## Lesson Proper for Week 5

**Programming**

As we learned in previous lessons, software is created via programming. Programming is the process of creating a set of logical instructions for a digital device to follow using a programming language. The process of programming is sometimes called “coding” because the syntax of a programming language is not in a form that everyone can understand – it is in “code.”

The process of developing good software is usually not as simple as sitting down and writing some code. True, sometimes a programmer can quickly write a short program to solve a need. But most of the time, the creation of software is a resource-intensive process that involves several different groups of people in an organization. In the following sections, we are going to review several different methodologies for software development.

**Systems Development Life Cycle**

The first development methodology we are going to review is the systems-development life cycle (SDLC). This methodology was first developed in the 1960s to manage the large software projects associated with corporate systems running on mainframes. It is a very structured and risk-averse methodology designed to manage large projects that included multiple programmers and systems that would have a large impact on the organization.

Various definitions of the SDLC methodology exist, but most contain the following phases.

1.      Preliminary Analysis. In this phase, a review is done of the request. Is creating a solution possible? What alternatives exist? What is currently being done about it? Is this project a good fit for our organization? A key part of this step is a feasibility analysis, which includes an analysis of the technical feasibility (is it possible to create this?), the economic feasibility (can we afford to do this?), and the legal feasibility (are we allowed to do this?). This step is important in determining if the project should even get started.

2.      System Analysis. In this phase, one or more system analysts work with different stakeholder groups to determine the specific requirements for the new system. No programming is done in this step. Instead, procedures are documented, key players are interviewed.

3.      System Design. In this phase, a designer takes the system-requirements document created in the previous phase and develops the specific technical details required for the system. It is in this phase that the business requirements are translated into specific technical requirements. The design for the user interface, database, data inputs and outputs, and reporting are developed here. The result of this phase is a system-design document. This document will have everything a programmer will need to actually create the system.

4.      Programming. The code finally gets written in the programming phase. Using the system-design document as a guide, a programmer (or team of programmers) develop the program. The result of this phase is an initial working program that meets the requirements laid out in the system-analysis phase and the design developed in the system-design phase.

5.      Testing. In the testing phase, the software program developed in the previous phase is put through a series of structured tests. The first is a unit test, which tests individual parts of the code for errors or bugs. Next is a system test, where the different components of the system are tested to ensure that they work together properly. Finally, the user-acceptance test allows those that will be using the software to test the system to ensure that it meets their standards. Any bugs, errors, or problems found during testing are addressed and then tested again.

6.      Implementation. Once the new system is developed and tested, it has to be implemented in the organization. This phase includes training the users, providing documentation, and conversion from any previous system to the new system. Implementation can take many forms, depending on the type of system, the number and type of users, and how urgent it is that the system become operational. These different forms of implementation are covered later in the chapter.

7.      Maintenance. This final phase takes place once the implementation phase is complete. In this phase, the system has a structured support process in place: reported bugs are fixed and requests for new features are evaluated and implemented; system updates and backups are performed on a regular basis.

The SDLC methodology is sometimes referred to as the waterfall methodology to represent how each step is a separate part of the process; only when one step is completed can another step begin. After each step, an organization must decide whether to move to the next step or not. This methodology has been criticized for being quite rigid. For example, changes to the requirements are not allowed once the process has begun. No software is available until after the programming phase.

Again, SDLC was developed for large, structured projects. Projects using SDLC can sometimes take months or years to complete. Because of its inflexibility and the availability of new programming techniques and tools, many other software-development methodologies have been developed. Many of these retain some of the underlying concepts of SDLC but are not as rigid.

**The RAD methodology consists of four phases:**

1.      Requirements Planning. This phase is similar to the preliminary-analysis, system-analysis, and design phases of the SDLC. In this phase, the overall requirements for the system are defined, a team is identified, and feasibility is determined.

2.      User Design. In this phase, representatives of the users work with the system analysts, designers, and programmers to interactively create the design of the system. One technique for working with all of these various stakeholders is the so-called JAD session. JAD is an acronym for joint application development. A JAD session gets all of the stakeholders together to have a structured discussion about the design of the system. Application developers also sit in on this meeting and observe, trying to understand the essence of the requirements.

3.      Construction. In the construction phase, the application developers, working with the users, build the next version of the system. This is an interactive process, and changes can be made as developers are working on the program. This step is executed in parallel with the User Design step in an iterative fashion, until an acceptable version of the product is developed.

4.      Cutover. In this step, which is similar to the implementation step of the SDLC, the system goes live. All steps required to move from the previous state to the use of the new system are completed here.

As you can see, the RAD methodology is much more compressed than SDLC. Many of the SDLC steps are combined and the focus is on user participation and iteration. This methodology is much better suited for smaller projects than SDLC and has the added advantage of giving users the ability to provide feedback throughout the process. SDLC requires more documentation and attention to detail and is well suited to large, resource-intensive projects. RAD makes more sense for smaller projects that are less resource-intensive and need to be developed quickly.

**Agile Methodologies**

Agile methodologies are a group of methodologies that utilize incremental changes with a focus on quality and attention to detail. Each increment is released in a specified period of time (called a time box), creating a regular release schedule with very specific objectives. While considered a separate methodology from RAD, they share some of the same principles: iterative development, user interaction, ability to change. The agile methodologies are based on the “Agile Manifesto,” first released in 2001.

**The characteristics of agile methods include:**

·       small cross-functional teams that include development-team members and users; daily status meetings to discuss the current state of the project;

·       short time-frame increments (from days to one or two weeks) for each change to be completed; and

·       at the end of each iteration, a working project is completed to demonstrate to the stakeholders.

The goal of the agile methodologies is to provide the flexibility of an iterative approach while ensuring a quality product.

based on the feedback, until a final product is completed.

The biggest difference between the lean methodology and the other methodologies is that the full set of requirements for the system are not known when the project is launched. As each iteration of the project is released, the statistics and feedback gathered are used to determine the requirements. The lean methodology works best in an entrepreneurial environment where a company is interested in determining if their idea for a software application is worth developing.

**Programming Languages**

As I noted earlier, software developers create software using one of several programming languages. A programming language is an artificial language that provides a way for a programmer to create structured code to communicate logic in a format that can be executed by the computer hardware. Over the past few decades, many different types of programming languages have evolved to meet many different needs. One way to characterize programming languages is by their “generation.”

**Generations of Programming Languages**

Early languages were specific to the type of hardware that had to be programmed; each type of computer hardware had a different low-level programming language (in fact, even today there are differences at the lower level, though they are now obscured by higher-level programming languages). In these early languages, very specific instructions had to be entered line by line – a tedious process.

First-generation languages are called **machine code**. In machine code, programming is done by directly setting actual ones and zeroes (the bits) in the program using binary code. Here is an example program that adds 1234 and 4321 using machine language:

Third-generation languages are not specific to the type of hardware on which they run and are much more like spoken languages. Most third-generation languages must be compiled, a process that converts them into machine code. Well-known third-generation languages include BASIC, C, Pascal, and Java. Here is an example using BASIC:

Fourth-generation languages are a class of programming tools that enable fast application development using intuitive interfaces and environments. Many times, a fourth-generation language has a very specific purpose, such as database interaction or report-writing. These tools can be used by those with very little formal training in programming and allow for the quick development of applications and/or functionality. Examples of fourth-generation languages include: Clipper, FOCUS, FoxPro, SQL, and SPSS.

**Compiled vs. Interpreted**

Besides classifying a program language based on its generation, it can also be classified by whether it is compiled or interpreted. As we have learned, a computer language is written in a human-readable form. In a compiled language, the program code is translated into a machine-readable form called an executable that can be run on the hardware. Some well-known compiled languages include C, C++, and COBOL. An interpreted language is one that requires a runtime program to be installed in order to execute. This runtime program then interprets the program code line by line and runs it. Interpreted languages are generally easier to work with but also are slower and require more system resources. Examples of popular interpreted languages include BASIC, PHP, PERL, and Python. The web languages of HTML and Javascript would also be considered interpreted because they require a browser in order to run. The Java programming language is an interesting exception to this classification, as it is actually a hybrid of the two. A program written in Java is partially compiled to create a program that can be understood by the Java Virtual Machine (JVM). Each type of operating system has its own JVM which must be installed, which is what allows Java programs to run on many different types of operating systems.

**Procedural vs. Object-Oriented**

A procedural programming language is designed to allow a programmer to define a specific starting point for the program and then execute sequentially. All early programming languages worked this way. As user interfaces became more interactive and graphical, it made sense for programming languages to evolve to allow the user to define the flow of the program. The object-oriented programming language is set up so that the programmer defines “objects” that can take certain actions based on input from the user. In other words, a procedural program focuses on the sequence of activities to be performed; an object-oriented program focuses on the different items being manipulated.