SYNOPSYS®

How to Build Security Into Your Software Development Process

Whether you use Agile, Waterfall, or something in between

BUILDING SECURITY INTO YOUR SDLC

can improve efficiency and reduce costs

if it’s done the right way

TABLE OF CONTENTS

Laying the Tracks

How SDLCs Work (at least in theory)

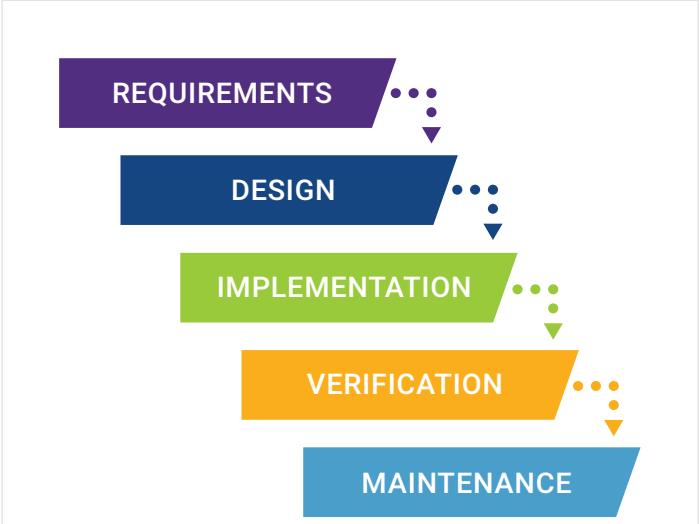
A software development life cycle (SDLC) defines the activities performed when developing an application or a piece of software. To standardize this process, organizations implement software development methodologies to ful ll their objectives in a way that best suits their organizational goals. Let's kick things off with an overview of two of the most popular software development methodologies: Waterfall and Agile.

What is Waterfall afterall?

The Waterfall model of software development is a sequential process in which progress is achieved steadily downwards through the different phases of the life cycle, often in a single pass with no iteration.

Waterfall is characterized by long phases.

These phases can take months, or even years, when major work is conducted. They also emphasize accountability and documentation which is preferable for government agencies and organizations that highlight accountability and predictability. Additionally, writing down what the software will do well in advance, and then building deliberately to a well-defined set of specifications is sometimes necessary—especially in long-lived systems that can't easily be updated once deployed.



However, nobody truly builds software in a Waterfall process. Everyone iterates. But there are aspects of Waterfall that are more prominent in some organizations than others.

Is Agile really all that?

Inspired by the Agile Manifesto, Agile is a software development methodology based on collaborative decision making between requirements and solutions teams. It is a cyclical, iterative progression of producing working software. Work is done in regularly iterated cycles, known as sprints, that usually last two to four weeks.

Agile is all about the sprint.

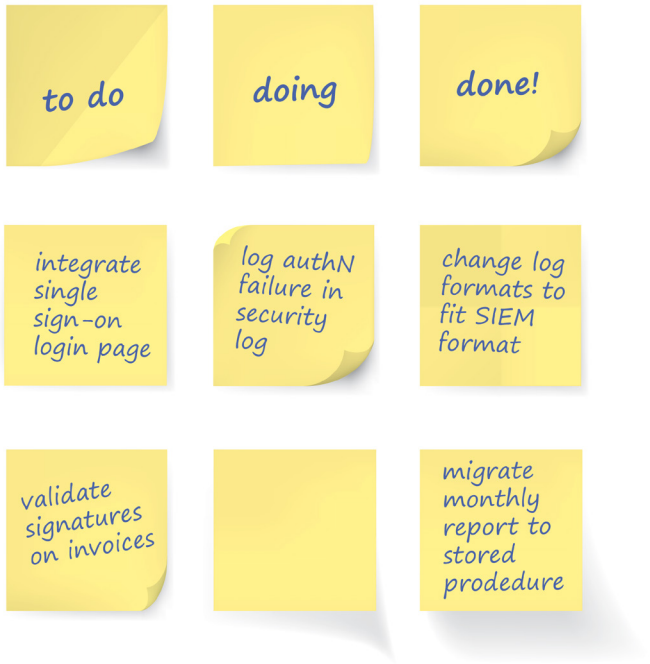
This is a key takeaway for security people to remember. If your security requirement is not in the backlog, it won't be scheduled for delivery in a sprint. If it isn't scheduled in a sprint, it won't get done. When security needs are articulated in the backlog, they are prioritized alongside everything else (new feature requests, bug reports, and so on). *The Agile process often includes*:

1. Daily stand-up meetings

A common fixture for Agile software teams are daily stand-up meetings (which are by no means restricted to Agile teams). The goal of these meetings is to talk about tactical concerns—the tasks that will be done that day and any complications that need to be addressed immediately. Security team members might attend stand-ups weekly, rather than daily, in order to get a high-level overview of upcoming implementations and problems blocking developers.

2. A Working System

Even in its earliest versions, software does something. It works, if only in a minimal way. Agile development aims to build something primitive, iterating over time to add more and more sophisticated features. If a feature cannot be implemented in a single sprint, it's broken down into incremental pieces that can.



3. The simplest thing that could possibly work

A well-known principle in Agile software development is the implementation of the simplest element (e.g. code, design, module) that will satisfy the requirements of the sprint. In other words, don't over-engineer!

In Agile, you often don't design for needs that could come up in the future,

EVEN IF IT SEEMS OBVIOUS

This is a point where development teams and security teams tend to struggle. Security teams aim to anticipate attacks, attackers, and risks. As needs emerge and are refined over time, security requirements can emerge that weren't anticipated at the beginning of the process. This is normal and natural in Agile, but it can be disorienting to security people who aren't able to secure against various likely attacks.

Moving Forward

How Software Development Works in the Real World

Pure Agile and pure Waterfall don't occur in the real world. Instead, software development takes place on a spectrum. There are infinite variations and most real world software teams incorporate elements of both Waterfall and Agile into their SDLC. Imagine Agile and Waterfall as theoretical ends on a continuum.

While these aren't the only two software development methodologies in existence, and they're not necessarily opposites, this visual allows us to understand that firms, no matter how hard they try, aren't purely one development methodology. There's a lot of grey area, but that's not necessarily a bad thing.

Software life cycles are as unique and varied as the organizations they serve. At each organization, security has to fit into that SDLC and give good, competent advice.

AN SDLC IN ACTION

Let's say that a company adopts long release cycles with extensive documentation. This company also plans intermediate releases every three weeks. This three-week release cycle feels very agile. They may even refer to each cycle as a sprint. However, the documentation they release and formality of the approval process reflect a waterfall approach. It's important to understand that these hybrid approaches aren't wrong.

If a firm's methodology achieves its aims as desired, then whatever approach they're using is working, whatever that might be. On the other hand, if the methodology isn't working well, then it's not necessarily true that making it "more agile" or "more waterfall" will make it any better.

NEXT STOP

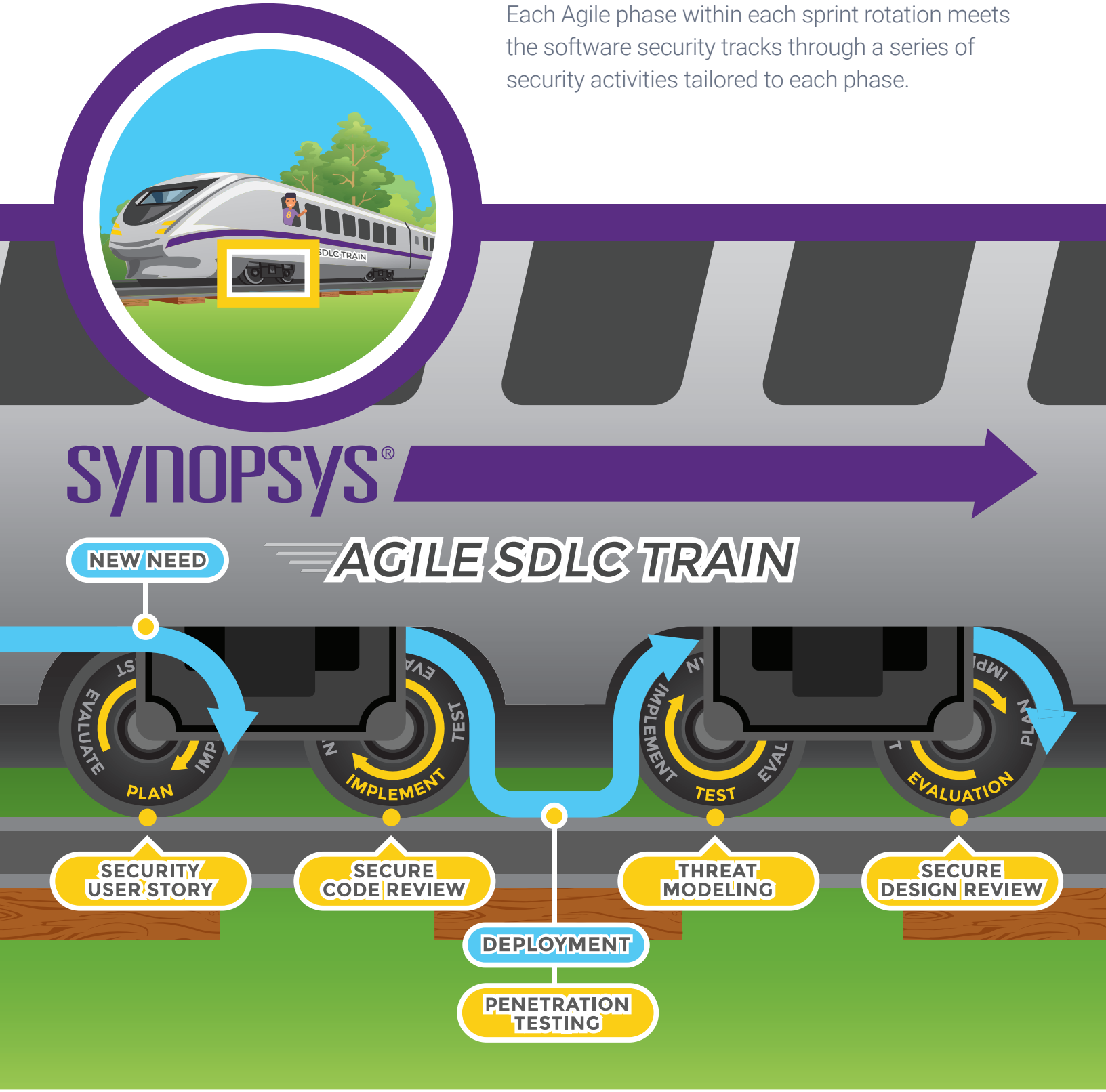
The Agile SDLC Train

Imagine Agile as a high-speed train flying down the tracks of software security. A train won't get very far without the tracks. Similarly, software isn't secure unless security measures guide the development process.

How the Agile SDLC train runs

Each rotation of a wheel represents a sprint. During each sprint rotation, new needs are coming in from the backlog, rolling through the planning, implementation, testing, evaluation, and deployment phases of the Agile SDLC.

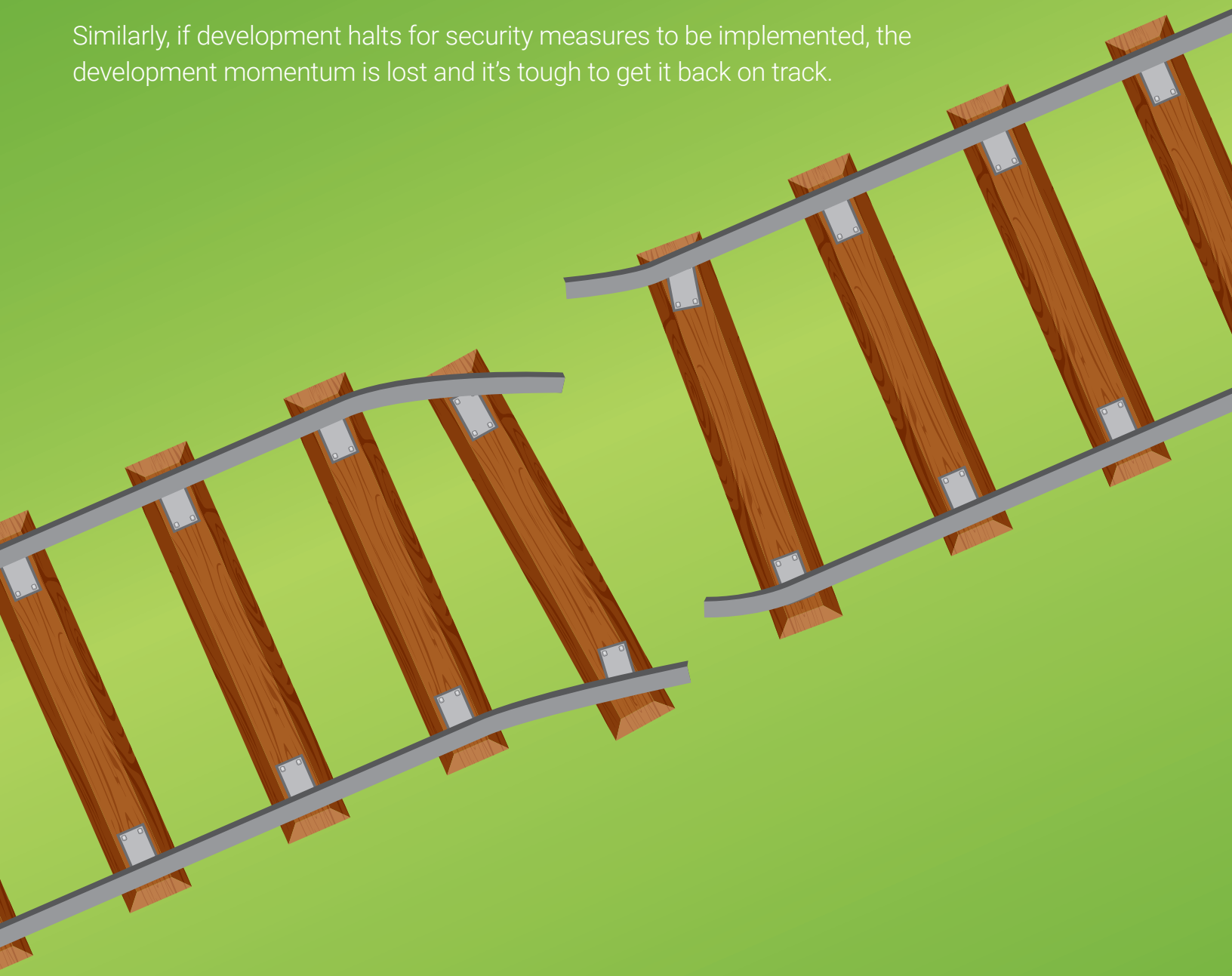
Each Agile phase within each sprint rotation meets the software security tracks through a series of security activities tailored to each phase.



Don’t jump the tracks

If the track is broken, the train will either crash or will need to stop so the track can be fixed before travel resumes.

Similarly, if development halts for security measures to be implemented, the development momentum is lost and it’s tough to get it back on track.



Fasten Your Seatbelts

7 Spots You Can Add Security Into Your SDLC

Regardless of the methodology your organization follows, all organizations developing software produce artifacts such as use cases, user stories, operational configurations, and code (among others). It's important to secure software by securing the artifacts that are produced throughout the SDLC. Most importantly, the processes that are in place around the production of the artifacts must include security measures.

Below we have listed seven SDLC artifacts and suggestions for integrating security into each of these processes.

1. Stories and requirements

One way or another, you document what the software ought to do. Right? Right. When you do, we suggest adding security to your plotline.

What are you trying to achieve?

The primary goal of incorporating security into the requirements and story phases of the SDLC is to make security a natural outcome of the development process. Stories are scored, groomed for the backlog, scheduled, developed, and tested.

The underlying principle is to ensure that the SDLC delivers security, which means that security team members need to translate the needs into a form that is digestible by the development team.

There is an old saying in Turkish:

"Dağ sana gelmezse, sen dağa gideceksin."

It translates to:

"If the mountain won't come to you, you must go to the mountain."

In other words, development will not come to security. Security must go to development.

Pitfalls and gotchas

There are a number of usual suspects when it comes to implementing security into the early phases of the SDLC. Here are some common mistakes to avoid:

- **Writing negative requirements.** Good software requirements and stories tell you what to do. By contrast, poorly written security requirements and stories often list vulnerabilities and exploits that software should not do. Software has a mission to accomplish something. Writing a long list of things not to do doesn't help developers do their work.
- **Turning security into a feature.** If security stories aren't articulated clearly, they can be turned into security features. For example, if the story stipulates that a green padlock needs to appear "when the connection is secure," the developers may simply include a green padlock. However, if the definition of a secure connection is ambiguous, the connection may not actually be secured.
- **Putting security in the wrong place.** Sometimes security stories and security requirements are implemented in the wrong part of the business. For example, let's say that online store A puts a robust credit card checking and validation process directly into their shopping cart checkout process. Store A won't accept an order unless the credit card passes fraud and other established approvals. Online store B accepts orders with minimal checks in the checkout process. Unlike store A, store B performs fraud and approval checks asynchronously in the order process (after accepting the order, but before fulfilling it).

  Neither of these approaches is objectively wrong. But, sometimes security controls are stipulated in a way that is wrong for a particular business. Adding robust card checks to store B's system might create problems for store B's established processes, even though it is technically possible. Security has to walk a fine line between prescribing the solution and describing the solution.
- **Having the wrong user in the story.** The users in user stories must be important to the software's mission. The developers of the software are not typically the appropriate user for user story inclusion. After all, they are the builders of the software, not its users. In this common pitfall, if the user stories are about the builders rather than the users, such stories will be countercultural.
- **Creating generic stories.** Generic security stories are often impossible for a team to assign points and prioritize correctly. A user story needs to cover functionality that can be delivered in a single sprint. (Alternately, it can be broken down and delivered incrementally over multiple sprints.)

The right way to do it

To successfully integrate security requirements into a user story or requirements document, there are several important principles to consider:

- **Express security requirements naturally.** User stories or requirements that include security elements should resemble regular requirements and stories as much as possible. Security stories need to sit in the backlog comfortably with other stories. They need to be schedulable and testable just like anything else. This way, the development team can apply the same processes to the security requirements as they do to all other requirements. The more you make security needs look like any other software needs, the more likely the development team is to deal with them quickly and effectively.
- **Describe affirmative behavior.** Whenever possible, discuss what the software does, rather than what it doesn't do. Additionally, describe a condition that occurs when something bad happens. For example, if too many password attempts are made in a certain period of time, what does the software do? Does it display an error? Does it impose a waiting period? Does it send an alert to an administrator's email?
- **Avoid purely technical stories when possible.** Technical stories tend to describe functions that aren't user-facing, but they're necessary to deliver what the users want. There is usually a limited scope in each sprint for technical stories, so stories that actually impact the users are more likely to get prioritized and delivered.

  If the user story says something affirmative, exposed to the users, and testable, then the normal business process can take its course. The functionality can be prioritized, the cost/benefits can be discussed, and the usual activities make the story work.

2. Code

Half of all software defects occur in the implementation—in the code itself (as opposed to the design). Mistakes in the code result in bugs. Many of these bugs have security implications. Attackers count on these bugs. They know how to find and exploit them. Luckily, the nature of these mistakes and attacks are known and there are very common ways to fix them that make the attacker's life much harder.

> *"All software projects are guaranteed to have one artifact in common—source code. Because of this guarantee, it makes sense to center a software assurance activity around code itself."*
>
> – Gary McGraw, *Software Security: Building Security In*

What are you trying to achieve?

While developing code, it's malleable. You can adjust, fix, and enhance it with much more freedom than after it's been deployed. The goal is to locate, resolve, and prevent problems in source code while writing it. Most organizations have too much code and not enough staff to do something naïve like reading through it line by line looking for vulnerabilities. You need practical means to either check the code while you write it, check the code you already have, or avoid the mistakes in the first place.

Pitfalls and gotchas

In practice, finding mistakes in code takes considerable effort. It's important to remember that:

- **Manual code review doesn't scale.** Manual efforts alone can't get the job done unless you're only dealing with very small amounts of code. Manual code review is time consuming and requires an extremely high level of expertise to look for a wide variety of vulnerabilities at the same time. Humans also get tired and lose focus over time. Today's high-speed delivery pressures also make manual code review unrealistic as the sole means of finding vulnerabilities.

- **SAST tools are a collection of trade-offs.** Because static application security testing (SAST) tools are created to focus on speed, the depth of their analysis suffers, resulting in more false positives. Conversely, if tools emphasize finding fewer false positives, they run slower.

  Inexpensive and free tools exist, but there is less expertise and less original research behind them. Tools with more expertise and more research cost more. It's important to note that tools are also specialized. One tool may be very good at catching some classes of bugs, another tool may be good at catching other classes of bugs; none are likely to be good at catching all classes of bugs. No tool is good at all languages or all frameworks. These trade-offs will affect the tool results. Bad results waste developers' time and reduce their confidence in the process.

- **You must also account for security outside of the code.** All tools suffer from a lack of understanding the environment regarding the software they are analyzing. They also lack a real understanding of the context of what they are analyzing. Because a tool only looks at the code, it can't know how environmental security precautions might effect a piece of software.

- **Tools lack context.** Tools have little, if any, understanding of the context of what the code is doing. This is why so many flag all instances of random number generators or all uses of date/time functions for inspection by the reviewer. The tool cannot know if the random number generation is rolling dice or providing a security function; it may not be able to tell if the date/time function is just a normal use or if it is there to trigger a time-based backdoor or logic bomb. The tool simply flags them for the reviewer to figure out based on the context of the function's use.

  Additionally, while they provide remediation advice for what they find, the advice is usually generic in nature and not tailored for the specific code in question. Developers cannot bounce ideas off a tool or have a discussion of the issues or their solutions.

The right way to do it

Once the SAST tools have scanned the code, the best approach is a human review of the tool's results. This allows the tools to do the heavy lifting and tedious work while the manual reviewer can focus on likely problem areas and business logic considerations. To review code the right way, it's important to:

- **Tune the tool.** Conducting a manual review after running the tool helps identify areas where the tool can be optimized to provide more reliable results. For instance, custom rules can be established or a simple configuration change allows the tool to better understand what various parts of the code are doing. Keep in mind that no tool runs at its best in a default configuration.
- **Perform a manual follow-up.** The reviewer understands the tool, what rules provide reliable results, and what rules provide weak results. Based on this knowledge, they are able to sort through the results before they ever get to the developer—shielding them from the noise that all SAST tools create. The reviewer also has the ability to help the developer understand the issues and answer questions in ways that tools cannot. Manual follow-up to the tools helps overcome their trade-offs.
- **Let insight guide training.** These human reviewers of SAST results can glean higher-level lessons about the code and the team. They can spot consistent issues in a certain class of bug, or particular teams with higher rates of security defects, for example. This insight can guide training efforts for one-on-one training or broader group efforts. This, in turn, can keep the problems from recurring over and over again; reducing the time spent fixing the same old bugs and making tight schedules easier to meet.
- **SAST as you type.** There are lightweight tools that actually check code while the developer works. They often integrate into the IDE as a plug-in. These tools operate like a spell checker or grammar checker works in a word processor, scanning as the developer codes. If the code looks potentially vulnerable, the tool highlights the lines of code immediately and prompts a description of the potential vulnerability.

Some SAST tools that examine the code can run automatically. Coverity and SecureAssist, for example, check code for issues anytime a file is opened or saved. When tools automatically check a developer's code, they provide a secondary benefit of actually teaching the developer to avoid making these common mistakes. Other tools require the developer to explicitly, manually launch the scan, but they still provide tremendous value by finding common bugs.

3. Unit tests

Written as developers write code, unit tests comprehensively cover a wide variety of input and situations that a given module might experience. At the same time, unit tests are highly localized. They don't have an end-to-end view, and can only test within a narrow context.

What are you trying to achieve?

Developers write unit tests to provide deep test coverage of isolated components in code and differ from other types of testing by conducting them during development. In test-driven development, developers write unit tests first and then code against them until the tests pass. Security unit tests will, for example, simulate malicious input that somehow slips past upstream or downstream filters that are intended to filter or correct it.

Pitfalls and gotchas

Security in unit tests should follow reasonable boundaries, testing appropriate security concerns based on the functionality of the application layer being tested. Beware of:

- **Building unit tests to confirm discreet elements of functionality.** Without integration tests, along with the developer's knowledge of the system, unit tests alone cannot detect bugs caused by the lack of clarity regarding the interface between two modules.
- **Assuming that the security control is implemented elsewhere.** An all-too-common pitfall is to assume that the important security controls for the software are all implemented somewhere else. Unit tests often lack checks for malformed input, degenerate data structures that are inherited from elsewhere (e.g., a user object with no role assigned), and illogical combinations of inputs. A naïve interpretation of the programming contract allows naïve unit tests that do not simulate malicious invocations of methods.

The right way to do it

When writing unit tests, developers need to address security concerns with the same level of care that they would give to functional issues. Give special consideration to the context. Some security tests may fit best within the context of a variety of considerations, while others may be best suited as stand-alone testing modules. For example, authentication and session management would be appropriate components to test with mainly security-focused unit tests. Tests of input validation, on the other hand, may best fit with a more general unit test suite covering business logic.

4. Integration testing

Once all unit tests pass, integration testing combines and tests individual software modules as a group. This testing approach has been gaining visibility in recent years with the emergence of service-oriented architecture (SOA), RESTful APIs, and microservices. During the integration testing phase, QA and security teams work together to create a large suite of functional and security tests.

What are you trying to achieve?

The integration testing phase gives security teams the opportunity to verify whether security requirements and user stories are implemented correctly on a running system. Scope and coverage of security tests are subjective regarding the level of modularity of the application or piece of software. For instance, a well-designed microservice architecture enables support for better functionality and security test coverage.

Fundamentally, integration testing is the time to start checking assumptions about how security mechanisms work. This is the time to create end-to-end security tests that check cross-module security assumptions.

Pitfalls and gotchas

Some things to be aware of when doing integration testing include:

- **Third-party APIs require special consideration.** A security testing strategy is based on whether the external module is upstream or downstream to the module being tested. It's a common practice to use "mocks" to test the integration with external third-party interfaces. These simulations are used to achieve faster test execution and should be handled with caution.
- **Server configurations and testing mocks aren't always production-ready.** Integration testing is performed on a deployable build, but that doesn't necessarily mean that all configuration settings are the same as they are in the production environment.

  Additionally, using mocks and simulations during integration testing with external modules doesn't always allow the execution of all possible real attack scenarios. Some test coverage is only possible on a real production environment that has final production configurations and is fully integrated with real-world modules.

- **DAST and SAST are only possible in limited measures.** The coverage of dynamic application security testing (DAST) is limited since some of the aspects of client-side logic are not reproducible during API or service-layer testing. The use of simulations or mocks also limits DAST coverage. These limitations can be addressed by executing DAST against a production-like user acceptance testing environment.

  Performing static application security testing (SAST) during the integration test phase has the same pitfalls that are seen during code development. The key difference between SAST while coding and SAST during integration testing is the availability of deployable binaries in addition to source code. There are various SAST techniques that operate on binaries instead of source code. Those tools can be used on the binaries that are available during integration testing.

- **Context is king. SAST tools lack the operational context of the software.** They can't account for security controls that are applied elsewhere. Likewise, many security controls are missing in test environments that are present in production (e.g., rate limiting, network edge devices, identity and access management systems).

The right way to do it

To ensure you get the most out of integration testing, remember to:

- **Test data and procedures for that hard-to-reach business logic.** QA teams work with security testers, providing test data, test tools, and test procedures that execute business logic without having to complete end-to-end flows. Likewise, QA testers know how to find log files and enable debugging output in test environments.
- **Perform fuzz testing.** Most fuzz testing is either generational or mutational. Generational fuzz testing generates input data randomly (typically with some constraints to ensure they fit a basic format). Mutational fuzz testing takes a known good input (e.g., a packet, web form, or data file) and mutates specific parts of it randomly.

  Fuzz testing differs from DAST because dynamic tests use carefully crafted attack strings (e.g., cross-site scripting), whereas fuzz tests are random. Applying fuzz testing during the integration testing phase can uncover how unexpected inputs cross modular boundaries.
- **Scope DAST to an individual module.** DAST tools often want to test an entire application. But, when software is built from individual modules, you can test those modules independently to discover vulnerabilities. You can also scope dynamic testing to an individual module, or a subset of components. If you cannot scope the tool, you can filter the results themselves so that only results from testable modules are considered. You can use many DAST tools, such as web proxies and fuzz testers, in a limited scope for cases such as these.

5. User accceptance testing

User acceptance testing (UAT) is often the last testing phase before a product is released into production. The UAT environment is the most representative of the production environment. Here, business users are re-engaged with the software, testing the product to ensure that it fulfills the planned business requirements for the current iteration (e.g., a formal release, a sprint, or something in between) and formally accepting the product and/or changes.

What are you trying to do?

The UAT process is much more than validating the user interface (UI) of the software at hand; rather, it's about verifying the core functionality. In addition to testing the product against business requirements, security test cases pertaining to end users need to be tested during this phase.

The tests cover realistic, functional scenarios including authentication and authorization tests. User acceptance testers act as end users, mimicking these real-life scenarios. Security issues they encounter in UAT are also possible in production.

Pitfalls and gotchas

One of the most common pitfalls in user acceptance testing is not testing for security issues. Common pitfalls do include:

- **Staying on the path.** The UAT process follows well-defined scripts that cover the essential "happy path." Following these scripts mimics realistic scenarios of proper usage, but may not deviate from the path in creative ways an attacker might consider. A great deal of testing literature discusses the difference between "checking" and "testing," and a common failure in UAT is to perform too much checking and not enough exploratory testing.
- **Not mimicking the production environment.** The UAT environment must mimic the production environment, and often it does. Dummy data, test connections, and other artifacts of the UAT environment make some security testing less accurate.
- **Technical team members performing UAT.** End users may or may not have technical expertise, but they are extremely proficient in the business requirements. Thus, having technical team members perform UAT testing will not result in real life test scenarios.
- **Depending on automated test scenarios.** Testers often miss certain scenarios if UAT is 100% automated.

The right way to do it.

A UAT plan provides thorough and realistic exposure of the software to all normal usage scenarios and unexpected events. The end users testing the system are well-versed in the business requirements of the software at hand. They must be able to validate the end-to-end business flow of the product, while being mindful of security considerations.

- **Test for unusual conditions.** Testing unusual business conditions ensures that no security bugs exist that could hamper the business logic of the product. For example, if business logic dictates that a user shouldn't be able to shop unless there is credit in their account, then the UAT test plan should include tests attempting to shop with a zero balance. Testers can then confirm or deny that the system disallows shopping with a zero or a negative balance, and that the error screen displays a defined error message.
- **Tour the software.** In Exploratory Software Testing[1], James Whittaker describes ways to "tour" software while testing. These tours include "The All-Nighter Tour," "The Money Tour," and "The Back-Alley Tour," focusing on specific ways to interact with an application. Tours help testers to focus on leaving the happy path throughout the testing process and often yield important security findings that need to be addressed.

> The UAT environment is the earliest available environment where the product can be tested using IAST.

- **Implement DAST during UAT.** In an Agile life cycle, DAST can be focused on the updated or newly developed UI modules, but it is recommended to test the entire UI of the application. Often, logic changes in one module also project into unrelated UI modules. Since the UAT environment mimics the production environment, DAST discovers runtime and environment issues in the UAT phase of testing.

- **Don't forget Interactive Application Security Testing (IAST).** The UAT environment is the earliest available environment where the product can be tested using IAST. This testing method combines elements of SAST and DAST simultaneously to produce better results than each on its own. It is typically implemented as an agent within the runtime environment that observes attacks and identifies vulnerabilities. As a hybrid approach, IAST works like a debugger would—it sees input and output along with the logic and data flows and performs behavioral analysis.

  Performing IAST requires little human involvement and is extremely fast. This process is also framework—and language—independent and does not require the source code to perform testing. Since it monitors the application during runtime to identify vulnerabilities, it has a low false positive ratio and the findings are often accurate and easily verifiable. It can provide a less expensive alternative to traditional DAST scanning for small businesses without a budget for comprehensive security programs. An emerging technology, IAST requires more research to achieve the same level of maturity that SAST and DAST have achieved.

6. DevOps and deployment

Organizations aim to respond quickly to market changes by shipping new software and updating existing software. DevOps integrates development and operations activities to improve productivity and collaboration. Repetitive tasks are automated like testing and infrastructure provisioning. In doing so, DevOps enables organizations to:

- Embrace change
- Develop, test, and deploy quickly and securely
- Fail fast, understand customer feedback, and gather metrics
- Deploy consistently with repeatable and predictable results

What are you trying to achieve?

Security shouldn't hinder DevOps adoption. Instead, security should support DevOps teams delivering reliable software quickly and securely. The close relationship between developers and operational roles reduces the number of human errors—whether it's an implementation bug, a design flaw, a missed security patch, or infrastructure misconfiguration leading to security vulnerabilities.

- **Deploying some tools and declaring a DevOps victory.** Automation is a hallmark of success in DevOps, but the presence of tools like Jenkins and Chef doesn't make an organization successful in DevOps. DevOps aims for end-to-end collaboration throughout the life cycle. It reduces manual errors and saves time for repetitive deployments. Automation also enforces consistent security-related configurations and deployment choices. It is not enough, however, to deploy some tools and declare a DevOps victory.
- **Having developers handle operational tasks.** DevOps embraces collaboration between operations and development teams—not development teams handing operational tasks themselves. The operations team members are subject matter experts who automate the continuous deployment process and connect it to the developers' workflow so that developers can consistently perform a successful release after a successful build.

The right way to do it

Incorporating security into DevOps activities maintains the speed of deployment and delivery while ensuring security requirements and configurations are maintained. To ensure success, remember that:

- **The build fails if security tests don't pass.** Choose the appropriate security tool (SAST or DAST) and integrate it into the build pipeline using continuous integration tools like Jenkins. A mature organization fails the build if the security tests don't pass. That is, if security issues appear in the build (perhaps issues higher than medium severity) the build fails and the development team is notified. Thus, security is treated with the same level of importance as business requirements.

  In some cases, the security tools are too unreliable and too noisy to allow them to fail the build. Let's say, for example, that there is legacy code that produces many security warnings, or perhaps the tool is just generating too many false positives. The "security fails the build" principle is not all-or-nothing. Some modules, for which security tools work reliably can be set to fail on security issues, while other modules can be set to deploy despite security warnings. It is important to keep a scoreboard, dashboard, or other dynamic record of outstanding security warnings. Incentivize teams to make their modules fail the build on security issues. Greenfield development should default to include security tools in the build pipeline, and they should default to allow security issues to break the build.
- **Security should be integrated into continuous delivery.** Some organizations take continuous deployment a step further into continuous delivery. Code that passes all the tests and builds properly is automatically promoted and deployed to production. In such an environment, it is vital to render all security controls and configurations into some format that is automatically deployed with the software. This means service account user IDs, filesystem permissions, ACLs, roles, credentials, and all the security controls that are required for the software to run correctly.

  Integrating security into continuous delivery also means expressing the software's own security controls in an automatically-deployable format. For example, if the software allows tax advisors to access some client tax records, but not others, continuous delivery will have to deploy user accounts, account roles, and changes to the security framework as part of the software deployment itself. This is a tall order that is suitable only for the most mature DevOps teams.

7. Post-deployment and live testing

Periodically testing applications once they're deployed is necessary to identify security issues that are a product of code, production, and environment configuration. Production environments sometimes differ from QA or staging environments, especially in the data they host. Testing production regularly ensures that security controls implemented in the software or underlying platform work in production.

What are you trying to achieve?

The goal of testing live software is to augment security testing that was carried out in pre-production environments. In doing so, you're ensuring that the security behavior of the software remains consistent with what was observed in development. You'll want to see the interaction between the security controls in the live environment and the code.

It's also important to consider that the threat landscape is constantly changing. Because of this, it's critical to make sure that the security tools used in security testing are up-to-date. If you're using up-to-date tools and techniques to test the production environment, you can identify security issues in platforms and application frameworks that you couldn't find during development.

Pitfalls and gotchas

During post-deployment and live testing, you'll want to avoid:

- **Using live testing as the single stage of security testing.** Some organizations defer security testing until long after a piece of software is deployed into production. If the software hasn't been tested in previous stages of development, testing in production won't cover enough ground. Post-deployment testing typically provides breadth, but not depth. It may not discover defects arising from issues in the software's architecture and business logic.
- **Ignoring terms and conditions.** Some applications are deployed on third-party infrastructure-as-a-service (IaaS) or platform-as-a-service (PaaS) options like Amazon Web Services (AWS) or Microsoft Azure. The platform may have terms and conditions that govern security testing activities. Most platform providers allow for application security testing and network security testing if they are notified in advance. The organization deploying the software onto

the platform (not the security testing organization) is responsible for adhering to any terms and conditions relating to security testing. Also, be careful that the third-party doesn't quarantine or shut down instances that are behaving strangely due to security testing.

- **Keeping teams in separate silos.** Testing in a production environment requires close coordination between testers and production support teams. Testers need access to the production environment and should ensure that any high-risk issues are reported as they are discovered. Operational teams can then plan to set up interim workarounds until issues are remediated. Everyone who deals with normal failures in production should be notified that security testing is occurring. This can include the network operations center (NOC), security operations center (SOC), customer support, or other relevant departments within the organization.

The right way to do it

Ideally, testing post-deployment provides an ongoing level of assurance that changes to the production environment haven't introduced security issues. A good approach to this is to have a process that periodically triggers security testing. This includes:

- **Scheduled security scanning.** Scanning pieces of software and applications from time to time ensures that any newly discovered security issues in frameworks are identified. Additionally, a periodic scan schedule can help catch bugs that may have slipped through pre-production testing activities.
- **Vulnerability scanning.** Scanning the planning infrastructure (e.g., web servers or application servers) ensures that all modules have been patched to protect against newly discovered threats. Vulnerability scanning can also uncover configuration mistakes. For third-party managed platforms, running vulnerability scans may require permission from the platform support team.
- **A bug bounty program.** Some organizations choose to run a bug bounty program in which outside security researchers are rewarded for discovering and reporting issues. This requires having a mature SDLC and the bandwidth to triage and investigate issues reported by users. It's important to emphasize that a bug bounty program isn't a replacement for other production security testing. It's a means of receiving assistance from the outside, but that outside assistance cannot replace security activities conducted by the development team itself.
- **An incident response plan.** Having a well-defined incident response program is critical to ensure that teams know what needs to be done when a security incident occurs. An incident response process may involve notifying users or taking key systems down for investigation while the incident is being resolved. Depending on the type of industry, organizations may be required to inform law enforcement or regulatory bodies when an incident occurs; this should be accounted for in the incident response plan.

  In many organizations, incident response includes rehearsals in which the business, support, operations, and development teams practice responding to potential incident scenarios. They collect information about running systems, make sure they can communicate with each other effectively, and follow emergency procedures. Like fire drills, this exercise helps technical and non-technical staff know what to do in an actual emergency.
- **A threat intelligence program.** A number of vendors offer services that give insight into security issues affecting popular frameworks and deployment stacks. Operations teams can use insight obtained from this threat

intelligence to drive testing and patching of production resources. Threat intelligence is most effective for mature organizations who have a strong understanding of their application portfolio and environments. A threat intelligence program can help teams stay ahead of the curve in proactively responding to newly discovered security issues affecting applications and platforms.

- Continuous monitoring. Many applications have built-in tooling for performance monitoring in production. This allows an organization to gain insight into the types of traffic a given application is receiving. Additionally, collecting application-level security metrics helps identify patterns of malicious users. For example, logging failed authentication attempts by IP address helps applications identify brute-force attempts to gain access to user accounts.

  Based on this data, network controls can be implemented to rate, limit, or block offending IP addresses. Alternatively, applications can automatically trigger rate-limiting controls. The goal is almost always to collect data that are actionable and available to the people or systems that can take preventative actions.

Now that you know which activities are best suited for the various SDLC phases, how can your firm get started?

If security is only an element of someones's job description, odds are that security,

WON'T BE MUCH OF A PRIORITY

within your organization. However making security someone's job promotes a successful security initiative.

Ready. Set. GO!

3 Steps to Get You Started

1. Put someone in charge

If security is an element of someone's job description, odds are that security won't be much of a priority within your organization. Making security someone's job promotes a successful security initiative. One area to focus on when hiring someone to head up your security program is competency management. Ensure that the person tasked with leading your security initiative knows exactly what they need to know to be successful in performing those activities.

2. Take advantage of the expertise of professionals

A straightforward way to get security started in your firm is to hire security experts to join your staff. Easy, right? After all, internal security people will come to know your portfolio. They can work with various internal teams to learn stakeholder values and expectations. One option is to utilize individuals who have grown within your organization and have obtained security expertise through certifications and continuing education.

Companies often look for a skill range including malware, threat mitigation, cryptography, forensics, industry-specific knowledge, cloud and mobile security, advanced analytics, and network virtualization. But, that's a lot to ask from a small team, or in the case of many organizations, a single expert. Utilizing professional services is an effective way to:

- **Build a new security initiative.** If your firm is new to security, an external security guru, or team of gurus, can provide expertise and a fresh perspective. It's hard to set a strategy for application security if you have never done it before. It is wise to bring in a professional services expert who can help you see your problem through fresh eyes and offer perspective based on what has and hasn't worked for organizations like yours. They have insight into how to approach strategic problems and can help build a program or an actionable roadmap for your organization. This allows you to make better use of your internal resources and tools to increase your breadth, depth, and cost effectiveness.

- **Support an existing security initiative.** It is also common to bring in professional services resources to fill in capability gaps or help meet capacity needs in an existing security initiative. Professional services teams are cross-trained with diverse skills that are up-to-date, and offer fresh perspectives to your current operations. If your needs required it, a professional services firm would allow you to engage with a Java expert on Monday, a COBOL expert on Wednesday, and a Natural expert on Friday.
- **Train your internal team.** Working with professional services can also boost the skillsets of your internal team members—learning by example. Elements that make conducting security activities yourself difficult are the same things that external professional services resources can make easy.

3. Multiply your capabilities with Managed Services

Scaling your efforts can take a few different forms. First, you need to know why you are scaling. Is the goal to expand your breadth to cover a larger percentage of your portfolio? Is the goal to shorten the project timeline? Is there a need to increase scale drastically and then reduce it in bursts to adapt to change? Is there a need to increase cost effectiveness without sacrificing quality? Some of these reasons (among others) motivate organizations to look at consuming security services at scale, rather than professional services or individual engagements.

Beyond the ability to accommodate several types of engagements consistently at scale, a managed services partner offers the flexibility to tackle the unknown factors such as the number of applications, frequency of testing, and even depth or type of assessment. For example, it isn't practical to hire a team of penetration testers to get through 500 applications and then repurpose them when the project is complete. Utilizing managed services is a cost-effective way to:

- **Create consistency at scale.** A managed services partner allows for this consistency at scale with offerings that can cost fractions, and in many deployment models, with your partner absorbing the cost risk instead of you. Full-time staff cost money whether there is testing available for them to do or not. In a managed service model, there is a flat fee associated with each test. Costs are predictable and tied directly to your portfolio. Beyond scale and flexibility, using a partner can reduce the need for tool costs, tool deployments, and internal infrastructure. Many also offer a dashboard to gather metrics, statistics, and visibility into your entire portfolio.
- **Increase internal bandwidth.** While it's wise to keep a small team of security experts in-house, another advantage of outsourcing is the additional bandwidth it provides to do more testing. More testing means increased portfolio coverage. If conducted with a managed services partner, this allows you to more effectively utilize your budget while increasing internal bandwidth. For example, a managed services team can take care of the tactical testing elements of your firm's software security initiative. Meanwhile, your in-house team ensures that strategy is properly executed, with an added benefit of allowing them to work more closely with your development teams.
- **Drive policy and strategy decisions with data.** Once outsourced testing is up and running, the in-house team can continually examine the vulnerabilities found and evolve your program accordingly. Expanded application security testing allows your in-house team to effectively deal with vulnerability management by monitoring results and prioritization on your terms.

Get Rolling:

What are you waiting for?

When implementing security into the various phases of the SDLC, it's important to implement these activities with purpose. Beyond fielding tactical situations and challenges, ask yourself where each activity fits into the overall program. If you don't have a program yet, reach out to a partner that can guide you through your secure software development journey in a way that fits your organization and its objectives.

While bringing in one or more tools can be a beneficial approach to scaling security activities, simply buying a tool and placing it in your environment will get about as far as a train without tracks. Plugging activities into other activities creates capabilities. Capabilities can do more together than apart. Trust in your own capabilities where you can, and supplement with trustworthy guides to fill in the gaps.

Ready to build security into your SDLC?

START HERE

THANK YOU

Thank you to our contributors:

Gopal Addada

Neil Bahadur

Kevin Glavin

Paco Hope

David Johansson

Michael Lyman

Apoorva Phadke

Ritesh Sinha

Brian Sowers

Meera Subbarao

Vinay Vishwanatha

SYNOPSYS®

The Synopsys difference

Synopsys helps development teams build secure, high-quality software, minimizing risks while maximizing speed and productivity. Synopsys, a recognized leader in application security, provides static analysis, software composition analysis, and dynamic analysis solutions that enable teams to quickly find and fix vulnerabilities and defects in proprietary code, open source components, and application behavior. With a combination of industry-leading tools, services, and expertise, only Synopsys helps organizations optimize security and quality in DevSecOps and throughout the software development life cycle.

For more information, go to www.synopsys.com/software.

Synopsys, Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

Contact us:
U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com