Chapter 8 – Approaches to System Development

Table of Contents
- Chapter Overview
- Learning Objectives
- Notes on Opening Case and EOC Cases
- Key Terms

Chapter Overview
There are two closely related yet independent concepts in this chapter. The first important concept is that there are two types of Systems Development Life Cycle approaches—a predictive approach and an adaptive approach. The second important concept is that there are two types of development methodologies—a structured approach and an object-oriented approach. These are two separate concepts. Projects can be any mix of these two approaches, the approach to the life cycle and the approach to the methodology—predictive with structured, predictive with object-oriented, adaptive with structured, or adaptive with object-oriented.

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 given 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 other important concept that you should learn from this chapter are the difference between the two main methodologies to system development that are currently used to develop business systems: the structured approach and the object-oriented approach. The object-oriented approach refers to system development using newer object technologies that require a different approach to analysis, design, and programming.

Finally the chapter concludes with a discussion of the Agile philosophy and Agile modeling principles. You 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.

Learning Objectives
After reading this chapter, you should be able to:
- Compare the underlying assumptions and uses of a predictive and an adaptive system development life cycle (SDLC)
- Describe the key activities and tasks of information system support
- Explain what comprises a system development methodology—the SDLC as well as models,
tools, and techniques

- Describe the two overall approaches used for software construction and modeling: the structured approach and the object-oriented approach
- Describe the key features of Agile development

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 development 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 you will encounter various combinations when they begin working in industry. When you 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.

On the Spot Courier EOC Case

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, the authors introduce four subsystems. You are asked to organize an adaptive, iterative project to develop these four subsystems. The questions in the case will help you think about the ramifications of organizing the project in different ways. You are also asked about which models are necessary and to what depth is modeling require based on the Agile principles given in the chapter.

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
- **spiral model** – an adaptive SDLC approach that cycles over and over again through development activities until completion
• **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

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 you 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 8-1 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 8-3. This approach is almost never used any more. It never worked very well. Modified 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 8-4
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. An early version of an adaptive SDLC is called the spiral model, which cycles over and over again throughout development activities and makes adjustments until the project is complete. After each cycle or iteration, a working prototype is available as a preliminary model that shows some aspect of the system that is ready for testing and user evaluation. An iteration is one cycle in the spiral model during which work activities—analysis, design, implementation—are used to complete a version of the working prototype. Completing a project using multiple iterations breaks the problem of system development into more manageable mini-projects.

Other approaches to the adaptive approach are shown in Figures 8-6 and 8-7. Figure 8-7 is an adaptive approach with iterations shown as columns, and the various core processes. The core processes are used to group a set of related activities together, much like a phase does in the predictive approach. In fact, you can think of each iteration as a mini modified waterfall project with a very narrow scope. Within the iteration you will do the various activities of the core processes much like a predictive project.

Related to the idea of an iterative project is the concept of incremental development. Obviously by having a limited scope for each interaction, you 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.

The Support Phase

Key Terms

- **support activities** – the activities in the support phase whose objective is to maintain and enhance the system after it is installed and in use

- **help desk** – the availability of support staff to assist users with technical or processing problems of the information system

Support can be considered as one of the project phases. However, it is more common now a days to consider support as a separate project with its own set of activities. Three major activities occur during support:

- Maintaining the system
- Enhancing the system
Supporting the users

System maintenance are those activities that are required to keep the system working correctly. New systems are usually so complex that it is impossible to test every combination. Hence there are always “bugs” that must be fixed. Many new programmers will start their careers doing system maintenance.

It is also normal for a business to grow and change, which also requires new capabilities to be added to the system. This is referred to as system enhancement. Normally these are done under the auspices of an upgrade or enhancement project. An enhancement project is a project and can be carried out with the six core processes as any other project.

User support is often required in the form of a help desk, or some other mechanism to answer questions, provide training, and verify user identified bugs.

Methodologies, Models, Tools, and Techniques

Key Terms

- **system development methodology** – a set of comprehensive guidelines for the SDLC that includes specific models, tools, and techniques
- **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

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.

**Two Approaches to Software Construction and Modeling**

**Key Terms**

- **structured approach** – system development using structured analysis, structured design, and structured programming techniques
- **structured programming** – a programming approach where each module has one start point and one end point and uses sequence, decision, and repetition constructs only
- **top-down programming** – the concept of dividing a complex program into a hierarchy of program modules
- **structured design** – the design process of organizing a program into a set of modules and organizing those modules into a hierarchical structure
- **structure chart** – a graphical diagram showing the hierarchical organization of modules
- **structured analysis** – a technique to determine what processing is required and to organize those requirements by using structured analysis models
- **data flow diagram (DFD)** – a structured analysis model showing inputs, processes, storage, and outputs of a system
- **object-oriented approach** – system development based on the view that a system is a set of interacting objects that work together
- **object** – a thing in an information system that responds to messages by executing functions or methods
- **object-oriented analysis (OOA)** – the process of identifying and defining the use cases and the sets of objects (classes) in the new system
- **object-oriented design (OOD)** – defining all of the types of objects necessary to communicate with people and devices in the system, showing how objects interact to complete tasks, and refining the definition of each type of object so it can be implemented with a specific language or environment
- **object-oriented programming (OOP)** – programming using object-oriented languages that support object classes, inheritance, reuse, and encapsulation
Earlier in this chapter, the authors discussed approaches to projects, in other words, how a project was organized and executed. Those two approaches, each with several variations, were either predictive or adaptive. In this section they discuss approaches to software development, in other words, how to model and structure the solution software application.

The Structured Approach

**Structured System Development:** Structured analysis, structured design, and structured programming are the three techniques that make up the structured approach. Sometimes, these techniques are collectively referred to as the structured analysis and design technique (SADT).

**Structured programming** produces a program that has one beginning and one ending, with each step in the program execution consisting of one of three programming constructs:

- A sequence of program statements
- A decision point at which one set or another set of statements executes
- A repetition of a set of statements

**Top-down programming** divides more complex programs into a hierarchy of program modules (see Figure 8-13). One module at the top of the hierarchy controls program execution by “calling” lower-level modules as required. The modules and the arrangement of modules are shown graphically by using a model called a structure chart.

There are two main principles of **structured design**: Program modules should be (1) loosely coupled and (2) highly cohesive. Loosely coupled means that each module is as independent of the other modules. Highly cohesive means that each module accomplishes one clear task.

The **structured analysis** technique helps the developer define what the system needs to do (the processing requirements), what data the system needs to store and use (data requirements), what inputs and outputs are needed, and how the functions work together to accomplish tasks. The key graphical model of the system requirements that are used with structured analysis is called the **data flow diagram (DFD)**. A model of the needed data is also created based on the types of things the system needs to store information (data entities) about. The data modeling is done with the entity-relationship diagram.

The Object-Oriented Approach

The **object-oriented approach** views an information system as a collection of interacting objects that work together to accomplish tasks. An **object** is a thing in the computer system that is capable of responding to messages.

Given that the object-oriented approach views information systems as collections of interacting objects, **object-oriented analysis (OOA)** defines the objects that do the work and determines what user interactions (called use cases) are required to complete the tasks. **Object-oriented design (OOD)** defines all the additional types of objects that are necessary to communicate with people and devices in the system, it shows how the objects interact to complete tasks. **Object-oriented programming (OOP)** is the writing of statements in a programming language to define what each type of object does. This is done using “methods,” which are smaller groups of program code. Within a given method
programmers normally use structured programming constructs, of sequence, decision, and repetition.

An object is a type of thing. It could be a customer or an employee or it could be a button or a menu. Identifying types of objects means classifying things. A classification or “class” represents a collection (in reality it is a set) of similar objects; therefore, object-oriented development uses a UML class diagram to show all the classes of objects that are in the system.

The object-oriented approach yields several key benefits, among them naturalness and reusability. The approach is natural—or intuitive—for people because we tend to think about the world in terms of tangible objects. Because the object-oriented approach involves classes of objects and many systems in the organization use the same objects, these classes can be used over and over again whenever they are needed.

**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

**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*

Final Thoughts

You might sometimes get the SDLC approach (predictive/adaptive) mixed up with the software development approach (structured/object-oriented). You will have to work hard and keeping them separate.

It is important that you use the correct terms. Like any discipline, if you do not understand and use the vocabulary, you can never converse intelligently about the subject matter.

You may have difficulty understanding the differences between a model and a tool. Also you may confuse the terms methodology, technique, and approach. Be sure to read the chapter for comprehension and not just skim through it.

Sometimes you might think that Agile development means “code and fix” without any in-depth analysis or design. Also you might think that Agile development means that it is unnecessary to develop any models, or that modeling does not need to be done correctly. It is still very importance to create models to capture decisions, understand the requirements, and think thoroughly about solutions. The key to Agile development is creating the right models. Not too many, not too few.