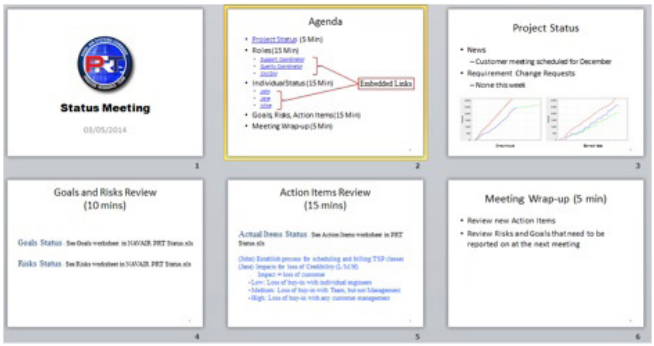
Figure 1: An example of a team meeting folder on a shared drive. The core of this set of presentations is the "Team Leader" presentation.

Being located on a shared drive eliminates many of the issues associated with accessing, updating, and then presenting team status information. While each person having their own status presentation can add up to a lot of files, the advantage is that everyone can update their project status without file-access-interference from other team members. The presentations can be projected for review during the meeting and they can easily be archived by copying the folder for the current week and renaming it accordingly.

The Team Leader's Core Presentation

At the core of the presentation-driven status meeting is the team lead's presentation. It includes the overall agenda, project status, and embedded links to supporting presentations from the team Role Managers and the individual team members (Figure 2).

Figure 2: An example of a team leader's core presentation.



All the necessary status information is covered in a consistent fashion with minimal disruption from moving back and forth between the core and supporting presentations.

The Supporting Presentations

First, the team leader should set expectations for the minimum required content of the supporting presentations and set firm boundaries for propriety. Then they should let the team take ownership.

At NAVAIR, the PRT found that those teams which followed this practice began to tailor the content and format of their presentations. They added and removed information as appropriate, and as they thought interesting. They used different backgrounds, graphics, and in several cases animations to make the material more visually interesting. In this process they became far more cognizant of their data and also more interested in the data being presented by others. Judging from the discussions that the new style of presentation inspired, team members showed overall increase in interest, understanding and retention.

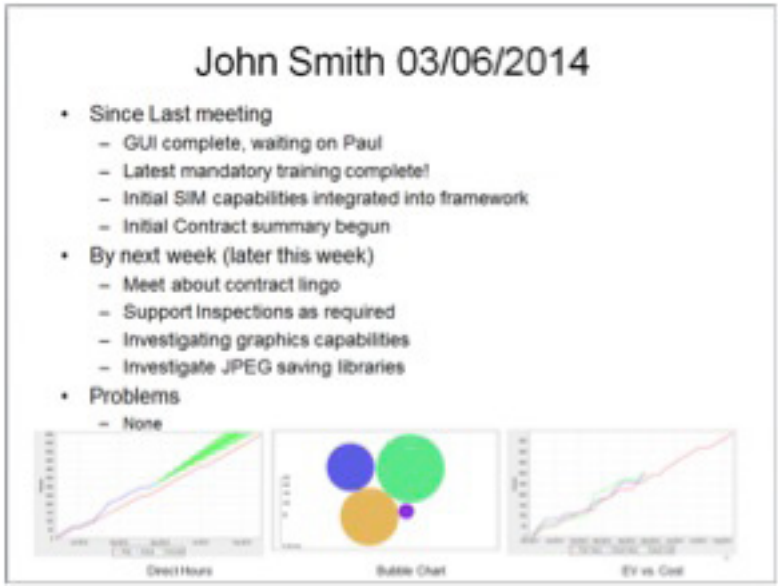
The Team Role Manager Presentation

A team Role Manager, or Role Coordinator, is a team member on a "self-managing" team who takes on some aspect of project management that is typically the purview of middle management: e.g. they track product quality metrics, monitor and report on the progress of the overall plan, or act as the requirements interface with the customer [1]. Prior to the institution of the presentation-driven status meeting, their reports were occasionally printed on handouts, but were more often just spoken, with a brief period afterwards for questions. Now they are concise, focused, and being captured.

The example of a Role Manager's report shown here is taken from a team where the quality measures for the work they were doing were not yet defined, so the Quality Coordinator focused on the quality of the process data being collected (Figure 3).

Figure 3: An example of a Role Manager Report presentation.

Figure 4: An example of an Individual Status Report presentation.



The Individual Team Member's Status Presentation

The next example is from an individual on a software development team member (Figure 4). The charts were taken from the process data collection and analysis tool chosen by the team: the Process Dashboard © 1998-2014 Tuma Solutions, LLC.

Using this format, the PRT teams are now getting complete status reports from the individual members. They are reporting their data and explaining it to their team mates. The increased transparency is uncovering issues, improving coordination, and encouraging cooperation.

Final Comments

By changing the way the agenda was presented, the team status meetings of those teams coached by the NAVAIR PRT have evolved into more relevant, easier to understand, and more visually interesting formats; the most innovative of which turned a mundane chore into a pleasant exercise. These meetings have focused information which often includes important details that might have escaped notice if the format had been rigidly controlled. Information is now presented in a manner where meeting attendees can inspect it, ask probing questions, and develop a more thorough understanding of the situation: both of the overall project, and of the status of their team mates. Even better, a complete and detailed project status can be easily captured and archived. By adopting a similar approach, any team can enhance their team status meeting.

ABOUT THE AUTHOR

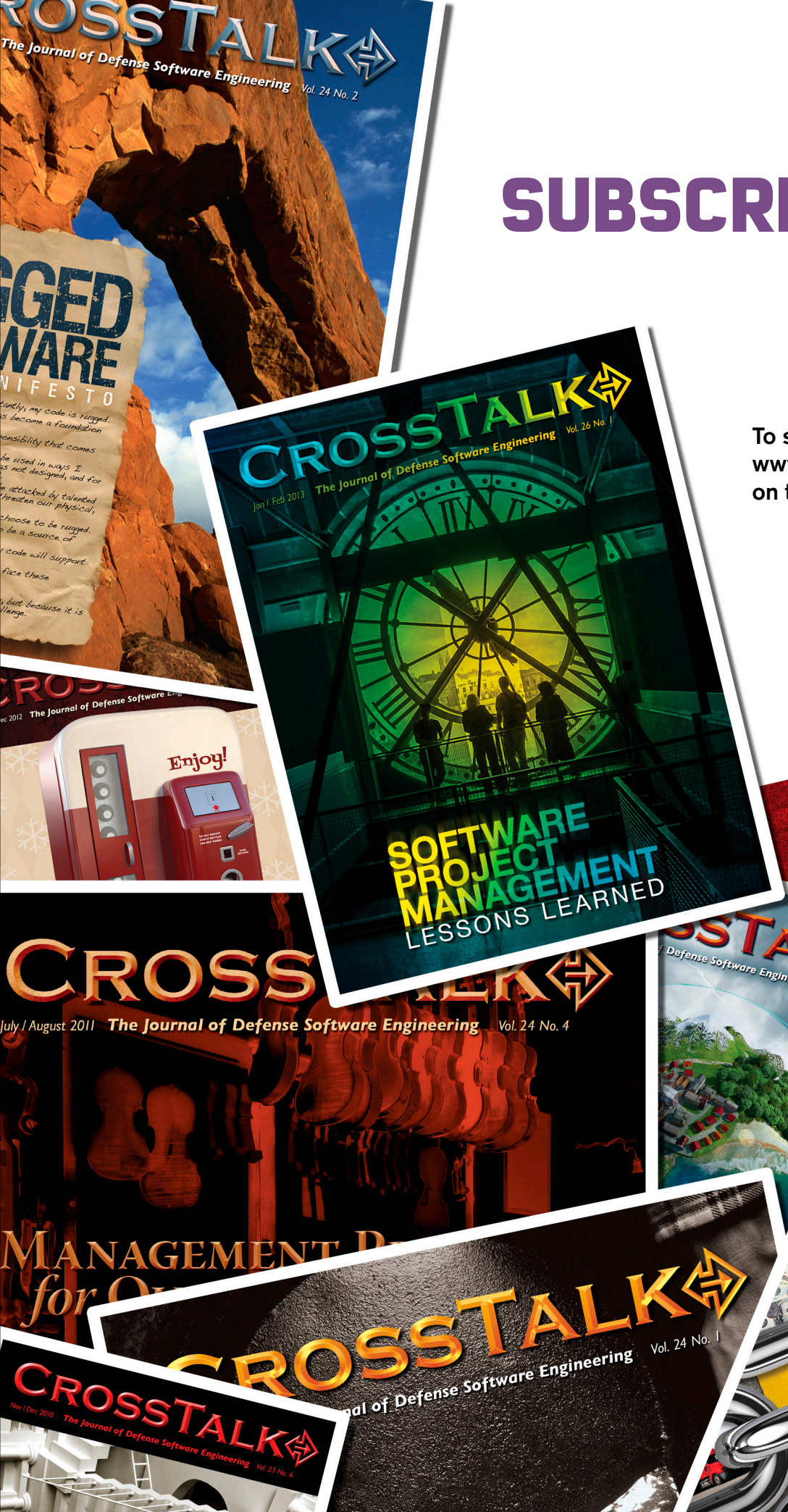
David Saint-Amand is a process improvement coach with the Process Resource Team of the Naval Air Systems Command (NAVAIR). His previous positions include DCS Corporation Section Manager, Naval Operations Research Analyst, Engineering Geologist, and Seismic Safety Consultant.

He holds a B.A. in Geology from the University of California at Santa Barbara with a secondary emphasis in Computer Science. He is a Defense Acquisition University Certified Level III Life Cycle Logistician, a Software Engineering Institute (SEI) Certified Personal Software Process (PSP) Developer, an SEI-Authorized PSP Instructor, and a NAVAIR Internal Team Software Process Coach.

Mail Stop 6308, 1900 N. Knox Rd.
China Lake, California 93555-6106
Phone: 760-939-2372
Fax: 760-939-0150
E-mail: David.Saint-Amand@navy.mil

REFERENCES

1. Humphrey, Watts S. TSP: Leading a Development Team. Upper Saddle River, N.Y: Pearson Education, Inc., 2006: pp 84-85 and pp 251-275.



SUBSCRIBE TODAY!

To subscribe to CrossTalk, visit www.crosstalkonline.org and click on the subscribe button.

Upcoming Events

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

2014 T3 Conference - Enterprise Edition
Hilton - Downtown Atlanta, GA
11-13 Nov, 2014
http://www.t3enterpriseconference.com/

COMSNETS 2015
7th International Conference on COMmunication Systems & NETworkS
Bangalore, India
6-10 January
http://www.comsnets.org

15th System-of-Systems Engineering Workshop
El Paso, Texas
27-30 January, 2015
http://www.itea.org/share/conferences-and-workshops

2015 International Symposium on Code Generation and Optimization
San Francisco, CA
7-11 February, 2015
http://cgo.org/cgo2015

20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming
San Francisco, CA
7-11 February, 2015
http://ppopp15.soe.ucsc.edu

3rd International Conference on Model-Driven Engineering and Software Development
Angers, Loire Valley, France
9-11 February, 2015
http://www.modelsward.org/

HotMobile 2015: the 16th International Workshop on Mobile Computing Systems and Applications
Santa Fe, New Mexico
12-13 February, 2015
http://www.hotmobile.org/2015

International Conference on Computing, Networking and Communications (ICNC 2015)
Anaheim, Ca
16-19 February, 2015
http://www.conf-icnc.org/2015

International Conference on Software Quality
Long Beach, CA
9-11 Mar, 2015
http://asq-icsq.org/index.html

ETAPS 2015 - 18th European Joint Conferences on Theory and Practice of Software
London, United Kingdom
11-19 April, 2015
http://www.etaps.org/2015/

WICSA 2015 – 12th IEEE Conference on Software Architecture
Montreal, QC, Canada
20-24 April 2015
http://www.computer.org/portal/web/conferences/calendar

Wibbly-wobbly, Process-y Wocess-y Stuff

Are you wondering what in the world the title of this column refers to? Let me start with a quick quiz—just one question: What was the most simultaneously watched TV event in the world?

Many of you are thinking the Apollo 11 landing, right? That's the obvious answer. Wrong. It was the Thanksgiving 2013 episode of Dr. Who, "The Day of the Doctor." According to Guinness, "World Records can confirm that the special 50th Anniversary episode of Doctor Who … has set a new world for the largest ever simulcast of a TV drama after the episode was shown in 94 countries across six continents. In addition to the TV broadcast, the episode was screened in over 1,500 cinemas worldwide, including the UK, US, Canada, Latin America, Germany, Russia and Scandinavia."

I am a "Whovian." I have been a fan of Dr. Who since the 1960s, have watched the escapades of every Doctor, and have a Tardis (Time And Relative Dimension In Space) replica in my office (for you non-Whovians, it's a replica of a British police box). A properly maintained and piloted TARDIS can transport its occupants to any point in time and any place in the universe. The interior of a TARDIS is much larger than its exterior, which can blend in with its surroundings and adapt to changing conditions using the ship's "chameleon circuit."

Which brings us to processes. In one of my favorite episodes of Dr. Who (Blink, from 2007), the Doctor has a great line, talking about time. "People assume that time is a strict progression of cause to effect, but actually from a non-linear, non-subjective viewpoint - it's more like a big ball of wibbly wobbly... time-y wimey... stuff."

Just like a process. A good process, from a non-linear viewpoint is not a strict progression of cause-and-effect. If it's going to work, it's more like a big ball of….well, wibbly wobbly process-y wocess-y stuff. Tools are not—you need a disciplined process that incorporates the use of your tools, and also modifies itself as needed to stay a useful process.

That's about the highest praise you can pay a process—it works under different situations, slightly modifying itself as needed. A process is a set of tools (or, actually, a meta-tool—it tells you how to use other tools). Need a screwdriver? Bet you have more than one! You have Philips, slotted, small, large, and that really old one at the back of your toolbox you use when you really should have a chisel. My Dad, who was a lifelong user of Sears Craftsman tools (which used to have a lifetime warranty, and for all I know, still do) used to say, "When you need a hammer, nothing beats a good Craftsman wrench!"

How do you know you have a "tailorable" process—one that is good enough to handle variations in the work process and use your tools effectively? It all about the maturity of the process—a reference to the Capability Maturity Model (CMM). Good processes produce better results—more reliable results. I was trying to come up with a good way to simply explain a good process, then it hit me.

See, I decided not to teach any classes this summer, and take the summer off. I spent a good deal of the time traveling (business related). I had a conference overseas, and then drove back-and-forth from Texas to DC twice. I drove to a family reunion near Atlanta, and took one final trip (driving) from Texas to Boston for my mother-in-law's birthday party. And, given all the driving, I spent a lot of time using my GPS. Wouldn't it be nice to have a process for directions? So, without further ado, I present Cook's Guide to GPS Process Levels.

Level 0 —You start out driving, having awoken in a strange car. You really don't know where you are, nor do you have a destination in mind. The GPS is broken, so you drive around a while, hoping that something will appeal to you. Every time you find something you think you like, you change your mind, return to your starting location (if you can find it) and start aimlessly looking again.

Level 1 —You and several of your co-workers need to find a drugstore, so you each take separate cars, and each drive in a different direction looking for one. Nobody has a working GPS. You find a drugstore by trial and error; and when you find one, you stand on top of the car, and yell, "I found one!" Few hear you, and nobody believes you.

Level 2 —You and several of your friends are still looking a drugstore. You all get in one car, and follow poorly written directions taped to the front of the GPS, directions that say things such as "Turn left where Sally's Bicycle Repair used to be" and "Take a slight left when passing where the big oak tree burned down." You forget to write down better directions as you drive, so you have to re-discover the route each time.

Level 3 —The GPS is pre-programmed with a single route to the drugstore. You have one route to get from your house to the drugstore. As long as nothing strange happens, you can always find the drugstore. If there is a detour—the GPS spends hours "searching for satellites."

Level 4 —You have a GPS that tells you how far to the drugstore and how how long until you get there, based on your speed. It will always pick the closest drugstore. It reroutes if you detour.

Level 5 —Your GPS receives traffic updates, reroutes as necessary, and takes you the whatever drugstore makes sense based upon traffic and construction.

Is your process like a wandering driver? Do you find yourself driving around, hoping to find your destination? Does it fail to tell you when or how to incorporate your latest tools? Or do you have a highly optimized and managed process, that updates itself as needed, and optimizes based on current events and needs.

Maybe it's time to update or replace your current GPS.

David A. Cook, Ph.D.
Stephen F. Austin State University