STUDY GUIDE FOR MODULE NO. 2

CHAPTER 2: SOFTWARE PROCESSES

MODULE OVERVIEW

When we provide a service or create a product we always follow a sequence of steps to accomplish a set of tasks. We do not usually put up the drywall before the wiring for a house is installed or bake a cake before all the ingredients are mixed together. We can think of a series of activities as a **process**. This module will introduce you to the idea of a software process – a coherent set of activities for software production.

Generic software process models will also be introduced in this module to explain different approaches to software development. You can think of these models as process frameworks that may be extended and adapted to create more specific software development processes.

MODULE LEARNING OBJECTIVES

At the end of this chapter, the student should be able to:
1. Understand the concept of a software process and software process models and when they might be used.
2. Understand the process models for software requirements engineering software development, testing and evolution.
3. Understand why processes should be organized to cope with changes in the software requirements and design.

LEARNING CONTENTS (TOPIC 1: SOFTWARE PROCESS)

WHAT IS SOFTWARE PROCESS?

Software Process is a structured set of activities required to develop a software system. There are many different types of software/application, and there is no universal method that is applicable to all of them. Consequently, there is no universally applicable software process.

However, although there are many different software processes, they all must include, in some form, the four (4) fundamental activities of software process. This includes the following:

1. Software Specification – Also known as **Requirements Engineering**. This is the process of establishing what services are required and the constraints on the system's operation and development. The diagram below shows the Requirements Engineering Process.

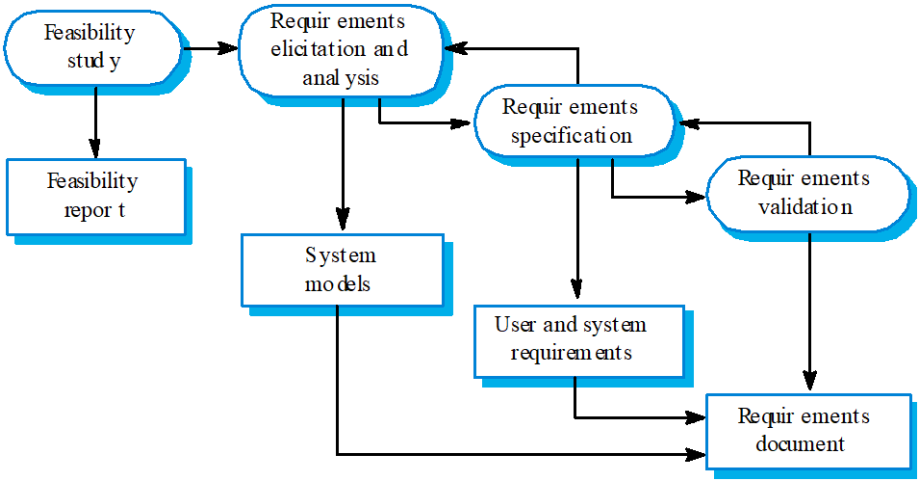

Figure 2.1 Requirements Engineering Process

2. Software Design and Implementation – The implementation stage of software development is the process of converting the system specification into an executable system or the process of developing an executable system for delivery to the customer.

- ***Software design***
  - Design a software structure that realises the specification;
- ***Implementation***
  - Translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved. The diagram below shows the software design process.

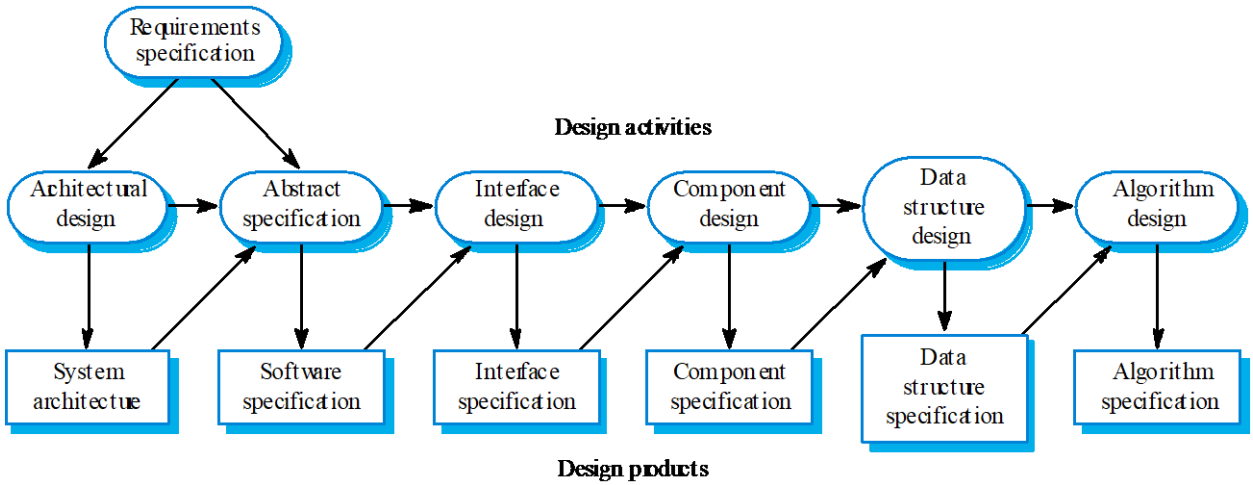

FIgure 2.2 Software Design Process

3. Software Validation or, more generally, Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.

IEEE-STD-610 defines **Verification** as "A test of a system to prove that it meets all its specified requirements at a particular stage of its development." **Validation**, on the other hand, is quite different and serves a very different purpose. The definition of Validation according to IEEE-STD-610 is "An activity that ensures that an end product stakeholder's true needs and expectations are met."

This process involves checking and reviewing processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system. Figure 2.3 shows a three-stage testing process in which system components are individually tested, then the integrated system is tested.

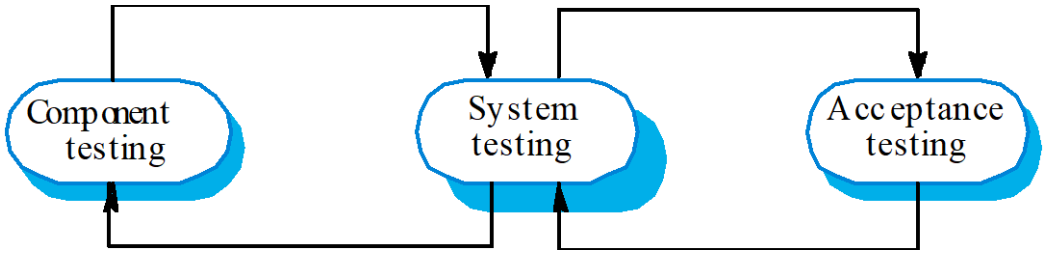

Figure 2.3 The Testing Process

A. Component or Unit Testing

- Individual components are tested independently;
- Components may be functions or objects or coherent groupings of these entities.

B. System Testing

- Testing of the system as a whole. Testing of emergent properties is particularly important.

C. Acceptance testing

- Testing with customer data to check that the system meets the customer's needs.

4. Software Evolution **-** this term refers to the process of developing applications initially, then then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The software must evolve to meet the changing customer needs.

Software or applications are inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. The figure below shows the process of system evolution. Figure 2.8 shows the evolutionary process and stages.

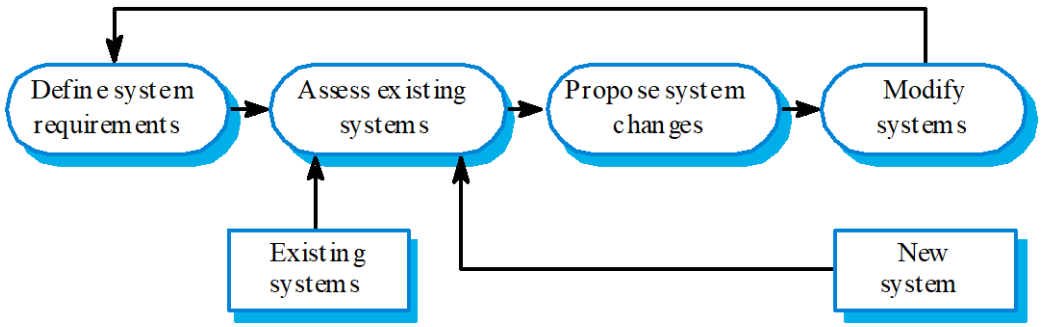

Figure 2.4 System Evolution

Software processes are complex and, like all intellectual and creative processes, rely on people making decisions and judgments. As there is no universal process that is right for all kinds of software, most software companies have developed their own development processes.

LEARNING CONTENTS (TOPIC 2: SOFTWARE PROCESS MODELS)

WHAT IS SOFTWARE PROCESS MODEL?

Software process model (sometimes called as **Software Development Life Cycle or SDLC**) is a simplified representation of a software process. It presents a description of a process from some particular perspective and thus only provides partial information about that process.

Here are some of the general software process models:

1. Waterfall Model – This takes the fundamental process activities of specification, development, validation, and evolution and represents them as a separate process. This is an example of plan-driven process. In principle at least, you plan and schedule all of the process activities before starting software development.

This model presents the software development process as a number of stages, as shown in the figure 2.5. Because of the cascade from one phase to another, this model is known as the waterfall model.

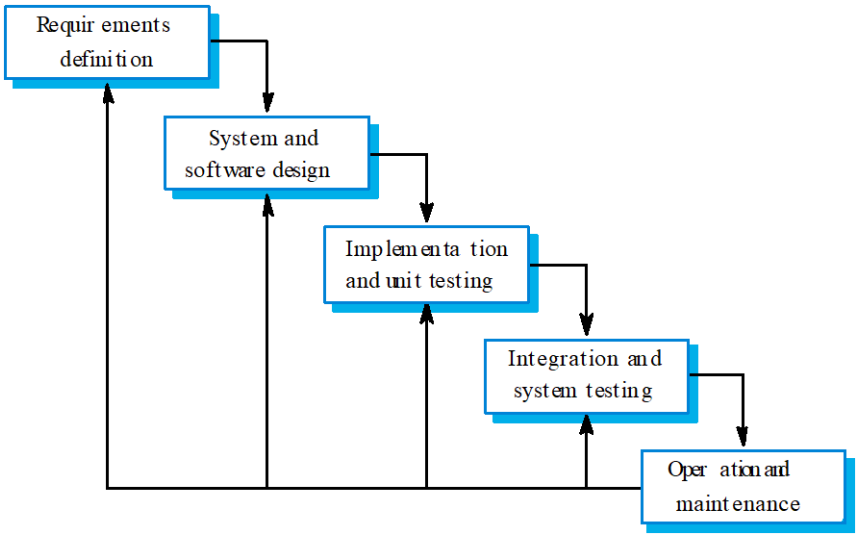

Figure 2.5 Waterfall Model

Waterfall model problems:

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be completed before moving onto the next phase. Its inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

Applicability:

- This model is mostly used for large system/application development projects where a system is developed at several sites.

2. Evolutionary Model - this model is a combination of iterative and incremental model software development life cycle. Delivering an application in a big bang release, delivering it in incremental process over time is the action done in this model.

The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle as shown in the figure 2.6.

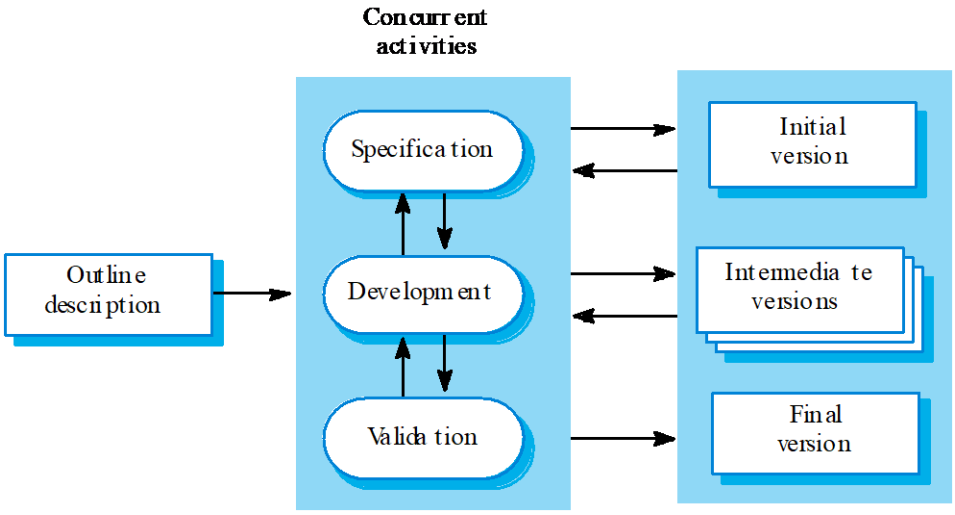

Figure 2.6 Evolutionary Model

Evolutionary Development problems:

- Lack of process visibility;
- Systems are often poorly structured;
- Special skills (e.g. in languages for rapid prototyping) may be required.

Applicability

- For small or medium-size interactive systems;
- For parts of large systems (e.g. the user interface);
- For short-lifetime systems.

3. Reuse-oriented Development - Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems. The system or application is assembled from existing components. This approach is becoming increasingly used as component standards have emerged.

The figure below the process stages of Component-Based Development:

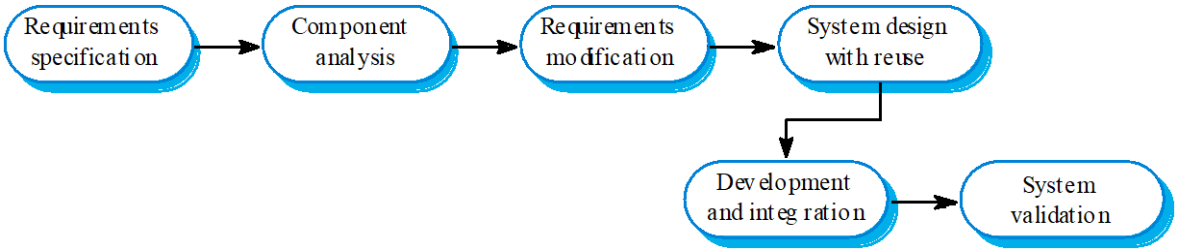

Diagram 2.7 Reuse-Oriented Development

LEARNING CONTENTS (TOPIC 3: COPING WITH CHANGE)

COPING WITH CHANGE

Change is inevitable in all large software projects. The system requirements change as businesses respond to external pressures, competition, and changed management priorities. Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is call rework.

Two related approaches may be used to reduce the costs of rework:

1. *Change Anticipation*, where the software process includes activities that can anticipate or predict possible changes before significant rework is required.

2. *Change Tolerance*, where the process and software are designed so that changes can be easily made to the system. This normally involves some form of incremental development.

There are two ways of coping with the change and changing the system requirements:

1. System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This is a method of change anticipation as it allows users to experiment with the system before delivery and so refine their requirements.

2. Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

LEARNING ACTIVITY 2.1

Individual Activity 2.1

Essay (20 points)

Instruction | Briefly explain the following essay questions:

1. Why processes should be organized to cope with changes in the software requirements and design?

2. If you were to use one of the generic software process models for the development of a safety-critical system, what would it be? Justify your answer.

SUMMARY / KEY POINTS

- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model.
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Requirements engineering is the process of developing a software specification
- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets to its specification and user needs
- Evolution is concerned with modifying the system after it is in use
- Processes should include activities to cope with change. This may involve a prototyping phase that helps avoid poor decisions on requirements and design.

REFERENCES

Sommerville, Ian. Software Engineering, 10th Edition. 2016