Secure Coding.
Practical steps to defend your web apps.

Comparing Software Development Life Cycles

Jim Hurst

Comparing Software Development Life Cycles

Introduction

This paper compares several different models of the software development life cycle (SDLC). The SDLC is a structure imposed on the process of developing software, from the scoping of requirements through analysis, design, implementation, and maintenance.

Developing a piece of software is an interesting problem. After all, software is nothing more than bits: information in its purest form. But it also represents human ingenuity, effort, experience, and yes, fallibility. Building good software is difficult, and there is no single best approach. This paper contrasts several of the models used to manage the software development life cycle (SDLC). The SDLC is also sometimes defined as the Systems Development Life Cycle, which is a development process defined by the United States Department of Justice. This paper addresses the first definition.

All the development models examined here can provide excellent results. It is important to remember that these models all solve the same problem and, therefore, must address the same activities and steps. These steps are planning and requirements definition, architecture and design, implementation (where the code is actually written), testing (also known as validation), and deployment.

Software development is a risky proposition. Many projects fail, at great expense. The development methodologies described here represent ways to manage that risk. This paper briefly examines the top-down and bottom-up design philosophies before reviewing the waterfall model, iterative development, spiral model, and agile family, which includes extreme programming.

Top-Down and Bottom-Up Design

Top-down and bottom-up design represent two contrasting approaches for designing a large system. The top-down approach requires a complete design view of the system before any actual coding can begin. It takes the big-picture view of the system and breaks the problem apart into many subsystems that require more design detail. The overall perspective of the project is monolithic: There is one design, and all modules are small parts of it. The top-down style of program design is traditionally associated with procedural languages.

The bottom-up approach, in contrast, emphasizes assembling the big picture by completing many small pieces that work together. These components are then integrated into the larger system. Coding of individual modules, and their testing,

commences before the entire design is complete. Proponents of the bottom-up approach argue that this produces reusable code that saves time later in the process. The bottom-up style of program design is traditionally associated with object-oriented languages such as C++ and Java.

In practice, some blend of top-down and bottom-up is a common compromise: Project managers and architects work on the overall design even as development teams begin building modules and tools. This approach is sometimes called *hybrid design* or *hybrid development*.

The Waterfall Model

The waterfall model is one of the oldest, and perhaps the best-known, software development method. It presents development as a sequential process, proceeding downhill through the phases in the sequence. W. W. Royce first explained the waterfall model in 1970, but he was criticizing the approach at the time, calling it risky and an invitation to failure. The waterfall model is considered one of the most direct approaches, with short development times and minimal costs. However, it does presume an unvarying target: The original specifications cannot change.

In the waterfall model, when a phase is completed, the phase is closed and cannot be revisited. This limitation has lead to widespread criticism of this approach. The software development process involves discovery, and the different phases of development often overlap. Critics maintain that the waterfall model limits options to correct mistakes: if a limitation of the requirements is discovered during the design phase, it is too late to fix it.

Several variations of the waterfall model address these criticisms by allowing some degree of feedback or overlap. These variants begin to blur the distinction between the waterfall model and the iterative processes.

Iterative Processes

When Royce explained the waterfall model in 1970, he was actually writing in defense of the iterative process. The iterative model proceeds by using the lessons learned from each phase to modify the results of the previous phase. Iterative development is sometimes referred to as *incremental development*.

An iterative process begins with a simple implementation of the project requirements. Each iteration adds more functionality until the full design is realized. The lessons learned during each incremental stage of development are applied to refining the design. Learning comes from the development process and from experience using the incomplete system, where possible.

Several iterations might be required before the project is complete. Like the waterfall model, the iterative process begins with a requirements phase followed by a design phase and an implementation phase. After this first round of implementation, an evaluation phase is initiated to evaluate the successes and failures of the work completed. User feedback, performance issues, coding difficulties, unclear or inadequate requirements, and program analysis tools are all used to set the objectives for the next round of requirements, design, and implementation. After these phases have been completed for a second time, another round of evaluation begins. This process repeats until the project is completed.

This approach, of building corrections, design changes, and discoveries into the process, more accurately reflects how most commercial software is developed. Designers miss requirements and make mistakes. Customers often have only a vague idea of what they require. Developers frequently find that initial designs do not adequately reflect the hardware, complexity of the problem, or needs of the users. By making allowance for modifications, iterative processes build flexibility and responsiveness into the process.

Several development models are considered iterative. Discussed here are the spiral model and the agile methods, including extreme programming.

The Spiral Model

Barry Boehm first proposed the spiral model in 1988. Although the iterative model was used well before this, Boehm was the first to explain why iteration is important to producing software that meets customer expectations. Development proceeds through the stages: requirements, design, implementation, and testing. When testing at the end of a cycle is completed, the next step is the planning phase of a new cycle that adds additional features and components. Each cycle of the spiral involves stepping through all the phases. The spiral model tries to combine the advantages of the top-down and bottom-up approaches and is often used in larger projects. For smaller projects, the agile methods are often preferred.

The Agile Model

The agile development processes were developed during the 1990s as a reaction to the “heavyweight” models being used at that time. Agile methods represent a family of development methods, rather than a single approach. They emphasize communication between all project members and encourage locating the entire team in one location. “Customers” may be actual customers or may be product managers or business analysts. In all cases, customer participation is welcome and encouraged.

The agile models attempt to minimize project risk by dividing the project into short iterations (called timeboxes) that normally last between a week and a month. Each timebox represents a project in miniature, with planning, requirements, design, coding, testing, and documentation. The goal of an agile project is to release software at the end of each iteration, even if this release is not a complete product. Project priorities are re-evaluated at the end of each timebox.

Agile methods aim to satisfy customers with rapid, continuous deliveries of useful software, delivered in weeks rather than months. The principal measure of progress is working software. Close, face-to-face communication is expected between developers and business people. The project should adapt to changing circumstances, and even late changes in the requirements are welcome.

Extreme Programming

The best known of the agile development methods is extreme programming, also known as XP. Extreme programming prescribes a set of day-to-day practices for developers and managers. Fans of XP say that these practices represent traditional software engineering taken to an extreme degree, and that this produces high-quality results more attuned to customer needs.

One of the key goals of XP is to reduce the cost of changes during the development process: XP sees requirements changes as an inevitable, normal, and even desirable part of the process. The development team should adapt to changes gracefully, without putting schedules or projects at risk.

Extreme programming recognizes a set of five values as being critical to development success: communication, simplicity, feedback, courage, and respect. Communication refers to communicating the requirements to the developers, spreading project knowledge rapidly among team members, and sharing results and ideas with customers in many interactions. XP recommends starting with the simplest solution and rewriting to more complex solutions only as required. This focus of coding and designing only for today’s needs is one of the key differences between XP and other methods. Proponents argue that, while this entails the overhead of rewriting, it is made up for by the advantage of only developing the pieces required by the final system. Feedback refers to input from the system, customers, and team. Feedback from the system comes from testing and validation efforts. Feedback from the customers requires customer involvement at a daily level. The development team must then respond to the new requirements and issues.

Courage seems an odd value for a development methodology. It requires team members to develop code for today, not tomorrow. It requires courage to rewrite (or refactor) the code as necessary, and courage to know when to throw away code that no longer meets the requirements. The respect value requires that

team members respect each other and, thus, commit to not making changes that invalidate completed work, and that they respect their work, maintaining a commitment to quality and the best design possible.

Rapid Prototyping

Rapid prototyping is less a development methodology than a software engineering tool. Rapid prototyping incorporates three steps: requirements, prototyping, and user evaluation. The goal is to quickly assemble a mock-up of the system for evaluation. This usually means quickly assembling a user interface, using dummy data, and getting customer evaluations.

Rapid prototyping can be useful in gathering requirements about the user experience. For small projects, prototype code can be incorporated into the final product. In larger projects, the prototype is looked upon as part of the requirements and evaluation steps, and the code is generally discarded.

Summary

This paper has reviewed different approaches in design and software development used to manage risk. Top-down design emphasizes a complete design, often composed of many subsystems, before any coding begins. Bottom-up design, instead, requires early coding and testing of modules.

Several software development methods were discussed. All these methods must include phases of planning and requirements, design and architecture, implementation, testing, and deployment. The waterfall model is direct and cost effective, but it is inflexible and does not gracefully handle changes to requirements after design has begun. It proceeds through the phases a single time, with no going back to previous phases. The iterative processes, in contrast, expect to revisit each phase, to incorporate changes and lessons learned. The spiral model uses several cycles that begin with requirements and design and end with evaluation of implementation during the cycles. The agile family of methods uses short iterations, or timeboxes, that are each mini-development projects. They emphasize team communication and flexibility in requirements. The best-known agile method is extreme programming, which prescribes practices meant to encourage the values of communication, simplicity, feedback, courage, and respect. Rapid prototyping is a technique of quickly assembling a user interface for evaluation.

References

Project Lifecycle Models: How They Differ and When to Use Them. Business Evolution. http://www.business-esolutions.com/islm.htm

A Spiral Model of Software Development and Enhancement. Barry Boehm. http://www.sce.carleton.ca/faculty/ajila/4106-5006/Spiral%20Model%20Boehm.pdf

Last Updated: September 20th, 2017

Upcoming SANS App Sec Training

Secure DevOps Summit & Training	**Denver, CO**	**Oct 10, 2017 - Oct 17, 2017**	**Live Event**
SANS Seattle 2017	**Seattle, WA**	**Oct 30, 2017 - Nov 04, 2017**	**Live Event**
SANS Austin Winter 2017	**Austin, TX**	**Dec 04, 2017 - Dec 09, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017	**Washington, DC**	**Dec 12, 2017 - Dec 19, 2017**	**Live Event**
SANS Cyber Defense Initiative 2017 - DEV522: Defending Web Applications Security Essentials	**Washington, DC**	**Dec 14, 2017 - Dec 19, 2017**	**vLive**
SANS Security East 2018	**New Orleans, LA**	**Jan 08, 2018 - Jan 13, 2018**	**Live Event**
SANS Amsterdam January 2018	**Amsterdam, Netherlands**	**Jan 15, 2018 - Jan 20, 2018**	**Live Event**
SANS Oslo 2018	**Oslo, Norway**	**Feb 05, 2018 - Feb 10, 2018**	**Live Event**
Community SANS Indianapolis DEV534	**Indianapolis, IN**	**Feb 05, 2018 - Feb 06, 2018**	**Community SANS**
Cloud Security Summit & Training 2018	**San Diego, CA**	**Feb 19, 2018 - Feb 26, 2018**	**Live Event**
Communtiy SANS Seattle DEV534	**Seattle, WA**	**Feb 26, 2018 - Feb 27, 2018**	**Community SANS**
SANS San Francisco Spring 2018	**San Francisco, CA**	**Mar 12, 2018 - Mar 17, 2018**	**Live Event**
SANS Northern VA Spring - Tysons 2018	**Tysons, VA**	**Mar 17, 2018 - Mar 24, 2018**	**Live Event**
SANS 2018	**Orlando, FL**	**Apr 03, 2018 - Apr 10, 2018**	**Live Event**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**