Chapter 10 – Approaches to System Development

Table of Contents

Chapter Overview

The chapter first presents and explains the differences in the life cycle approach—the predictive and the adaptive approaches. These two approaches are really a continuum and any give project may have elements of both approaches. The predictive approach to the SDLC is used for projects that are well understood and low risk. The adaptive approach to the SDLC is used for projects that are not well understood and are higher risk. Adaptive SDLCs are more iterative and allow the project team to adapt the project to changing circumstances.

The chapter includes a discussion of the Agile philosophy and Agile modeling principles. Students should understand that Agile philosophy recognizes that most users do not understand the complexities of business processes and supporting software and therefore most projects must be flexible and agile. Agile modeling has basic principles to encourage developers to use modeling as a means to the end product (working software), and not an end in and of itself.

Finally the chapter concludes with a discussion of three methodologies that embrace the Agile philosophy – Unified Process (UP), Extreme Programming (XP), and Scrum.

Learning Objectives

After reading this chapter, the student should be able to:

- Compare the underlying assumptions and uses of a predictive and an adaptive system development life cycle (SDLC)
- Explain what makes up a system development methodology—the SDLC as well as models,

tools, and techniques

- Describe the key features of Agile development
- Understand and describe the key features of the Unified Process, Extreme Programming, and Scrum Agile system development methodologies

Notes on Opening Case and EOC Cases

Opening Case

Development Approaches at Ajax Corporation, Consolidated Concepts, and Pinnacle Manufacturing: This case briefly describes various development approaches used by three different companies. Each company uses a different approach with different techniques. Some techniques were based on structured methods; another company used object-oriented develop techniques; and the third company were experimenting with Agile techniques. The point of the case is that there are many different ways to develop software and students will encounter various combinations when they begin working in industry. When students interview for employment it is also a good idea if they ask questions about the approach, techniques, and methods used by the company with which they are interviewing. Some companies like to focus on the latest approaches while at the other extreme are companies that have not updated their approaches for many years.

EOC Cases

A “College Education Completion” Methodology: This case is based on the system oriented principles covered in the chapter. The student is asked to think of his college education and to divide it into phases. Then students are asked to identify specific activities that might apply to each phase. Finally they are asked to identify specific techniques. This case requires that the students think in terms of abstract system concepts and development cycle concepts and apply them to an entirely different context.

Community Board of Realtors (running case)**:** Community Board of Realtors is a professional organization that supports real estate offices and agents. This case is a smaller case in that it has few use cases and a simple database structure. The students are asked to think about developing the system with both a predictive project and an adaptive project. They are asked additional questions about an iterative approach, including how to iterate if the total solution included support for mobile devices.

Spring Breaks 'R' Us Travel Services (SBRU) (running case)**:** SBRU is an online travel services that books spring break trips to resorts for college students. This is fairly sophisticated system with four subsystems and various users and different database tables. The student is asked to identify classes for all four subsystems (in addition to a previous chapter's solutions for one subsystem). Then students are asked how to structure the project as an iterative project, especially with regard to implementation of the various classes.

On the Spot Courier Services (running case)**:** On the Spot is a small, but growing, courier service that needs to track customers, package pickups, package deliveries, and delivery routes. In this chapter, we introduce four subsystems. The student is asked to organize an adaptive, iterative project to develop these four subsystems. Various questions help the student think about the ramifications of organizing the project in different ways. The student is also asked about which models are necessary and to what depth is modeling require based on the Agile principles given in the chapter.

Sandia Medical Devices (running case)**:** Sandia Medical Devices is a company that specializes in medical monitoring through remote, mobile telecommunication devices. This chapter describes the project team, including users assigned to the project, and the anticipated project schedule. The use cases to be implemented are also identified. The student is asked to comment on the anticipated project schedule, the use of Agile techniques, and the potential problems with user involvement.

Instructor's Notes

The Systems Development Life Cycle

Key Terms

- **predictive approach to the SDLC** – an approach that assumes the project can be planned in advance and that the new information system can be developed according to the plan
- **adaptive approach to the SDLC** – an approach that assumes the project must be more flexible and adapt to changing needs as the project progresses
- **phases** – related groups of development activities, such as planning, analysis, design, implementation, and support
- **waterfall model** – an SDLC approach that assumes the phases can be completed sequentially with no overlap
- **incremental development** – an SDLC approach that completes portions of the system in small increments across iterations, with each increment being integrated into the whole as it is completed
- **walking skeleton** – a development approach in which the complete system structure is built but with bare-bones functionality

Lecture Notes

A project is a planned undertaking that has a beginning and an end, and which produces a predetermined result or product. The term system development project describes a planned undertaking, which is normally a large job that produces a new information system. Success depends heavily on having an organized, methodical sequence of tasks and activities that culminate with an information system that is reliable, robust, and efficient.

One of the key concepts in system development is the systems development life cycle (SDLC). The SDLC refers to the entire process of building, deploying, using, and updating an information system.

The other major concept in this chapter that students should learn well relates to the two types of SDLC approaches. The nature of the project determines the best approach to use. A **predictive approach** to the SDLC assumes that the development project is planned in advance and that the new information system can be developed according to the plan. An **adaptive approach** to the SDLC is used when the exact requirements or needs of the users are not well understood. A more flexible approach is needed that allows the plan to be modified as the project progresses. Figure 10-1 shows the continuum from completely predictive to completely adaptive projects.

Traditional Predictive Approaches to the SDLC

In every project there must be activities associated with project initiation, planning, analysis, design, implementation, and deployment. Each of these sets of activities are called a **phase**. There is another phase, called support, which consists of those ongoing activities to maintain the system once it is in production. This text teaches the basic concepts associated with the initiation, planning, analysis, design, implementation, and the deployment phases of an SDLC. The following list identifies the objective of each of the six phases:

- initiation – activities to get the project identified, approved, and budgeted
- planning – activities to scope the project, plan and schedule the work, and identify the required resources
- analysis – activities to understand the user requirements
- design – activities to define and structure the solution system
- implementation – programming activities and other activities to build the solution and database
- deployment – activities associated with data conversion, final testing, and putting the system into production

The SDLC that is the most predictive is called the **waterfall model**, an SDLC that assumes the various phases of a project can be completed sequentially—one phase falls into the next phase, and there is no going back as shown in Figure 10-3. This approach is almost never used any more. It never worked very well. Modified waterfall approaches are more flexible and allow considerable overlap of the phases. The six phases tend to follow one after the other, but there is always a lot of overlap. Figure 10-4 shows how a modified waterfall model might work. Not only is a modified waterfall more effective, but it is more efficient by allowing developers to multi-task with analysis, design, and programming. However, the overall approach is to have one big project and to develop the system in one large continuous sequence of activities.

Newer Adaptive Approaches to the SDLC

In contrast to the predictive SDLC, the adaptive SDLC assumes project activities have to be adjusted as the project progresses. This is necessary because aspects of the project are not well understood at the beginning. Rather than having the analysis, design, and implementation phases proceed sequentially with some overlap, iterations can be used to create a series of mini-projects that address smaller parts of the application. One of these smaller parts is analyzed, designed, built, and tested during a single iteration; then, based on the results, the next iteration proceeds to analyze, design, build, and test the next smaller part.

Related to the idea of an iterative project is the concept of **incremental development**. Obviously by having a limited scope for each interaction, we are only developing a portion – and increment – of the total system. An increment may be completed in one or more iterations. Sometimes this is also referred to as an organic approach because the system is growing during the life of the overall project.

One approach to incremental development is to build the overall structure of the system first, but with very limited functionality. This is usually called a **walking skeleton**. The system is "fleshed out" over time as it is built tested.

Quick Quiz

Q: What is the basic difference between the predictive approach and the adaptive approach?

> A: The predictive approach tries to predict the entire scope of the project and execute it in one large process, while the adaptive approach recognizes that the project changes as the users and development team learn and so the project has to adapt during its lifecycle.

Q: What is the standard type of predictive approach called? How does it work?

> A: The Waterfall model. One phase completes and the project team moves (falls) to the next phase.

Q: What does incremental development mean?

> A: It means the system is developed in increments, also referred to as "growing" the system.

Q: What is a walking skeleton? What does the name mean?

> A: It is a skeleton because it has very little functionality, i.e. very little "flesh." It is walking because it can execute.

Methodologies, Models, Tools, and Techniques

Key Terms

- **tool** – a software application that assists developers in creating models or other components required for a project
- **integrated development environments (IDEs)** – a set of tools that work together to provide a comprehensive development and programming environment for software developers
- **visual modeling tools** – tools that help analysts create and verify graphical models and may also generate program code
- **technique** – guidelines to specify a method for how to carry out a development activity or task

Lecture Notes

Methodologies

A system development methodology provides guidelines for every facet of the systems development life cycle. Some methodologies (whether built in-house or purchased) contain massive written documentation that defines everything the developers may need to produce at any point in the project. Other methodologies are much more informal.

Models

A model is a representation of an important aspect of the real world. Sometimes, the term abstraction is used because we abstract (separate out) an aspect that is of particular importance to us. The models used in system development include representations of inputs, outputs, processes, data, objects, object interactions, locations, networks, and devices, among other things. Most of the models are graphical models, which are drawn representations that employ agreed-upon symbols and conventions.

Tools

In the context of system development, a tool is software support that helps create models or other components required in the project, such as a drawing tool to create graphical models. Tools have been specifically designed to help system developers. Programmers should be familiar with integrated development environments (IDEs), which include many tools to help with programming tasks.

Techniques

In system development, a technique is a collection of guidelines that helps an analyst complete an activity or task. It often includes step-by-step instructions for creating a model, or it might include more general advice on collecting information from system users. A methodology includes a collection of techniques that are used to complete activities within each phase of the systems development life cycle.

Quick Quiz

Q: What is the difference between a methodology and a technique?

A: A methodology covers the entire process of development and defines the appropriate techniques, models, and tools to be used.

Q: What is the difference between a model and a tool?

A: A model is an abstraction of the real world that is produced, often by a tool.

Q: What is the difference between a technique and a tool?

A: A technique explains the how to do something, where a tool is something that is used, possible as part of the technique.

Agile Development

Key Terms

- **Agile development** – a guiding philosophy and set of guidelines for developing information systems in an unknown, rapidly changing environment

- **chaordic** – a term used to describe adaptive projects that are chaotic and ordered
- **Agile modeling (AM)** – guiding philosophy in which only models that are necessary, with a valid need and at the right level of detail, are created

Lecture Notes

Agile development is a philosophy and set of guidelines for developing information systems in an unknown, rapidly changing environment, and it can be used with any system development methodology. Agile modeling is a philosophy about how to build models, some of which are formal and detailed, others sketchy and minimal. All the models in this text can be used with Agile modeling.

Agile Development Philosophy and Values

The Agile philosophy takes a flexible approach with project schedules and letting the project teams plan and execute their work as the project progresses. The core philosophy of Agile development includes:

- Value responding to change over following a plan
- Value individuals and interactions over processes and tools
- Value working software over comprehensive documentation
- Value customer collaboration over contract negotiation

Some industry leaders in the Agile movement coined the term ***chaordic*** to describe an Agile project. Chaordic comes from two words: chaos and order. The Agile philosophy recognizes this unpredictability, handling it with increased flexibility and by trusting the project team to develop solutions to project problems. Another important aspect of Agile development is that customers must continually be involved with the project team. They don’t just sit down with the project team for a few sessions to develop the specifications. They become part of the technical team.

Agile Modeling Principles

Agile modeling (AM) isn’t about doing less modeling but about doing the right kind of modeling at the right level of detail for the right purposes. AM doesn’t dictate which models to build or how formal to make those models. Instead, it helps developers stay on track with their models by using them as a means to an end rather than end deliverables. The following Agile modeling principles indicate that building models is an essential technique for software development, but that the models are the means and not the ends.

- Develop Software as Your Primary Goal
- Enable the Next Effort as Your Secondary Goal
- Minimize Your Modeling Activity—Few and Simple
- Embrace Change, and Change Incrementally
- Model with a Purpose

- Build Multiple Models
- Build High-Quality Models and Get Feedback Rapidly
- Focus on Content Rather Than Representation
- Learn from Each Other with Open Communication
- Know Your Models and How to Use Them
- Adapt to Specific Project Needs
- Maximize Stakeholder ROI

Quick Quiz

Q: With which project methodology can you follow Agile modeling techniques?

A: With either, predictive or adaptive.

Q: With which project methodology does the Agile philosophy more closely follow?

A: Adaptive projects tend to be more flexible and are more agile.

Q: What are the four elements of the Agile philosophy?

A: Value responding to change over following a plan; Value individuals and interactions over processes and tools; Value working software over comprehensive documentation; Value customer collaboration over contract negotiation.

The Unified Process, Extreme Programming, and Scrum

Key Terms

- **UP discipline** – a set of functionally related activities that combine to enable the development process in a UP project
- **pair programming** – XP practice in which two programmers work together on designing, coding, and testing software
- **refactoring** – revising, reorganizing, and rebuilding part of a system so it is of higher quality
- **product backlog** – a prioritized list of user requirements used to choose work to be done in a Scrum project
- **product owner** – the client stakeholder for whom the system is being built
- **Scrum master** – the person in charge of a Scrum project—similar to a project manager
- **sprint** – a time-controlled mini-project that implements a specific portion of a system

Lecture Notes

The Agile philosophy only proposes principles; it isn't meant to be a complete methodology, with practices and action steps. This section presents three methodologies (UP, XP, and Scrum) that incorporate Agile principles, but are also complete methodologies, with specific techniques and practices.

The Unified Process

The Unified Process (UP) is an object-oriented system development methodology originally offered by Rational Software, which is now part of IBM. The UP is now widely recognized as a highly influential innovation in software development methodologies for object-oriented development using an adaptive approach.

A **UP phase** can be thought of as a goal or major emphasis for a particular portion of the project. The four phases of the UP life cycle are inception, elaboration, construction, and transition, as shown in Figure 10-11.

- Inception – Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimates for cost and schedule.
- Elaboration – Define the vision, identify and describe all requirements, finalize the scope, design and implement the core architecture and functions, resolve high risks, and produce realistic estimates for cost and schedule. Transition Complete the beta test and deployment so users have a working system and are ready to benefit as expected.
- Construction – Iteratively implement the remaining lower-risk, predictable, and easier elements and prepare for deployment.
- Transition – Complete the beta test and deployment so users have a working system and are ready to benefit as expected.

The four UP phases define the project sequentially by indicating the emphasis of the project team at any point in time. To make iterative development manageable, the UP defines disciplines to use within each iteration. A **UP discipline** is a set of functionally related activities that contributes to one aspect of the development project. UP disciplines include business modeling, requirements, design, implementation, testing, deployment, configuration and change management, project management, and environment. Each iteration usually involves activities from all disciplines.

The six main UP development disciplines are:

- Business modeling
- Requirements
- Design
- Implementation
- Testing

- Deployment

Three additional support disciplines are necessary for planning and controlling the project:

- Configuration and change management
- Project management
- Environment

Extreme Programming

Extreme Programming (XP) is an adaptive, Agile development methodology that was created in the mid-1990s. The word extreme sometimes makes people think that this methodology is completely new and that developers who embrace XP are radicals. However, XP is really an attempt to take the best practices of software development and extend them "to the extreme." Extreme programming has these characteristics:

- Takes proven industry best practices and focuses on them intensely
- Combines those best practices (in their most intense forms) in a new way to produce a result that is greater than the sum of its parts

The four **core values** of XP—communication, simplicity, feedback, and courage—drive its practices and project activities.

- **Communication.** Effective communication involves not only documentation, but also verbal discussion. The practices and methods of XP are designed to ensure that open, frequent communication occurs.
- **Simplicity.** XP includes techniques to reinforce this principle and make it a standard way of developing systems.
- **Feedback.** Feedback on functionality and requirements should come from the users, feedback on designs and code should come from other developers, and feedback on satisfying a business need should come from the client.
- **Courage.** Developers always need courage to face the harsh choice of doing things right or throwing away bad code and starting over.

XP's 12 practices embody the basic values just presented.

- Planning – Based on user stories. Planning involves two aspects: business issues and technical issues. In XP, the business issues are decided by the users and clients, whereas technical issues are decided by the development team.
- Testing – Every new piece of software requires testing, and every methodology includes testing. XP intensifies testing by requiring that the tests for each story be written first—before the

solution is programmed.

- **Pair Programming** – More than any other, this practice is one for which XP is famous. First, one programmer might focus more on design and double-checking the algorithms while the other writes the code. Then, they switch roles.
- Simple designs
- Refactoring the code – Before and after adding any new functions, XP programmers review their code to see whether there is a simpler design or a simpler method of achieving the same result.
- Owning the code collectively – In XP, everyone is responsible for the code.
- Continuous integration – This practice embodies XP's idea of "growing" the software. Small pieces of code—which have passed the unit tests—are integrated into the system daily or even more often.
- On-site customer
- System metaphor – This practice is XP's unique and interesting approach to defining an architectural vision. It answers the questions "How does the system work?" and "What are its major components?" And it does it by having the developers identify a metaphor for the system.
- Small releases
- Forty-hour week
- Coding standards

Figure 10-16 shows an overview of the XP Project Activities for the system development approach. It is divided into three levels: system (the outer ring), release (the middle ring), and iteration (the inner ring). System-level activities occur once during each development project. The first XP development activity is creating user stories, which are similar to use cases in object-oriented analysis. The first release-level activity is planning a series of iterations. Each iteration focuses on a small (possibly just one) system function or user story. When an iteration passes integration testing, work begins on the next iteration.

Scrum

Those of you who are familiar with rugby are aware that when a team gets possession of the ball, it attempts to go the entire distance in one continuous play—from point of possession to the score. Combining some of these principles of rugby with the Agile philosophy gave rise to a methodology—the objective of which is to be quick, agile, and intense and to go the entire distance.

The basic control mechanism for a Scrum project is a list of all the things the system should include and address. This list—called the **product backlog** includes user functions (such as use cases), features (such as security), and technology (such as platforms). The product backlog list is continually being prioritized, and only a few of the high-priority items are worked on at a time, according to the current needs of the project and its sponsor.

The three main organizational elements that affect a Scrum project are the **product owner**, the **Scrum master**, and the Scrum team or teams. The product owner is the client, but he or she has additional responsibilities. Remember that in Agile development, the user and client are closely involved in the project. In Scrum, the product owner maintains the product backlog list.

The Scrum master enforces Scrum practices and helps the team complete its work. A Scrum master is comparable to a project manager in other approaches. However, because the team is self-organizing and no overall project schedule exists, the Scrum master’s duties are slightly different. He or she is the focal point for communication and progress reporting. In other words, the Scrum master is a facilitator.

The Scrum practices are the mechanics of how a project progresses. The basic work process is called a sprint, and all other practices are focused on supporting a sprint. Every day during the sprint, the Scrum master holds a daily Scrum, which is a meeting of all members of the team. The objective is to report progress. The meeting is limited to 15 minutes or some other short time period. Members of the team answer only three questions:

- What have you done since the last daily Scrum (during the last 24 hours)?
- What will you do by the next daily Scrum?
- What kept you or is keeping you from completing your work?

Quick Quiz

Q: What are the four UP Phases? What do they represent?

A: Inception, Elaboration, Construction, Transition. They represent a “focus” during that part of the project. For example, inception is how to get things started.

Q: What are UP Disciplines? What are the six primary disciplines? What are the three project control disciplines?

A: The UP disciplines are the related activities that carry out the activities of the projects The six primary disciplines are business modeling defining requirements, design, implementation, testing, and deployment. The three project control disciplines are Configuration, project management, and environment.

Q: What does XP stand for? What are the four core values for XP?

A: XP means Extreme Programming. The four core values are Communication, Simplicity, Feedback, and Courage.

Q: Explain how pair programming works.

A: Pair programing is where two programmers work together to design and program a part of the system. One programmer will work on design while the other writes code. Then they change. They also check each other’s work and provide assistance to each other.

Q: What is a Scrum sprint? What is the primary purpose of a sprint?

A: A Scrum sprint is a front to back development and programming period whose purpose is to finish (i.e. produce working code) a portion of the system. Its purpose is to produce working code.

Q: What is a product backlog? Who maintains it?

A: The product backlog is the list of items that need to be completed. It is maintain by the product owner, which is usually a client person.

Classroom Activities

This little activity will help students think about an SDLC. If you have formed teams, have the students do the activity in their teams. If not, just have work in groups of two, three, or four. Another way may be to have teams from half the class try to do a predictive project p;an and the other half try to do an adaptive, iterative project plan.

The college has just started a new Masters of Information Systems program, and you need to develop a brochure to help recruit new students.

1. You are in charge. This is a project. You are assigned as a team leader or Project Manager.
   Consider that you are helping put together a team and project.
   Who do you need on your team?

2. Layout a project plan – identifying tasks and participants
   Do not worry about schedule.

3. Class discussion as a whole –
   Did you use predictive or adaptive? What is the difference?
   When would you use predictive, when adaptive? Some combination.

4. Back in groups – rework your schedule, thinking about Phases, Iterations and Disciplines.
   Try to organize tasks somewhat in order (phases)
   Look at schedules again. Think about disciplines and iterations.

Troubleshooting Tips

Use the correct terms and force the students to use the correct terms when they ask questions. Like any discipline, if you do not understand and use the vocabulary, you can never converse intelligently about the subject matter.

Students will frequently have difficulty understanding the differences between a model and a tool. They also may confuse the terms methodology, technique, and approach. This material needs to be carefully presented with clear examples. (Notice we did not use the term “method” because it can cause more confusion between methodology and technique.)

Sometimes students think that Agile development means “code and fix” without any in-depth analysis or design. They also think that Agile development means that it is unnecessary to develop any models, or that modeling does not need to be done correctly. You will need to emphasize the importance of models to capture decisions, understand the requirements, and think thoroughly about solutions.

Discussion Questions

1. The need for a development methodology

Many students will have developed some applications for themselves or for some simple mobile app. Some may even have developed larger systems. In many cases, the students have done this without any type of formal development methodology. You can have an interesting discussion about the importance and usefulness of a development methodology.

If you have developed software, did you use a development methodology? If so what kind of methodology?

If you did not use a development methodology did you do planning, analysis, design, construction? In other words, even though you did not formally identify different activities, did you in fact do them? Would it have been helpful to have a more organized approach? What were the problems you encountered? Did you do enough analysis? Enough design? Enough planning?

If you did use a development methodology, how well was it executed? Was it helpful? Did it cause problems or slow down the development activities? Was there a project manager or leader? Did he/she help or hinder? How much time was spent in overhead activities such as meetings? Was the project formality too much? Too little? A help or a hindrance?

How does the size of the project team affect the need for a development methodology? How does it affect the formality of the methodology? (formal meetings versus around the water cooler meetings).

2. The need for models and modeling

As with the first question, students who have developed software will probably not have used any models or done any modeling. You can have an interesting discussion about the need for models and modeling.

- Is it important to build models? Can you develop a system just as well without any models?
- If you developed a software system did you have problems that could have been resolved more easily with some models? Did you do any rework because you had not completely thought through the problem (analysis) or the solution (design)?
- When is it important to save the models you have developed? Then when is it important to make the models more formal so that they remain? How long should models be maintained or kept?
- Which models might be most helpful? Analysis (understanding) models or Design (solution) models?
- How does the size of the project team affect the modeling requirements?
- How does the size of the solution system affect the need for models?