Software Engineering

Lecture 2

Software Processes

Dr. Obuhuma James

Description

This topic covers software processes extensively by exploring the software process activities that are packaged into different software process models. It further categorized the various software process models into two broad categories, namely, plan-driven and agile methodologies that will be expounded upon in topic 3 and 4 respectively. Concepts of prototyping, incremental development and delivery and the Rapid Unified Process are also covered as part of modern approaches to software development.

Learning Outcomes

By the end of this topic, you will be able to:

- Describe software development process activities that lead to formation of software process models and their essence in Software Engineering.
- Differentiate between the two broad categories of software process models.
- Appreciate the application of prototyping, incremental development and delivery and the Rapid Unified Process in software development.

Overview of Software Processes

A software process can be viewed as a structured set of activities required to develop a software system. In the first topic, we introduced the key software process activities that are applicable to software development. This topic picks up from there by expounding on these activities and trying to package them into process models.

Software Process Activities

There exist many different software processes. However, they all exhibit the following development elements:

1. Specification

This process activity entails definition of the requirements for the software to be developed and the constraints under which the software will operate. Software specification is normally a joint task done by customers and developers. The main tasks carried out under the activity include feasibility study, requirements elicitation and analysis, requirements specification and requirements validation as summarized in Figure 1.

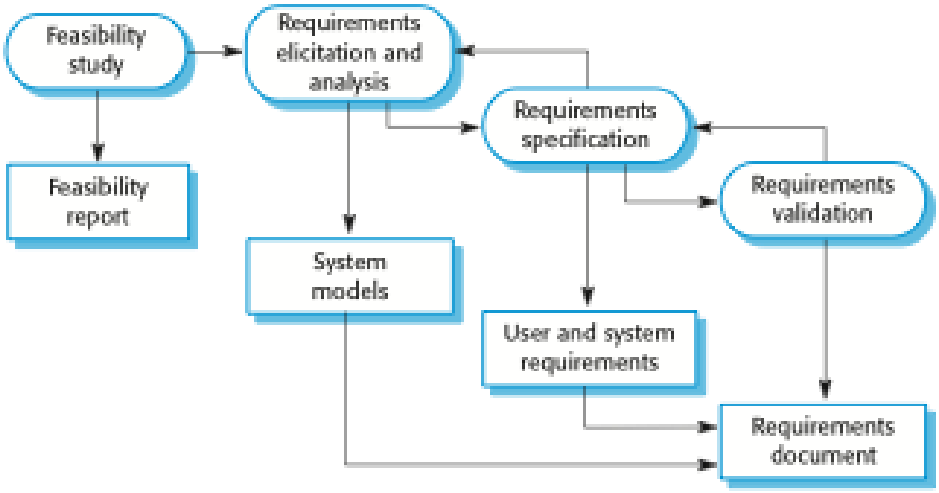

Figure 1. Requirements Engineering Process [1]

2. Design and Implementation

This process activity involves actual software design and programming. It is the point during which the specifications are transformed into an actual system. During software design, the software structure is formulated (designed) in line with the specifications while implementation translates the structure into an executable program. In most cases, the design and implementation activities are normally interleaved into one phase.

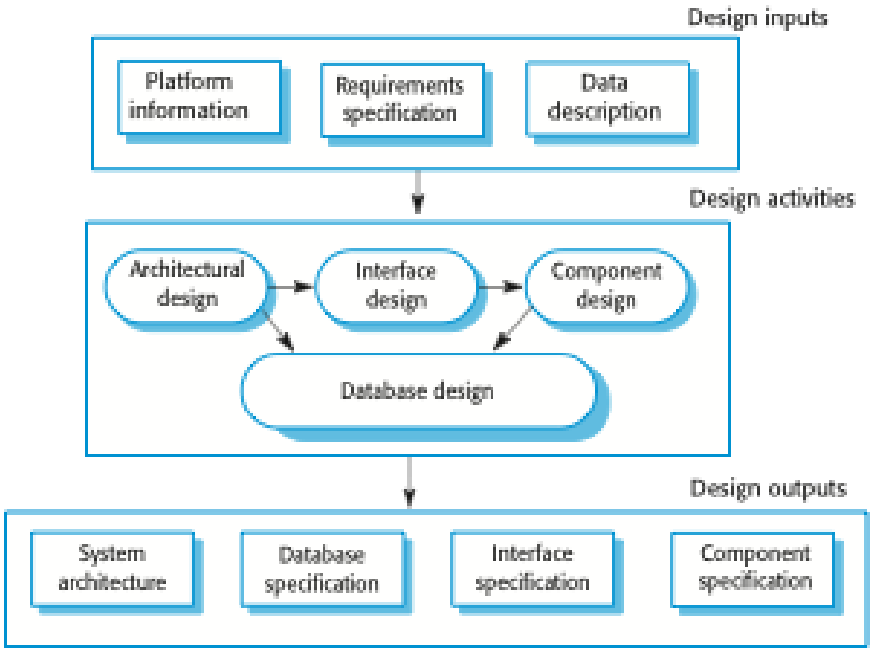

Figure 2. General Model for the Design Process [1]

3. Validation

This process activity gives room for confirmation of whether the developed software addresses customer requirements or not. The software is tested against the requirements specified during the software specification process activity. The testing involves executing the system with real test data based on test cases derived from system specifications. It is normally accomplished in three stages, namely, development or component testing, system testing and acceptance testing. This will be covered in detail later in the course.

4. Maintenance

This process activity applies to the period when the software is in use by the customer. It entails modification of the software to reflect any changes in customer and market needs. It is worth noting that change is inevitable in all large software projects [1]. Why?

- Business changes lead to new and changed system requirements
- New technologies open up new possibilities for improving implementations
- Changing platforms require application changes

Change leads to rework. Thus the cost of change include both the cost of rework as well as the cost of implementing new functionality.

Software Process Models

The software development process activities discussed in the previous section are in most cases normally packaged into what is called a software process model. Such a package reflects an abstract representation of a process. A process model presents a description of a process in some perspective. Software process models are normally defined and formulated by organisations that prefer to define the approach of doing things. Such process models end up being internal organizational processes while others are published and recognized in the public domain. This course will focus on publicly documented process models.

Examples of publicly documented software process models include the Waterfall model, Spiral model, V-model, Incremental development, eXtreme Programming (XP), Feature Driven Development (FDD), Lean Software Development, Scrum, Dynamic Software Development Method (DSDM), among others. For instance, the traditional waterfall model, V-model, and the incremental development take structures shown in Figure 3, 4 and 5 respectively.

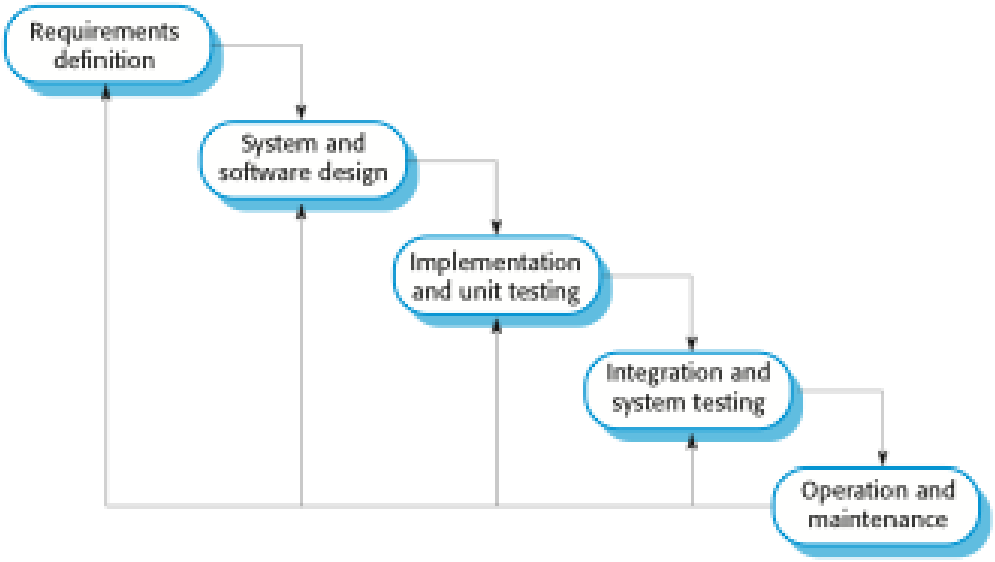

Figure 3. The Waterfall Model [1]

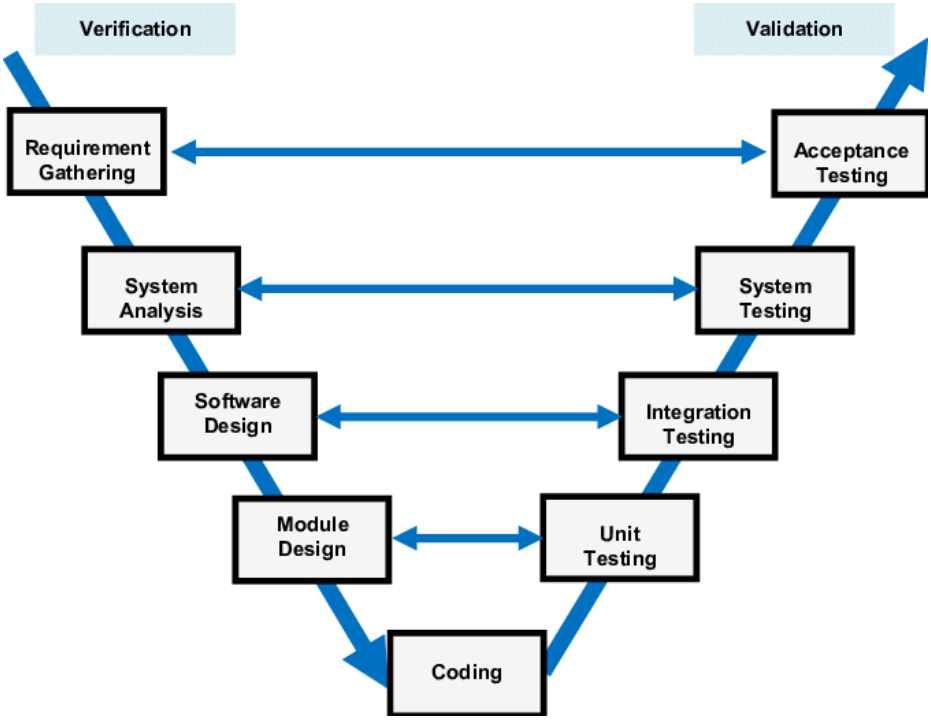

Figure 4. The V-Model

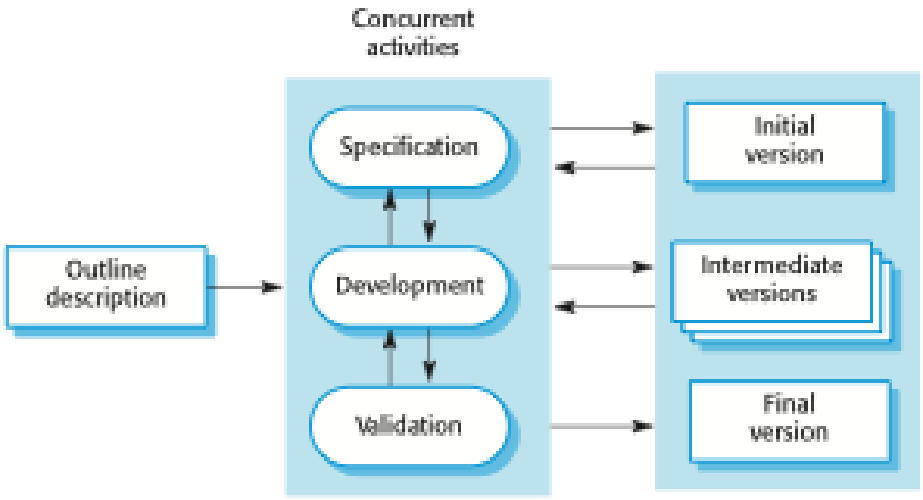

Figure 5. The Incremental Development [1]

Considering the three process models, it is clearly evident that despite the different phases as outlined in each model, each phase activities clearly map to one of the four main software process activities. Table 1 shows the mapping of the phases with the main development process activities.

Table 1. Mapping of Process Models Phases with the Main Software Process Activities

Software Process Activities	**Waterfall Model Phases**	**V-Model Phases**	**Incremental Development**
Specification	▪ Requirements definition	▪ Requirements Gathering ▪ System Analysis	▪ Specification
Design and Implementation	▪ System and software design ▪ Implementation and unit testing	▪ Software Design ▪ Module Design ▪ Coding	▪ Development
Validation	▪ Integration and system testing	▪ Unit Testing ▪ Integration Testing ▪ System Testing ▪ Acceptance Testing	▪ Validation
Maintenance	▪ Operation and maintenance	▪ Comes afterwards	▪ Comes afterwards

Based on Table 1, it is clear that different names are adopted for different process models, but they still map to the main process activities. For instance, the waterfall model splits design and implementation as two separate phases, namely, system and software design and the implementation and unit testing phases. Furthermore, the validation activity is undertaken

during the integration and system testing phase of the waterfall model. On the other hand, the V-model achieves specification through two separate phases, namely, requirements gathering and system analysis; design and implementation through three separate phases, namely, software design, module design, and coding; validation through four separate phases, namely, unit testing, integration testing, system testing and acceptance testing. The maintenance process activity is not represented in the V-model. The same concept applies to the incremental development software process model.

The same approach is applicable to other software process models in existence. The next topic will explore many more software process models with the aim of categorizing them based on the approach used by each model.

Categorisation of Software Process Models

Software process models can be categorized as either plan-driven or agile based on the order or nature of how the various phases (activities) are carried out. This is as summarized in Figure 6 [1].

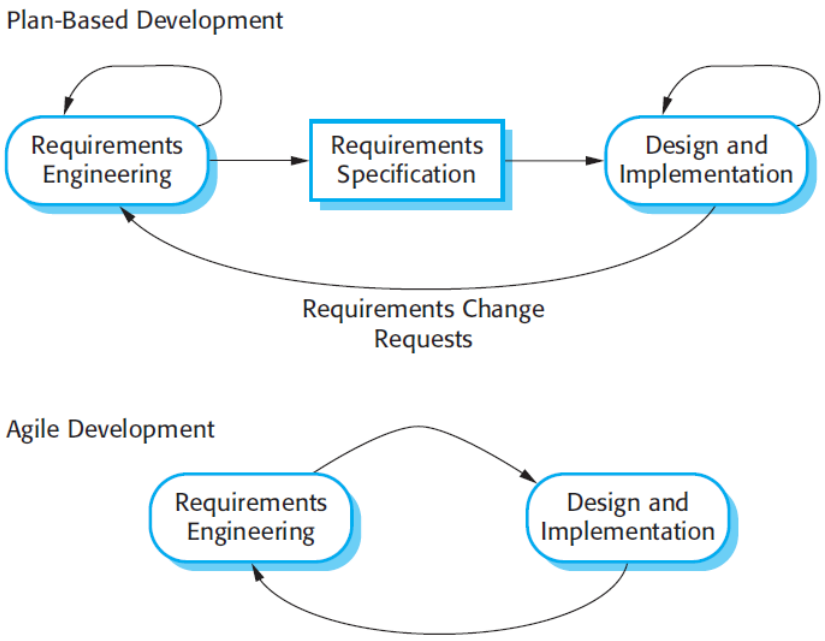

Figure 6. Software Development Methodologies [1]

According to Figure 6, the agile approach considers design and implementation as central activities in the development process which incorporates requirements elicitation and testing, into design and implementation [1]. Contrary, the plan-driven approach preplans separate phases of software development such that outputs from one phase serves as a basis for planning for the next phase [1]. Even though the agile methodology does not follow the phases in a preplanned manner, the critical software development activities of specification, design and implementation, validation and testing are still embraced.

According to Sommerville [1], in practice, most practical processes include elements of both plan-driven and agile methodologies. Such process models are referred to as hybrid models. Thus, there is no right or wrong software process. For instance, in the waterfall model, the basic process activities are organized in sequence while in the incremental development, they are interleaved. Thus, the choice of process model to use may however depend on the nature of the project. The next two topics will expound further on the two methodologies.

Software Prototyping

Prototyping is becoming a famous technique used in modern day software development. A prototype can generally be viewed as an initial visual (or tangible) version of a system used to demonstrate concepts and try out design options. Prototypes are normally shared with customers as a way of soliciting for feedback that could aid in improvement of the desired final product. Figure 7 outlines the process of prototyping as a guide for software engineers.

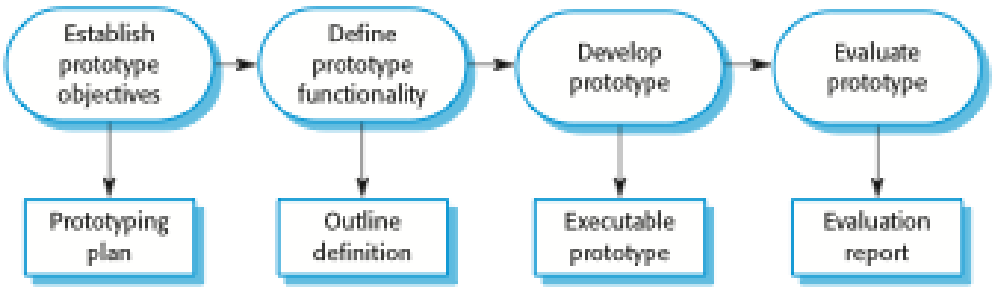

Figure 7. The Process of Prototyping [1]

Prototyping can be used throughout the system development phases to achieve different objectives. This may include:

- In the requirements engineering phase for better requirements elicitation and validation.
- In the design phase as a way of exploring options and developing user interface designs.
- In the validation (testing) phase to get final product feedback from customers. One of the final prototypes always end up being fine-tuned into the final product.

In an ideal case, a product whose development process involves prototyping is guaranteed of the following advantages [1]:

- Ending up as a closer match to users' real needs.
- Achieving improved system usability.
- Better design quality.
- Guarantee of better maintainability.
- Achieving decreased development effort.

Despite the pros of employing prototyping in software development, it may be difficult to try tuning the system (prototype) to meet non-functional requirements. Moreover, prototypes are normally undocumented, and their structure is usually degraded through rapid change [1]. Finally, prototypes possibly won't meet normal organisational quality standards. Thus, the best strategy is to consider using throwaway prototypes that are normally discarded after development, as they are not a good basis for a production system.

Incremental development and delivery

In addition to prototyping, incremental development and delivery is also becoming the norm in the current world of rapid software development. Incremental development entails developing the system in increments as you evaluate each increment together with the user or customer before proceeding to the next increment. On the other hand, incremental delivery entails deployment of increments for use by end-users as a way of achieving a more realistic evaluation based on practical use of software. An increment in this case refers to a

feature or a functionality being developed and released. Figure 8 outlines the activities involved in the process of incremental development and delivery.

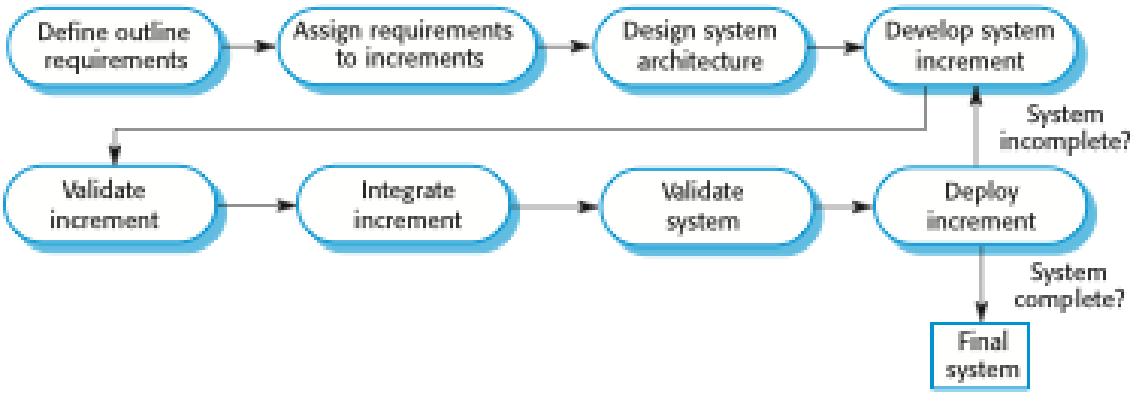

Figure 8. The Process of Incremental Development and Delivery

The use of incremental development and delivery is the prime concept behind the agile methodology as will be discussed later in the course. The use of incremental development and delivery offers the following advantages:

- Customer value can be delivered with each increment. Thus, system functionality is availed earlier in the development cycle.
- Early increments act as prototypes that aid in requirement elicitation for later increments.
- Incremental development and delivery lower the risk of overall project failure.
- Incremental development and delivery give room for additional testing for highest priority system services.

The following challenges are however associated with the use of incremental development and delivery:

- It is sometimes quite difficult to implement for replacement systems since increments have less functionality than the system being replaced.
- The fact that most systems require a set of basic facilities that are used by different parts of the system heavily hinders the application of incremental development and delivery.

- It is sometime difficult to identify common facilities that are needed by all increments since requirements are not defined in detail until a point when an increment is to be implemented.
- The spirit of iterative processes as embraced in incremental development and delivery is that specifications are developed hand in hand with the software. This however conflicts with procurement models of many organizations, where the complete system specification is part of the system development contract. It hence becomes a hindrance factor to the use of incremental development and delivery.

The Rapid Unified Process

The Rapid Unified Process (RUP) is another more modern generic process derived from the work on the UML and associated process. It works as outlined in Figure 9 with incorporation of incremental development and delivery techniques.

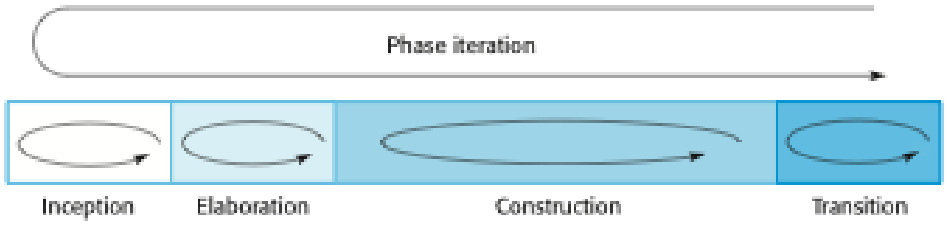

Figure 9. The Rapid Unified Process

The Rapid Unified Process (RUP) has two forms of iterations, namely, in-phase iteration and cross-phase iteration. The in-phase iteration allows each phase to be iterative with results developed incrementally while the cross-phase iteration permits the whole set of phases to be performed incrementally.

Summary

The topic has covered software processes by looking at the software process activities that end up being packaged into software process models. Two broad categories of software process models, namely, plan-driven and agile methodologies, have been introduced as a foundation to the next two topics. Furthermore, concepts of software prototyping, incremental development and delivery and the Rapid Unified Process have also been discussed.

Check Points

1. Discuss the main software process activities.
2. Describe the essence of software process models.
3. Relate software process activities to software process models.
4. State and explain the differences between the two broad categories of software process models.
5. Describe prototyping, incremental development and delivery and the Rapid Unified Process as used in software development.

Learning Resources

Core Textbooks

1. Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

Other Resources

2. Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[3] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

[4] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.