Improving RoI by Using an SDL

This paper discusses how you can improve return on investment (RoI) by implementing a secure development lifecycle (SDL). It starts with a brief introduction to SDLs then explains how implementing an SDL can save your organization money, and concludes with a discussion of how threat modeling and penetration testing complement SDLs.

Introducing SDLs

SDLs integrate security into all phases of the software lifecycle, and they were envisioned once software development organizations realized that trying to bolt security onto software after it had been developed produced an insufficiently secure product. The first SDL was created as a way to integrate security with software from inception through maintenance.

Microsoft pioneered the most currently mature and widely utilized secure development methodology in January 2002 when Bill Gates released a memo to all Microsoft employees known as the “Trustworthy Computing” letter. With this letter, Microsoft committed itself to fundamentally changing its mission and strategy in the key areas of security, privacy, reliability, and business practices. This revolutionized Microsoft's approach to security, changing it from a reactive to a proactive process.

The Microsoft SDL is a 14-stage, waterfall-oriented process that begins with education and awareness prior to a project's inception and ends with post-release response planning; refer to Figure 1.

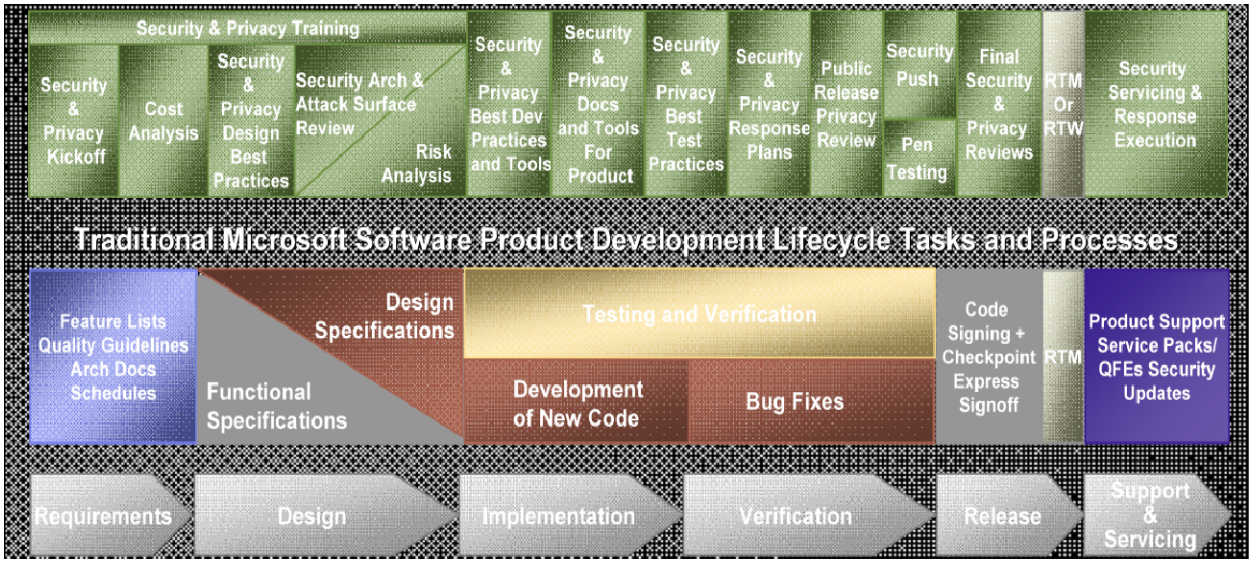

*Figure 1: Waterfall SDL**[1]

The SDL that your organization implements may vary from the Microsoft SDL since it will be based on your internal development methodologies, which could include Agile, Spiral, or

[1] *This graphic is copyrighted by the Microsoft Corporation.

Prototyping, to name a few. For example, Figure 2 shows an SDL implemented for an Agile development methodology.

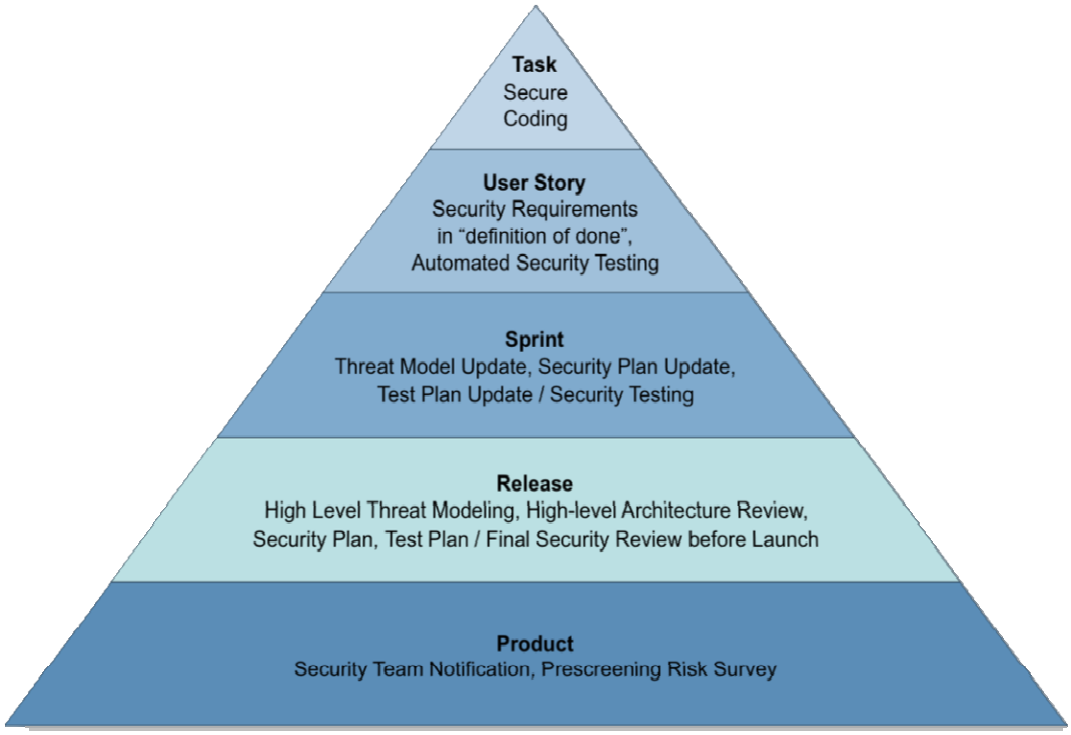

Figure 2: Agile SDL

Security becomes an iterative effort, mirroring the Agile development process, so each sprint should produce software that not only can be demoed, but can be demoed and proven secure. It also is common to add one or more sprints that allow for performance of dedicated security tasks, such as the Security Push and Final Security Review. The point is to build security into your development efforts, not try and add them after the fact.

Ignoring Security and the Cost to RoI

It is absolutely true that the initial deployment of an SDL may be disruptive to your current development processes, and that the initial cost of software development may increase with the added rigor of an SDL. However, research shows that implementing an SDL early in a development project saves money overall; in fact, Microsoft estimates it has saved millions because of their SDL. Just remember that waiting to fix a flaw until later—when you're further into the development lifecycle—will be vastly more expensive than fixing it earlier.

A study performed by the IBM Systems Sciences Institute, as cited by Kevin SooHoo, Andrew W. Sudbury, and Andrew R. Jaquith in "Tangible RoI through Secure Software Engineering" in the Secure Business Quarterly, Volume 1, Issue 2 illustrates the cost multipliers of discovering and fixing security flaws later in the software lifecycle. As shown in Figure 3, catching design flaws at design time is nearly free—it may cost a few hours for the design personnel, but little to no development, test, or support time has been spent. The later in the lifecycle, the more expense.

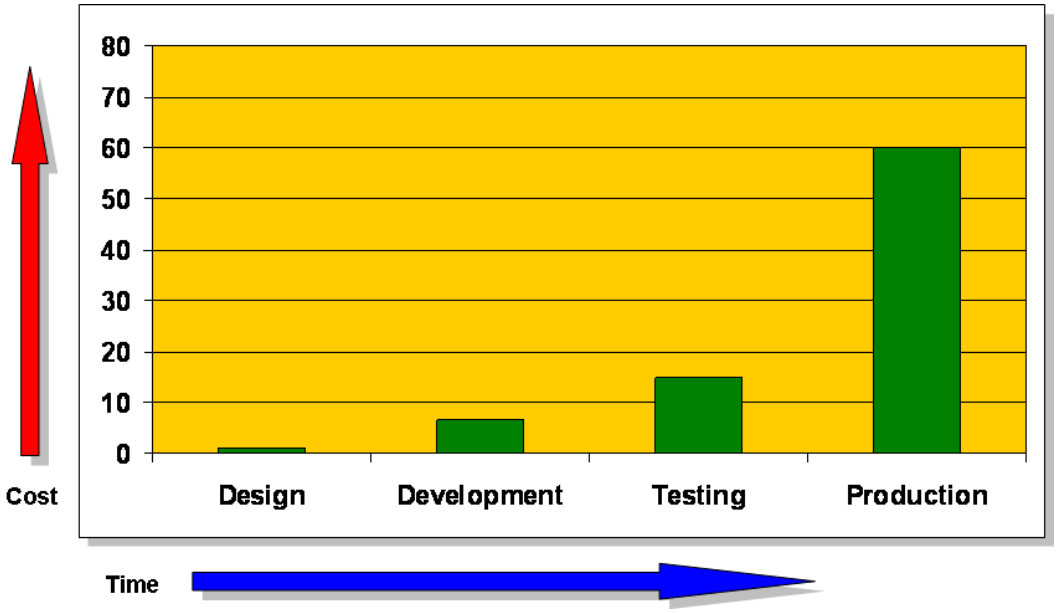

Figure 3: The cost of security

A 2002 study illustrates this point perfectly: the U.S. Department of Commerce National Institute of Standards and Technology concluded that software errors (or *bugs*) cost the U.S. economy $59 billion annually, or about 0.6% of the gross domestic product. Microsoft estimates that their average cost to fix a bug after product release is $250,000. Of course, your organization's costs likely will be much lower than Microsoft's, but it is worth considering how much effort is required to regress and compatibility test every change.

It is important to note that the damage caused by software bugs is not exclusively financial, and that these errors also can cost human life.

> In 1994, an RAF Chinook helicopter (serial number ZD576) crashed in Scotland, killing all 29 passengers, and one of the possible causes was the FADEC engine control's upgraded software, which reportedly exhibited 486 anomalies in only 18% of its complete codebase.
>
> Between 1985–1987, the Therac-25 radiation therapy machine caused at least six accidents where patients were given massive overdoses of radiation—three patients died as a direct result. The machine's software contained a flaw that is now recognized as a race condition.

All this goes to show that paying attention to security sooner rather than later is going to improve your return on investment. In fact, Jeffery Jones, Security Strategy Director in Microsoft's Trustworthy Computing group published the "Windows Vista One-Year Vulnerability Report" and noted that as a result of its implementing an SDL, there was a marked decrease in the total number of vulnerabilities that required patching between the first year of Windows Vista and Windows XP releases.

Table 1: Windows Vista versus Windows XP

Metric	*Windows Vista (year 1)*	*Windows XP (year 1)*
Vulnerabilities fixed	36	65
Security updates	17	30
Patch events	9	26
Weeks with at least 1 patch event	9	25

But it's not just development costs associated with defects and vulnerabilities that you need to worry about—an organization must consider compliance costs as well. The primary cause-and-effect relationship between an SDL and compliance is in the provability of your security stance and efforts. A well-documented development process can aid compliance with:

> Sarbanes-Oxley, data protection and breach notification laws, corporate policies, contractual commitments, service level agreements, business partner audits, customer audits and SAS 70, external/internal auditors and regulators, and PCI assessors.

A note about PCI compliance: organizations that lack an SDL have a much harder time proving compliance with the following Payment Card Industry Data Security Standards:

- Requirement 6: Develop and maintain secure systems and applications.
- Requirement 11: Regularly test security systems and processes.

Threat Modeling

Threat modeling is a method of determining, enumerating, measuring, and documenting risk presented to or by an application or system, which means that development teams can then systematically and methodically decompose and analyze potential issues and how much harm they could inflict. This allows the organization to make informed decisions about how to best control risks and determine cost effective mitigation strategies.

Because threat models can help production teams identify design issues before a product is implemented, they can better determine the impact and resultant cost of implementing that product with and without applying a solution. Being able to fully view a problem through threat modeling also can provide insight onto solutions the team either would not have considered or didn't know were available, and can powerfully drive future design and/or implementation decisions.

So, when should you create a threat model? Early and often. Threat modeling should be built into a product's design phase and performed regularly afterward so that new features and functionality are accounted for when considering risk. Previous and existing threat models provide an excellent starting point as these build on the organization's collective wisdom and can offer important insight. The length required to build a threat model depends on your team's experience with writing them, the availability of documentation and

subject matter experts, and the target product/system's complexity. However, rolling the process into design and documentation work can alleviate much of the potential resource burden.

Penetration Testing

The purpose of a penetration test is to assess the level of security posture evident in an application or system's design, and one way to do that is by attacking, modifying, and hijacking its client/server applications to discover any high-level vulnerabilities. By examining and attempting to exploit security flaws, penetration testing can uncover risks such as privilege escalation, disclosure of sensitive information, injection of malicious code into trusted components, invalid transactions, and other conditions generally recognized as posing security vulnerabilities.

Application-oriented penetration testing illuminates attack methods that are unique to a given organization and demonstrate the potential impact of a real-world attack. Adding a review of application source code and configuration files can uncover attack risks from the most common and dangerous source: malicious developers.

Threat models can help focus penetration tests and code review efforts on the areas thought to represent the highest impact.

So, what are the different types of penetration test?

Black box. Conducted from the perspective of an attacker who possesses minimal or zero knowledge of the application or system.

White box. Conducted from the perspective of an attacker who possesses full knowledge of the application or system; that is, they have access to source code.

Gray box. A combination of black- and white-box tests.

The next logical question posed by many organizations has to do with automated security tools and, if they're being used, why endure the resource expenditure to implement a manual evaluation? Automated tools provide a great starting point and can often guide review efforts, but they simply aren't sensitive enough to catch every issue; they overlook complex vulnerabilities frequently and do not handle code loops or architecture branches well.

When should an organization perform code reviews or penetration testing? Code reviews should occur incrementally as developmental milestones are met; automated tool scans can be performed whenever new code is checked in, and a formal review should be conducted when the code is deemed complete. Incremental penetration tests should occur at all major developmental milestones with a full, detailed review when code is deemed complete.

The benefits of security testing:

- Threat modeling uncovers design issues early.

- Code review uncovers implementation errors.
- Penetration testing identifies overall system weaknesses.
- Tools help gather data and perform automated, brute-force tests.

Conclusion

Implementing a secure development lifecycle will save your organization money in the long run because less money is spent on fixing bugs or issues and you realize a greater return on development efforts. To recap:

- Bugs are expensive in the later stages of a software lifecycle.
- Change requests can be incredibly disruptive to development efforts.
- Finding bugs earlier can save massive amounts of money.
- Security efforts can help find bugs earlier in the software lifecycle.
- Building security into your development efforts is more efficient than tacking it on at the end.