Comparison of Software Development Lifecycle Methodologies

James E. Purcell, CISSP, GSEC, GCIH, PMP, MCSE

Comparison of Software Development Lifecycle Methodologies

Introduction

This purpose of this paper is to give the CISSP student an understanding of the Software Development Lifecycle (SDLC) models available to software developers. The student will also see examples of where each model is most appropriate for various types of software development projects. The models covered are the ones most likely to be included in CISSP exam questions.

Note that the SDLC acronym is also used to represent System Development Life Cycle. In many cases, a decision is made to purchase or outsource the software and associated hardware and network systems needed to implement a new application. This is often referred to as the "buy or build" decision. This paper focuses on the Software Development Life Cycle models, but most of the phases, terms, and issues discussed apply to both the "buy" and "build" process.

SDLC Models

To help understand and implement the SDLC phases (Project Initiation, Design Analysis, System Design Specification, Programming and Testing, Installation and Maintenance, and Destruction), various SDLC models have been created by software development experts, universities, and standards organizations. New SDLC models are introduced on a regular basis as new technology and new research (and sometimes new fads) requires new SDLC techniques. Recent new SDLC models include Extreme Programming and Agile Development. This paper looks at the most commonly known and used models and describes situations where the model is an appropriate choice.

Waterfall Model

The Waterfall Model is the oldest and most well-known SDLC model. The distinctive feature of the Waterfall model is its *sequential* step-by-step process from requirements analysis to maintenance.

The major weakness of the Waterfall Model is that after project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team. Because requirements almost always change during long development cycles, often the product that is implemented at the end of the process is obsolete as it goes into production. The Waterfall Model is a poor choice for software development projects where requirements are not well-known or understood by the development team. It might not a good model for complex projects or projects that take more than a few months to complete. Think about doing a home improvement project (such as new hardwood floors) for the first time and only being allowed to go the hardware store one time. The risk the project will fail is high.

What are good candidate software development projects for the Waterfall Model? Systems that have well-defined and understood requirements are a good fit for the Waterfall Model. For instance, a military system aiming an artillery shell is a system with a single, simple requirement; put the shell on the target. This also assumes that the developers have worked on similar systems in the past and are experts in the application domain (artillery fire control systems). To follow the home improvement example, after a visit to a home to get specifications, an experience flooring contractor could install new hardwood floors with only one trip to the hardware store. In this case, the risk of project failure is low.

Spiral Model

In response to the weaknesses and failures of the Waterfall SDLC Model, many new models were developed that add some form of *iteration* to the software development process. In the Spiral SDLC Model, the development team starts with a small set of requirements and goes through each development phase (except Installation and Maintenance) for those set of requirements. Based on lesson learned from the initial iteration (via a risk analysis process), the development team adds functionality for additional requirements in ever-increasing "spirals" until the application is ready for the Installation and Maintenance phase (production). Each of the iterations prior to the production version is a *prototype* of the application.

The advantage of the Spiral Model over the Waterfall Model is that the iterative approach allows development to begin even when all the system requirements are not known or understood by the development team. As each prototype is tested, user feedback is used to make sure the project is on track. The risk analysis step provides a formal method to ensure the project stays on track even if requirements do change. If new techniques or business requirements make the project unnecessary, it can be canceled before too many resources are wasted. In today's business environment, the Spiral Model (or its other iterative model cousins) is the most used SDLC model. An example application development project that would a good candidate for the Spiral Model is an online customer support system where it is not well-understood what services customers want or can accomplish online. Each iterative prototype helps answer the question, "Can and will customers use this system?" The Spiral Model also lets the development team build application domain experience with every iteration. In the home improvement project example, it would be like starting the hardwood flooring project in the upstairs hall to get experience with the flooring process in a low-risk area before moving to living areas where a mistake would be costly.

The Spiral Model combines elements of the Top-down and Bottom-up SDLC models that are discussed in the next sections.

Top-Down Model

The Top-down SDLC model was popularized by IBM in the 1970s, and its concepts are used in other SDLC models such as the Waterfall and Spiral Models previously discussed. In a pure Top-down model, high-level requirements are documented, and programs are built to meet these requirements. Then, the next level is designed and built. A good way to picture the Top-down model is to think of a menu-driven application. The

top level menu items would be designed and coded, and then each sublevel would be added after the top level was finished. Each menu item represents a subsystem of the total application.

The Top-down model is a good fit when the application is a new one and there is no existing functionality that can be incorporated into the new system. A major problem with the Top-down model is that real system functionality is not added and cannot be tested until late in the development process. If problems are not detected early in the project, they can be costly to remedy later.

Bottom-Up Model

In the Bottom-up SDLC model, the lowest level of functionality is designed and programmed first, and finally all the pieces are integrated together into the finished application. This means that, generally, the most complex components are developed and tested first. The idea is that any project show-stoppers will surface early in the project. The Bottom-up model also encourages the development and use of reusable software components that can be used multiple times across many software development projects. Again, think of a menu driven system where the development starts with the lowest level menu items.

The disadvantage of the Bottom-up model is that an extreme amount of coordination is required to be sure that the individual software components work together correctly in the finished system. Few systems are developed purely from the Bottom-up model.

Hybrid Model

The Hybrid SDLC model combines the top-down and bottom-up models. Using the menu driven application example, the design team primarily works top down, but the development team identifies two types of lower level components to work on at the same time as the high-level components. The first type of low-level component would be existing software modules from other projects that can be reused in the new project. The other type of low-level component that would be developed early in the project would be software components with a high risk of failure. This approach allows the development team to make changes to the system early in the project if problems occur with the high-risk components. A new data access technique the team has not used before or a component that might require high amounts of CPU processing time is an example of a high-risk low-level component.

In reality, many of the SDLC models are a variation of the Hybrid Model. The Spiral Model is an example discussed previously.

Rapid Prototyping

With the demand for faster software development, and because of many well-documented failures of traditional SDLC models, Rapid Application Development (RAD) was introduced as a better way to add functionality to an application. The main new tenant of RAD compared to older SDLC models is the use of prototypes. After a quick requirements gathering phase, a prototype application is built and presented to the application users. Feedback from the user provides a loop to improve or add functionality

to the application. Early RAD models did not involve the use of real data in the prototype, but new RAD implementations do use real data.

The advantage of Rapid Prototyping Models is that time-to-market is greatly reduced. Rapid Prototyping skips many of the steps in traditional SDLC models in favor of fast and low-cost software development. The idea is that application software is a "throw-away." If a new version of the software is needed, it is developed from scratch using the newest RAD techniques and tools.

The big disadvantage of the Rapid Prototyping Model is that the process can be to fast, and, therefore, proper testing (especially security testing) may not be done.

The Rapid Prototyping Model is used for graphical user interface (GUI) applications such as web-based applications. Extreme Programming (XP) is a modern incarnation of the Rapid Prototyping Model.

Object-Oriented Model

Object-oriented programming became popular 1990s with the increased use of Smalltalk, C++, and other new object-oriented programming languages. To take full advantage of object-oriented features (i.e. classes, inheritance, methods), the Object-oriented SDLC model was developed.

An important feature of Object-oriented (OO) systems is that software objects represent real-world objects. Objects are derived from Classes, and a class hierarchy allows objects to inherit characteristics from parent classes. This allows software object reuse, less coding, encapsulation of functionality, and many other advantages. A major problem that arose with OO programming is that if the Class hierarchy is not properly designed, all the OO advantages disappear. The object-oriented model attempts to properly define and document the Class hierarchy from which all the system objects are created and object interactions are defined.

For example, in an accounts payable system for GIAC Bikes, GIAC Bikes is an Object. Spacely Space Sprockets is another Object (of class Supplier). Supplier is a Relationship between GIAC Bikes and Spacely Space Sprockets. A Supplier Class has Attributes such as Name, Address, and so on. The Relationship itself may be considered as an Object, having Attributes like Prime Supplier, and so forth. A Supplier Method may be Pay, Drop, Request Refund, and such.

Defining all the application Classes, Methods, Relationships, and Properties is very difficult. The object-oriented SDLC model provides the development team the tools to accomplish this task.

The object-oriented SDLC model has these phases that roughly correspond to the traditional SDLC phases noted in brackets:

1. Object-Oriented Requirements Analysis (OORA) [Design Analysis]: This is where classes of objects and the interaction between them are defined.
2. Object-Oriented Analysis (OOA) [Design Analysis]: In terms of object-oriented concepts, understanding, and modeling a particular problem within a problem domain.

3. Object-Oriented Design (OOD) [System Design Specification]: The object is the basic unit of modularity; objects are instantiations of a class.
4. Object-Oriented Programming (OOP) [Programming and Testing]: Emphasizes the employment of objects and methods rather than types or transformations, as in other programming approaches.

The Object-oriented SDLC model is characterized by its attempt to model real-world entities (such as company, account, employee) into abstract computer software objects and all the interactions that can take place between those objects.

Other Models

Other SDLC models include Model Driven Development, Chaos Model, Agile Programming Model, and many others. One significant trend in the development of new SDLC models is the integration of software design tools into the programming environment (such as Microsoft Visual Studio, IBM Rational Rose). The design team develops a graphical model of the system, and the tool automatically creates a running application.

Summary

SDLC models are tools that allow the development team to correctly follow the SDLC steps to create software that meets a business need. The SDLC models have evolved as new technology and new research have addressed weaknesses of older models. Ideas have been borrowed and adapted between the various models. The CISSP student should recognize the main SDLC model families of sequential (Waterfall), iterative (Spiral), rapid prototyping (RAD), and object-oriented. The student should recognize the characteristics of top-down and bottom-up development.

It is the responsibility of the CISSP to make sure development teams follows an SDLC model so that proper confidentiality, integrity, and availability controls are designed and built into the software application throughout the software life cycle.

Comparing Software Development Life Cycles
>> http://www.giac.org/resources/whitepaper/application/257.php

Defining and Understanding Security in the Software Development Life Cycle
>> http://www.giac.org/resources/whitepaper/application/342.php

SECURITY 519
Web Application Security Workshop

From a mere 26 web servers operating in November 1992 growing to well over 100 million web sites today, we have come a long way in web technology over a short period of time. Today, almost every organization has its own web site for conducting business transactions or other critical functions. And for many companies, their online presence has become a major revenue generator. As everyone jumps on the bandwagon to do business on the web, many problems can arise which are directly related to the security aspects of web applications. The adage "where there is money, there is crime" has become true on a daily basis as we see credit cards and other financial data compromised through web application vulnerabilities. And that is not even the full extent of the problem because web-based malware and worms are still spreading in the wild.

Read More >> http://www.sans.org/info/10801