3.6.2 SDLC Facts

The Software Development Life Cycle (SDLC) is a systematic method for design, development, and change management used for software development and implementation of system and security projects. The primary purpose of SDLC management concepts is to increase the quality of software, both from a functional and security perspective.

The SDLC incorporates the confidentiality, integrity and availability (CIA) of security into the development cycle. Incorporating security at the beginning of the project is easier and less expensive than trying to implement security after development and coding have begun. The phases of the SDLC are as follows:

Phase	Description
Project Initiation	Project initiation involves the following actions: ▪ An original, profitable idea is recognized, and a cost justification is made. Initial security objectives are defined. ▪ Timelines for the project are identified. ▪ The potential users are contacted and involved in the concept development. ▪ Security objectives that the software needs to meet are created. ▪ Initial risk analysis is performed to see if an alternative approach might be beneficial.
Functional Design	Functional design involves the following actions: 1. A project plan is developed. 2. Security activities and checkpoints are identified. 3. Design documentation is developed. 4. Some limited resources are allocated to the project. The security framework is created. 5. The evaluation criteria is identified. 6. The framework of the application is designed, and a prototype of the most critical components is implemented.
System Design	System design identifies the: ▪ Functional model ▪ Behavioral model ▪ Informational model Key output from the system design includes: ▪ Data design ▪ Procedural design ▪ Architectural design Key security decisions made are: ▪ Access controls ▪ Rights and permissions ▪ Encryption algorithms

Software Development and Coding	Development and coding involves three main actions, each of which should be performed by individual groups:
	1. Coding
	2. Testing
	3. Validation
	
	The product goes into major production and is developed by programmers. In this phase:
	▪ The software should be tested in the same environment where it will be used.
	▪ Sometimes backdoors (known as *maintenance hooks*) are unintentionally left by developers. These can later be exploited by attackers. It is important that developers test all aspects of their product for possible backdoors and eliminate any that are found.
	▪ Using modular coding makes implementing changes easier.
	▪ Making sure no vulnerable function calls are used.
	▪ Use dynamic code analysis to detect dependencies.
	▪ Take advantage of peer code review.
	▪ Use design and architectural patterns to cover recurring software limitations or vulnerabilities.
	▪ Two concepts associated with modular coding are high cohesion and low coupling:
	  ▪ *High cohesion* in coding implies that the functions performed by a module are related and clearly defined.
	  ▪ *Low coupling* indicates that a module is not dependent on another module and that changes in the module will not require changes in another module.
	  ▪ Each task in this phase (coding, testing and validation) should be performed by a different group.

Software Installation and Implementation	Installation and implementation involves the following actions: ▪ Formal functional testing is performed by users. ▪ All bugs, vulnerabilities, and risks should be evaluated and documented. ▪ User guides and operational manuals are created. ▪ Certification, accreditation, and auditing are performed.
Release	Software should always be released to a librarian for disposition into production. Installations and routine operations of the application are performed. Security requirements should be included in proposals and contracts. The following list explains the *application vulnerability life cycle*, the chain of events that happen following the release of an application. 1. The application is released. 2. Any bugs that are released in the program are discovered by hackers. 3. Hackers publish the bugs and make them known to the public. 4. Venders develop and release patches. 5. Application users install the patches to their system. 6. Hackers continue to discover vulnerabilities, sometimes as a result of the patch itself. The cycle frequently repeats itself, starting at step two, until the application is no longer in production.
Operations and Maintenance	Operations and maintenance involve the following actions: ▪ As the software is operating in a live environment, operational testing and maintenance should be conducted. ▪ Different types of maintenance, such as patching and changes, might be necessary as the application evolves over time. ▪ Security functions should remain intact in order to efficiently respond to update requirements. ▪ Security-related patches and upgrades should be applied to a system as quickly as possible (with the standard caveat that all patches and upgrades should be tested on non-production systems first). When securing a workstation for use on a secured network, the application of any operating system updates and patches should be performed first. If your organization relies on high-end customized software developed by an external company, code escrow should be implemented. Code escrow is a storage facility hosted by a trusted third party that ensures access to the mission-critical code even if the development company goes out of business.
End of Life	Implementation of disposal. Disposal includes: ▪ Archiving ▪ Overwriting ▪ Destroying An accurate record should be kept of the product and any modifications that were made to it during its lifetime. Additionally, make sure that all applicable laws, regulations, and contractual obligations are met.

Change control is necessary any time a production system is altered. This includes modifications of existing applications, implementation of new applications, removal of old applications, and upgrading or patching software. Change control management is very similar to application development in its processes. A few things to remember about change control are:

- Developers must be isolated from production. If developers are in direct contact with production, the ability to control changes may be lost.
- Changes must be thoroughly documented.
- Security techniques must be implemented at all stages of the process.

Properly documenting and executing the change control process is essential for change to be effective and seamless. The general steps that should be considered are:

1. Recognize a need.
2. Submit a request. This step can be best documented with a change request form.
3. Start a feasibility analysis that includes:
   - Technical feasibility
   - Cost justification
   - Security review
4. Document the change plan.
5. Approach management for approval.
6. Submit the change plan to developers. Developers will then perform the coding for the change.
7. Test the change for conformance to the plan and for security purposes.
8. Document the change.
9. Release the new revision to production through the librarian.

The software product must adhere to organization policies, laws, and contractual obligations. Organizational policies should define the quality of service expected for coding practice and ensure that security practices are integrated.

TestOut Corporation All rights reserved.