Adopting a Software Security Improvement Program

Dan Taylor and Gary McGraw

IEEE SECURITY & PRIVACY
http://www.computer.org/security/

Vol. 3, No. 3
May/June 2005

Adopting a Software Security Improvement Program

Dan Taylor and Gary McGraw
Cigital

Adopting software security in a large organization is a challenge that takes careful planning. Cultural change of any kind is difficult in big companies, and the potential minefields surrounding software process, development tools, programming language, platform, and other technical decisions only exacerbate the problem. Regardless of these issues, leading software shops (including Microsoft) are working hard to improve the way they build security into their products.

Software security initiatives have proven beneficial for those organizations that have implemented them. Such initiatives involve the adoption and rollout of the kinds of best practices described in previous installments of this department (see the "Software security best practices" sidebar). This article describes an approach that works, with an emphasis on business process engineering that might be unfamiliar to technical practitioners. By following some commonsense steps, a software security improvement program has a greater chance of achieving its ultimate goal: software security that makes business sense.

The business climate

Market forces continue to pressure IT organizations to become as efficient as possible to stay competitive. As a cost-cutting maneuver during the recent economic downturn, many companies reorganized their IT organizations and cut them to the bone. With no obvious cost cuts remaining, current efficiency efforts focus on improving productivity. By harnessing this productivity momentum, efforts to formalize software process improvement programs and achieve productivity goals are flourishing.

Any organization can initiate change, but few can sustain it over time. So how can we define and manage a change program in today's dynamic business environment? How can we prepare for and take advantage of natural change? How can we build a sustainable improvement plan that's flexible enough to adapt over time?

The first priority is to align software development and operational processes with strategic business objectives. Technologists sometimes forget why they're doing what they're doing, yet most software today is created to service business. Those technologists who understand that security is a risk management process that unfolds over time will have little trouble understanding that business concerns are a fundamental driver in balancing and refining security best practices.

The software security best practices we've covered are process agnostic and can thus be adopted regardless of an organization's software development methodology. Because every organization is different, a software security improvement program involving these practices must be tailored to the given business and technical situation.

A well-defined roadmap lays out the specifics of how best to deploy software security best practices given a particular organization's approach to building software. It helps prioritize change to assure that we first address only those program initiatives that provide the highest value or quickest return. This type of roadmap should follow five basic steps:

- *Build a tailored plan*. Recognize potential dependencies between various initiatives and plan accordingly. Keep in mind how your organization develops software and determine the best way to gradually adjust what you're doing to fold in security best practices.
- *Roll out individual best-practice initiatives carefully*. Establish champions to drive and take ownership of each initiative, coaching and mentoring as needed. Run a successful pilot in part of your company before attempting to spread best practices far and wide.
- *Train your people*. Developers and architects remain blithely unaware of security and the critical role they play in it, making training and mentoring a necessity.
- *Establish a metrics program*. Apply a business-driven metrics scorecard to monitor progress and assess success. Metrics and measures—even relative metrics based on risk over time or business metrics such as maintenance budget—are critical to making forward progress in any large organization.
- *Establish and sustain a continuous improvement capability*. Create a situation in which you can sustain con-

tinuous improvement by measuring results and periodically refocusing attention on the weakest aspects of your software security program.

The goal of the roadmap is to build and deploy an approach focused on enabling organizational change. When applied well, it will provide the building blocks for creating and sustaining change over time.

Building blocks of change

Every cultural change program requires buy-in from both management and tactical technical staff; improvement programs will fail if either group is left out or underemphasized. Every entity within an organization has a different sensitivity toward change, and these differences must be understood and accounted for because they deeply affect expectations. Disconnects in expectation could eventually end up forcing an organization into a least-common-denominator approach that lacks impact.

Keeping things simple helps people understand and support a program. Breaking down a major change program into logical segments with specific deliverables tied to each segment (or *initiative*) is a proven tactical approach. In practice, we find that a reasonable time range for any given initiative is three to four months. This type of stepwise approach minimizes risk while enabling an organization to test the waters as it gauges receptivity to change.

In terms of breaking down a program, we recommend a multiphased planning approach that accounts for the dependencies among related initiatives. Dependencies can help when adjusting the general sequence to account for those items likely to require a dependent task prior to being kicked off—building a set of measurement tools, for example, will directly depend on the software development methodology used. If an early segment includes selecting or adopting a given methodology, tool choice issues should be deferred to a later segment because they require an in-place methodology to be effective.

A clear sequence of initiatives helps an organization achieve a specific level of adoption, test the waters, measure and validate accomplishments, and set the stage for the next level. We follow a six-phase change program maturity-path sequence.

Software security best practices

A successful software security improvement program is based on adopting best practices like the ones shown with arrows in Figure A. These best practices are applied to software artifacts created during the software development process, but they are "process agnostic" because they don't assume that any particular methodology is being used. An overview of all software development life cycle best practices appears in an earlier installment of this department.[1]

Reference

1. G. McGraw, "Software Security," *IEEE Security & Privacy*, vol. 2, no. 2, 2004, pp. 80–83.

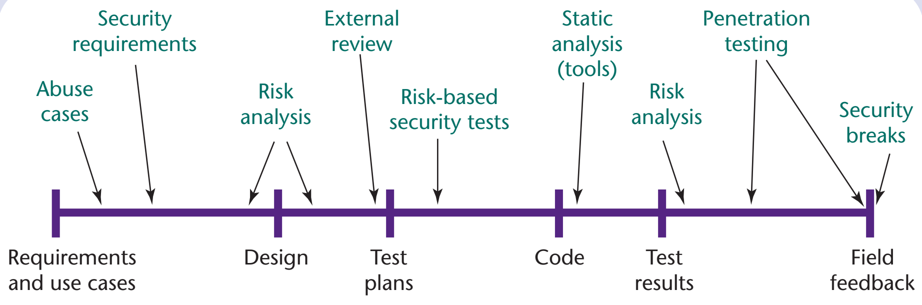

Figure A. Software development life cycle. We can apply several software security best practices (the arrows) throughout the SDLC, given a set of common software artifacts (shown along the bottom).

Phase 1

The first phase, *stop the bleeding*, is targeted at known problem areas in software development programs. If particular security bugs, such as buffer overflows, cause the biggest problem, a good phase 1 approach might involve the adoption of a code-scanning tool and an associated process for its use. If tens of thousands of security-critical applications have unknown risks, a good phase 1 approach might be to perform a high-level risk analysis of the application portfolio and organize the applications in order of criticality/security exposure so that the plan addresses those applications most at risk first.

Phase 2

The second phase, *harvest the low-hanging fruit*, focuses on finding the quick wins that are instrumental in getting buy-in from the organization and in helping a program build momentum. Both phase 1 and phase 2 are good barometers for determining the organization's receptivity toward change.

Phase 3

The third phase, *establish a foundation*, is about setting up components that provide building blocks for future initiatives. Typical tasks in this phase include creating change control programs, building a root cause analysis function, and setting up critical feedback loops. One such feedback loop can identify any security problems discovered via best practices such as code review and cycle them back into training (to teach developers how to avoid common security problem in the first place).

A well-defined roadmap lays out the specifics of how best to deploy software security best practices given a particular organization's approach to building software.

Phase 4

The fourth phase, *craft core competencies*, is driven by an organization's current as well as desired strengths. If an organization has a strong reputation for creating solid architecture documentation, for example, it will likely be more receptive to architectural risk analysis than it is to abuse case development. This phase explicitly involves the adoption of software security best practices in a manner tailored to the organization's strengths.

Phase 5

The fifth phase, *develop differentiators*, emphasizes and highlights those attributes and characteristics not easily duplicated by competitors, and that thus differentiate the organization from competitors. Measurement and metrics systems put in place with a software security improvement program can demonstrate how well things are going from a security perspective, which can serve as an important differentiator from the competition.

Phase 6

The final phase, *build out the "nice to haves,"* involves adopting those capabilities that aren't necessarily aligned to a given strategic business objective, but that bring value by achieving some improvement in productivity. We leave them last for obvious reasons.

Building an improvement program

Once we have a specific and actionable plan, a pragmatic approach should drive each initiative. A clear understanding of what will be built during each part of the program, who will own it, and how they will build, deploy, and continue to improve it over time is essential.

The general framework and plan discussed earlier should include several factors, including (but not limited to),

- tools,
- processes,
- decision criteria and associated actions,
- templates,
- examples and blueprints,
- best practices,
- guidelines, and
- metrics and measures.

Several other drivers, including current software architectures, security policies and guidelines, and regulatory requirements, can also help align the framework with the strategic business direction.

The most important decision for assuring success in a cultural change program is the selection of champions—those individuals who will build, deploy, drive, and own each initiative going forward. Should an initiative involve the adoption of static analysis tools for code review, for example, a champion well versed in implementation-level security analyses, the target language, and how to effectively use source-code analysis tools is vital. Ideally, such individuals won't be "freshly trained" in the area they're meant to own; rather, they should have a hand in developing the initiative and its components. A champion must be motivated, driven, and, most importantly, supported by the management team. They must also be good communicators and possess a strong capability to train and mentor others.

Establishing a metrics program

Overstating the importance of measurement and metrics is hard. Measurement provides management with critical insight into how to support strategic decision-making processes. Measures are numeric values assigned to a given artifact, software product, or process, and a metric is a combination of two or more measures that together provide some business-relevant meaning. When considered separately, for example, "lines of code" and "number of security breaches" are two distinct measures that provide very little business meaning because there's no context for their values. However, a metric comprising "number of breaches" per "lines of code" provides a much more interesting relative value: we can use it to compare and contrast a given system's security defect density against similarly sized systems and thus provide management with useful data for decision making.

Ideally, metrics and measures focus on four primary areas: project, process, product, and organization. The first three are specific to a given artifact or activity in a software development effort, whereas the latter's purpose is to determine trends across the three other areas.

Establishing a metrics capability is a challenging undertaking. Early standard software process approaches focused on sequentially building a level of sufficiency in four areas in a particular order: process, controls, metrics, and improvement. Unfortunately, following these basic steps in the prescribed order implies that we don't address metrics until late in the program. By then, it might be too late because processes and controls put in place early on might not be properly designed to provide the kinds metrics needed later. In such cases, some significant rework might be required to achieve business alignment. Consequently, it's now generally agreed that measurement and analysis must be included much earlier in the process development model.

All metrics should render decision criteria based on strategic business objectives. For this reason, business objectives must be articulated first and used to guide the entire program, from process and control development onward.

The targeted end state for any improvement program (security or otherwise) is a sustainable ability to evolve and change with the business climate. A critical foundation for continuous improvement is introspective in nature: each process must be carefully analyzed, assessed with respect to the need for change, adjusted as appropriate, and re-instantiated after refreshing. This feedback cycle is critical for ensuring that any given initiative stays relevant. Process for process's sake is a well-known pitfall that should be avoided. Unfortunately, many organizations have a tendency to become lazy and slip back into old habits; control processes can help counter this tendency.

A critical feature for the success of continuous improvement involves the periodic auditing and explicit reformulation of the organization's strategic objectives to ensure they haven't changed too much over time. If business needs have moved far enough to push processes and procedures off track, then the entire software security initiative must be reevaluated.

A critical challenge facing software security today is the dearth of experienced practitioners. Approaches that rely solely on apprenticeship as a method of propagation are unlikely to scale quickly enough; as the field evolves and establishes best practices, business process engineering can play a central role in encapsulating and spreading this emerging discipline more efficiently. ☐

Dan Taylor *is managing principal of Cigital's Productivity and Assurance Consulting practice. He is an active member of the American Society of Quality and the Information Systems Audit and Control Association. Taylor has a BA in psychology and quantitative analysis from Clark University and an MBA in finance from Rutgers University. Contact him at dtaylor@cigital.com.*

Gary McGraw *is chief technology officer of Cigital. His real-world experience is grounded in years of consulting with major corporations and software producers. McGraw is the coauthor of* Exploiting Software *(Addison-Wesley, 2004),* Building Secure Software *(Addison-Wesley, 2001),* Java Security *(John Wiley & Sons, 1996), and four other books. He has a BA in philosophy from the University of Virginia and a dual PhD in computer science and cognitive science from Indiana University. Contact him at gem@cigital.com.*