Security applies at every phase of the software development life cycle (SDLC) and needs to be at the forefront of your developers’ minds as they implement your software’s requirements.

With dedicated effort, security issues can be addressed in the SDLC pipeline well before deployment to production. This reduces the risk of finding security vulnerabilities in your app and works to minimize the impact when they are found.

Developers now need to be cognizant of potential security concerns at each step of the process. This requires integrating security into your SDLC in ways that were not needed before. As anyone can potentially gain access to your source code, you need to ensure that you are coding with potential vulnerabilities in mind. As such, having a robust and secure SDLC process is critical to ensuring your application is not subject to attacks by hackers and other nefarious users

Implementing SDLC security affects every phase of the software development process. It requires a mindset that is focused on secure delivery, raising issues in the requirements and development phases as they are discovered.

**PHASE 1: REQUIREMENTS**

In this early phase, requirements for new features are collected from various stakeholders. It’s important to identify any security considerations for functional requirements being gathered for the new release.

### ****PHASE 2: DESIGN****

This phase translates in-scope requirements into a plan of what this should look like in the actual application. Here, functional requirements typically describe what should happen, while security requirements usually focus on what shouldn’t.

* **Sample functional design:** page should retrieve the user’s name, email, phone, and address from CUSTOMER\_INFO table in the database and display it on screen.
* **Sample security concern:** we must verify that the user has a valid session token before retrieving information from the database. If absent, the user should be redirected to the login page.

### ****PHASE 3: DEVELOPMENT****

When it’s time to actually implement the design and make it a reality, concerns usually shift to making sure the code well-written from the security perspective. There are usually established secure coding guidelines as well as code reviews that double-check that these guidelines have been followed correctly. These code reviews can be either manual or automated using technologies such as [**static application security testing (SAST)**](https://snyk.io/learn/application-security/static-application-security-testing/).

That said, modern application developers can’t be concerned only with the code they write, because the vast majority of modern applications aren’t written from scratch. Instead, developers rely on existing functionality, usually provided by free open source components to deliver new features and therefore value to the organization as quickly as possible. In fact, 90%+ of modern deployed applications are made of these open-source components. These open-source components are usually checked using [**Software Composition**](https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/)[**Analysis**](https://snyk.io/series/open-source-security/software-composition-analysis-sca/)[**(SCA)**](https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/) tools.

#### **Secure coding guidelines, in this case, may include:**

* Using parameterized, read-only SQL queries to read data from the database and minimize chances that anyone can ever commandeer these queries for nefarious purposes
* Validating user inputs before processing data contained in them
* Sanitizing any data that’s being sent back out to the user from the database
* Checking open source libraries for vulnerabilities before using them

Using secure coding guidelines like…..

### ****PHASE 4: VERIFICATION****

The **Verification**phase is where applications go through a thorough testing cycle to ensure they meet the original design & requirements. This is also a great place to introduce automated security testing using a variety of technologies. The application is not deployed unless these tests pass. This phase often includes automated tools like CI/CD pipelines to control verification and release.

#### **Verification at this phase may include:**

* Automated tests that express the critical paths of your application
* Automated execution of application unit tests that verify the correctness of the underlying application
* Automated deployment tools that dynamically swap in application secrets to be used in a production environment

### ****PHASE 5: MAINTENANCE AND EVOLUTION****

The story doesn’t end once the application is released. In fact, vulnerabilities that slipped through the cracks may be found in the application long after it’s been released. These vulnerabilities may be in the code developers wrote, but are increasingly found in the underlying open-source components that comprise an application. This leads to an increase in the number of “zero-days”—previously unknown vulnerabilities that are discovered in production by the application’s maintainers.

These vulnerabilities then need to be patched by the development team, a process that may in some cases require significant rewrites of application functionality. Vulnerabilities at this stage may also come from other sources, such as external penetration tests conducted by ethical hackers or submissions from the public through what’s known as “bug bounty” programs. Addressing these types of production issues must be planned for and accommodated in future releases.

How to ensure SSDLC

## **5 Secure SDLC Best Practices**

### 1. EDUCATE YOUR DEVELOPERS

Secure SDLC goes hand in hand with multiple related initiatives, including:

* Creating secure coding guidelines
* Providing developers with security awareness and [**secure coding**](https://snyk.io/learn/secure-coding-standards/) training
* Setting clear expectations around how quickly issues discovered in production need to be addressed (also known as remediation SLAs).

### 2. HAVE CLEAR REQUIREMENTS

Whatever you create, it should be easy to understand. Development teams need clear requirements that are easy to act upon. This applies to all security advice, recommendations, and guidelines. Any vulnerabilities discovered in tests need to be easy to act on. It’s key that all people, processes, and tools involved bring solutions to the table instead of just pointing out problems

### 3. MAINTAIN A GROWTH MINDSET

Since SSDLC will change how multiple teams work and interact, it’s important for everyone to go into this experience with an open mind, and for the security team to have the mindset of empowering developers to secure their own applications

### 4. TIE IMPLEMENTATION TO OTHER INITIATIVES

For well-established applications and teams, it may often be easier to implement SSDLC changes when it’s tied to another modernization effort, such as a cloud transformation, a DevOps initiative, or its more security-conscious variation, **[DevSecOps](https://snyk.io/series/devsecops/)**.

### 5. TACKLE THE BIG PROBLEMS FIRST

Focus on the most important issues and actionable fixes rather than addressing every vulnerability found. While it may be possible for newer or smaller applications to fix every security issue that exists, this won’t necessarily work in older and larger applications. A triage approach can also be helpful. This focuses on not only preventing security issues from making it into production, but also ensuring existing vulnerabilities are triaged and addressed over time.

///new

* Providing secure coding training to developers, to ensure security from the start(implementation) i.e applying the develpers who are expert in securing the system
* Automating and integrating security tests to detect security risks as close to the point of remediation as possible(testing)
* Securing open source components and libraries present within projects(design)

#### Requirements and Analysis

This phase involves deciding which frameworks, languages, and technologies should be used. It is important to determine which vulnerabilities or insecure coding practices may be particularly relevant to the selected resources.

For example, certain frameworks may lack security competencies for your specific environment, or some technologies may be incompatible with security tools already in use elsewhere in your organizations. Failure to consider the full breadth of implications here can potentially threaten the security of all technologies chosen during this phase and those which are incorporated at later stages.

#### Design and Prototyping

The design phase involves using established patterns of application architecture and software development. For example, software architects may decide to leverage an architecture framework that enables the use of existing components and promotes standardization.

Proven design patterns help developers solve algorithmic problems in a consistent manner. Additionally, this phase may include rapid prototyping (or spike), which helps compare technologies and find the most suitable solution to achieve the requirements identified in the earlier phase.

Using good and effective design patterns are very helpful

#### Development and Testing

It is critical to include secure coding standards during the development phase, as well as encouraging selection of secure open source and third-party components being brought into the project. This typically includes a code review process that helps ensure the project has met the required features and functions, as well as various testing that identifies weaknesses in custom code, known open source vulnerabilities

#### Deployment

-private the environment in which the system is deployed

In keeping with DevOps and cloud native software methodologies, the deployment phase should be automated as much as possible. High-maturity enterprises often implement this phase in a manner that deploys software as soon as it is ready, at the end of a dedicated sprint or development cycle. This approach should not be employed, however, unless security tools and practices can accommodate this velocity and block potential security risks from being deployed into production environments.

Enterprises with lower DevOps maturity, or those operating within highly regulated industries, may require manual review and approval prior to deployment, particularly for business-critical applications or those handling sensitive data.

#### Maintenance

Even after performing extremely thorough testing processes, newly published vulnerabilities may impact applications that have been pushed into production. Additionally, an application may behave differently at runtime in production environments than it does in a static state or in development environments. This is why security efforts should not stop once your application is released. Security is a continuous cycle that should be maintained on a regular basis.

Your maintenance phase begins immediately after the deployment phase, and should ensure a path of direct feedback and communication between security and development teams. Preparations should be made for accelerated issue management and risk remediation to reduce the window of opportunity for an attack on production assets.

Operations or DevOps teams should also ensure proper security configuration of cloud environments and resources associated with application functionality, such as container engines and orchestration tools. Regularly perform these security checks against the software and the environments, update them regularly to meet evolving requirements, and ensure compatibility with any new tools being used elsewhere in the secure SDLC.

* Upgrading the developed software
* Hiring the ethical hacker to perform penetration testing
* perform penetration testing within scheduled time

## **How do you get started?**

If you’re a developer or tester, here are some things you can do to move toward a secure SDLC and improve the security of your organization.

* Educate yourself and coworkers on the best secure coding practices and available frameworks for security.
* Conduct an [architecture risk analysis](https://www.synopsys.com/software-integrity/software-security-services/software-architecture-design/risk-analysis.html?intcmp=sig-blog-sdlc) at the start.
* Consider security when planning and building for test cases.
* Use code scanning tools for [static analysis](https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html?intcmp=sig-blog-sdlc), [dynamic analysis](https://www.synopsys.com/software-integrity/security-testing/web-scanner.html?intcmp=sig-blog-sdlc), and [interactive application security testing](https://www.synopsys.com/software-integrity/security-testing/interactive-application-security-testing.html?intcmp=sig-blog-sdlc).

11 best practices

### 1. Specify your requirements

Your specifications, security guidelines, and recommendations should be presented in a way that is simple and easy to understand in order to assist your developers.

### 2. Perform security audits

Security testing is critical for determining your product's vulnerability to attacks. Ensure security tools and practices are in place from the outset and throughout the development process. It is also important to clearly define the functional requirements of your development teams, take into consideration common security vulnerabilities, and plan accordingly.

### 3. Educate your developers on best coding practices, tools, and frameworks

Organizing training sessions for your development team can help foster a culture of security awareness. These sessions should cover areas such as secure coding practices, cybersecurity, potential risks, and the available security frameworks.

### 4. Conduct an architectural risk analysis at the beginning

Conducting an architectural risk analysis to identify flaws and determine risks that might occur due to those flaws should be done early on in the SDLC process, before coding your application. You should also take advantage of threat modeling to detect and manage threats.

### 5. Tackle the big problems first

Instead of addressing every vulnerability identified, it is important to prioritize the risks based on your business needs and to concentrate on the most serious threats and feasible remedies. A triage approach—find, prioritize, and fix—can help you to focus on avoiding security risks from entering production and triaging and addressing existing vulnerabilities over time.

### 6. Secure planning and building for test cases

Conducting a code review is important for verifying whether your development team has adhered to secure coding standards, and allows you to uncover coding and configuration defects or weaknesses in the application. Make your plans ready for penetration testing on your application, and ensure that it is conducted by a third party. Organizations often employ third-party vendors to perform penetration testing. The primary objective of having a third-party vendor assess the security of your systems is to get an impartial, professional, and expert opinion on your security posture.

### 7. Use code scanning tools

There are two types of analysis tools: static analysis security tools (SAST) and dynamic analysis security tools (DAST). While SAST enables you to analyze your code to identify security flaws in the application without running it, DAST is capable of finding flaws in your infrastructure.

### 8. Cultivate a growth mindset

Secure SDLC changes how teams operate and communicate. Everyone on the team should be open to learning, and your developers should be encouraged to adhere to the guidelines and best practices to secure the applications they build.

### 9. Keep an eye on open-source security

Open-source components with known vulnerabilities are another important factor to take into consideration when building a secure SDLC. Because today's software products rely heavily on open-source code, it is critical to focus on open-source security management throughout the SDLC process.

Automated software composition analysis (SCA) tools can help determine security vulnerabilities in code and provide remediation insights and automatic patches.

### 10. Perform a gap analysis

A security gap analysis is a great way to check the integrity of your application. Performing a gap analysis will help you assess how effectively your system is operating based on your expectations. If there is any deviation from these expectations, you can identify which part of your SDLC needs to be re-examined in order to make the necessary improvements.

### 11. Create a software security initiative (SSI)

A software security initiative (SSI) is a process that allows you to plan for risk and allocate resources accordingly. An SSI guides you through the SDLC based on procedures and guidelines, and helps you to determine how much you should spend on application security. It also helps to ensure your team really understands their roles.

////BEST

While these elements are relevant to all phases of the SDLC, here are 10 phase-specific ways to infuse even more security into your software development life cycle:

## **1. Assess the landscape**

**SDLC phase: Requirements gathering**

Begin the cycle with a strong understanding of what the customer actually wants. Here’s how to make that happen:

* Establish the scope and boundaries
* Identify stakeholders
* Identify process gaps
* Institute tailored security-centric processes scaled to the organization and project scope

## **2. Incorporate an industry-standard security model**

**SDLC phase: Requirements gathering**

Secure the software you’re building from the beginning. This is the most cost-effective way to minimize the ‘test-patch-retest’ cycle that often negatively affects budget and scheduling goals near the end of the life cycle.

Integrate a trusted maturity model into your SDLC to infuse best practices and solid security design principles into the organization. The [Building Security In Maturity Model (BSIMM)](https://www.synopsys.com/software-integrity/software-security-strategy/bsimm-maturity-model.html) acts as a measuring stick that pinpoints strengths and weaknesses in your current security initiative. A BSIMM assessment can help your firm create data-driven goals.

## **3. Educate personnel on software security**

**SDLC phase: Requirements gathering**

Ensure that all personnel involved in the project are knowledgeable and up-to-date with software security standards to reduce insecure design and development practices. [Investing in training your staff](https://www.synopsys.com/software-integrity/training.html) is scalable, and aligns with the overall organization and the scope of each software development project at hand. The benefits resulting in a [well-trained staff](https://www.synopsys.com/blogs/software-security/can-you-afford-not-to-implement-security-training/) span all software development projects and can be an enterprise-wide asset.

## **4. Assign responsibility of software security**

**SDLC phase: Requirements gathering**

To ensure that software security is incorporated into the SDLC, formally assign responsibility for it. Depending on the size of your organization, creating a [software security group (SSG)](https://www.synopsys.com/blogs/software-security/software-security-group-ssg/) is an effective way to educate, assess, and enforce established security measures across the organization. This is key to maintaining change and risk management as your organization scales up, without degrading or ignoring security all together.

The SSG should act as the subject matter experts in software security, facilitating and conducting [third-party security assessments](https://www.synopsys.com/software-integrity/solutions/by-security-need/vendor-analysis.html) during critical stages within the SDLC.

## **5. Perform security-focused requirements gathering**

**SDLC phase: Requirements gathering**

Tailor your organization’s approach to generating security requirements as a part of the initial phase. This approach will aid in embedding a solid security mindset throughout the SDLC. Generate [abuse and misuse cases](https://www.synopsys.com/blogs/software-security/abuse-cases-can-drive-security-requirements/) and perform an initial risk analysis during the requirements gathering phase to promote security activities in additional phases within the SDLC. This will also drive focus on testability when generating requirements.

## **6. Establish and institute a comprehensive risk management process**

**SDLC phase: Requirements gathering**

It is critical to your SDLC’s success to identify major risks and execute a mitigation plan. These are also key aspects to:

* Ensure proper security design
* Ensure an effective guide in SDLC execution in terms of:
  + Controlling scope-creep
  + Staying within budget and schedule goals
  + Engaging with stakeholders

## **7. Perform architecture reviews and threat modeling**

**SDLC phase: Design**

It is far more cost-effective to identify and remediate design flaws early in the design process than to patch flawed design implementations once the software is deployed. Along with threat modeling, [architecture risk analysis](https://www.synopsys.com/software-integrity/software-security-services/software-architecture-design/risk-analysis.html) is a critical tool to detect design flaws. Flaws are identified by:

* Analyzing fundamental design principles
* Assessing the attack surface
* Enumerating various threat agents
* Identifying weaknesses and gaps in security controls

## **8. Carry out code reviews during implementation**

**SDLC phase: Implementation**

Along with secure coding standards and static code analyses, perform a [secure code review](https://www.synopsys.com/glossary/what-is-code-review.html) as a condition to passing a release gate. This drastically reduces the number of bugs escaping into the finished product. An effective defect containment and management system also aids in prioritization and tracking defects to resolution.

## **9. Execute test plans and perform penetration tests**

**SDLC phase: Verification**

Execute the test plans during the verification phase. This will verify whether the product performs as expected in runtime scenarios. [Penetration tests](https://www.synopsys.com/software-integrity/security-testing/penetration-testing.html) assess how the product handles various abuse cases, including:

* Malformed input handling
* Business logic flaws
* Authentication/authorization bypass attempts
* Overall security posture

## **10. Deploy software product**

**SDLC phase: Deployment/maintenance**

Generate a deployment plan. This is essential to a successful release to production once thorough QA and acceptance testing are complete. The plan should detail the environment in which the software will operate and the steps for configuration and launch.

Plans for software maintenance and a change management process should be in place at this stage to efficiently handle any bugs or enhancement requests that come out of production.

Rollback plans and disaster recovery requirements in this phase also help ensure continued customer confidence.

//////

Traditional security testing was done on steps (4) and (5) as an activity before and after production software release. For modern software development, it is important to embed security into each of these 5 steps.

## The Requirements Phase

The key concept here is to set the tone and define the security aspects of the upcoming product or product feature. It is an important opportunity to drive a “security first” mindset and promote developer awareness.

For example – a new notification service is being drafted which would notify customers on important events. Security requirements should include, for example, separating tenants such that a notification will never reach the “wrong” customer. Also – will notifications ever contain sensitive data? If so, this introducing a range of security and privacy concern that must be accommodated.

The goal in this phase should be to create a “threat model” that will be used during the entire development lifecycle of the product or feature. This model should be based on security best practices, such as the one that OWASP defines.

The threat model should be the basis of it’s own requirements sessions. A good recommended first step is to schedule such a requirements session using a generic threat model as a template. Within that meeting – define the principles and guidelines to be used during development and testing – what is key to secure? What are the security best practices? It should also define all security controls that should be implemented as part of the development lifecycle.

Another option to consider – require a threat modeling and design meeting prior to scheduling a feature to be part of the sprint and explicitly define who can approve the threat model.

## The Design Phase

Usually, engineering teams create the technical design of a feature. In our notification service example, there might be a new microservice, a message queue, a database to store undelivered notifications, etc. In this phase you must verify that the security design addresses the threat model generated in the requirements phase.

This is where security mechanisms are defined within the design. Key steps include:

* Review the security requirements.
* Identify, through brainstorming, weaknesses and possible design flaws. Use an “attacker” mindset to find weaknesses.
* Assess the different risks using threat modeling techniques – rank them by the severity and probability of the risk.

Development teams will ask for prioritization, and it is essential to come up with the absolutely non-acceptable risk, and the list of items where the risk can be accepted or reduced later.

## The Coding Phase

Embedding security into coding means developers write safe code, eliminate security risks while coding, and do this before they submit their final version for further testing and deployment. To do this, several things need to be considered:

* **Awareness and security mindset** – training and education would significantly raise the bar. There are many people that interact with code changes, and having constant training and awareness will help catch anti-patterns and dangers such as data sanitization, communication encryption, and more. OWASP provides many useful training resources and as a first step understanding the [OWASP Top 10 Security Risks](https://owasp.org/www-project-top-ten/) is a must.
* **Code Scanning** – there are plenty of code analysis scanners that can catch problems as they are being coded. This includes SAST tools and SCA solutions that focus on libraries with open-source vulnerabilities. For many infrastructure declaration languages, there is a specific type of scanner called “Infra-as-Code” scanners, or IaC for short, that can detect insecure configurations. It is important to map the set of technologies being used with the available set of scanners in your organization.
* **Build a Repeatable Process** – create an operational process implementing both points above. For example, have developers run the security scanning tools before they submit their code and also schedule periodic training refreshers for security best practices. One option for integrating security scanning in the process leverages source-code-management systems (like GitHub) that use “pull requests” where developers must submit code for a merge, but the change must first pass a series of checks before approval. This can require a mandatory SAST scan with specifically designated individuals who are authorized to overrule a bad result.

## The Testing Phase

This phase is a bit tricky. Having a true agile SDLC means there is no linear process where everything stops and only testing occurs. Most teams today build some sort of CI/CD pipeline where Continuous Integration occurs, and software is constantly tested.  
In this model, there are a series of tests occurring at multiple stages. Some during coding, some after every code submission, some nightly, and some testing the live production environment.

Within the requirement stage, it is important to map where security tests should run. Previously we mentioned tests during the coding phase. But it is possible to introduce additional tests before deployment – it all depends on your release strategy.

To sum up, two key areas are important for this phase:

* Decide on which tests you need and where based on your release strategy.
* Implement those tests, preferably in an automated fashion so feedback would reach the developer as soon as possible.

There are many other security testing tools available that we didn’t cover in this short blog. To see a comparison of all available security tools and techniques, visit [this page](https://www.softwaretestinghelp.com/differences-between-sast-dast-iast-and-rasp/).

## The Release Phase

In this phase, it is all about having a response plan and being one step ahead of the attacks. You need to build the processes that allow you to A) be alerted if a new external risk becomes known, and B) stay on top of emerging security attacks, trends and solutions.

* **Look for external risks** – processes like penetration tests are important. They will catch risks that might have slipped through all previous security gates. Perform them periodically to maximize the chance of finding them.
* **Handle alerts** – some tools will provide you with threat intel that comes from new disclosures. For example, the log4shell vulnerability had been around in production, but once the vulnerability had been fully recognized, SCA tools alerted their users and security teams to quickly block and remove the vulnerable components. You must assume there are “0-day” vulnerabilities lurking in your apps too that could be discovered at any moment.
* **Stay on top of security trends and technologies** - It’s an ongoing arms race between security professionals and cybercriminals. The security community publishes new threat models and there is constant innovation in the security tooling and techniques.