Chapter 3. Software processes

Table of Contents

A software development process, also known as a software development lifecycle, is a structure imposed on the development of a software product. A software process is represented as a set of work phases that is applied to design and build a software product. There is no ideal software process, and many organisations have developed their own approach to software development. Software development processes should make a maximum use of the capabilities of the people in an organisation and the specific characteristics of the systems that are being developed [1] [14] [15].

There are some fundamental activities that are common to all software processes:

1. Software specification. In this activity the functionality of the software and constraints on its operation must be defined.
2. Software design and implementation. The software that meets the specification is produced.
3. Software validation. The software must be validated to ensure that it has all the functionalities what the customer needs.
4. Software evolution. The software must evolve to meet changing customer needs.

3.1. Software process models

A software process model is an abstract representation of a software process. In this section a number of general process models are introduced and they are presented from an architectural viewpoint. These models can be used to explain different approaches to software development. They can be considered as process frameworks that may be extended and adapted to create more specific software engineering processes. In this chapter the following process models will be introduced:

1. The waterfall model. In this model of software process the fundamental process activities of specification, development, validation and evolution are represented as sequential process phases such as requirements specification, software design, implementation, testing and so on.
2. Evolutionary development. This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. Then the initial system is refined by customer inputs to produce a system that satisfies the customer's needs.
3. Component-based software engineering. The process models that use this approach are based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them.

These three generic process models are widely used in current software engineering practice. They are not mutually exclusive and are often used together, especially for large systems development. Sub-systems within a larger system may be developed using different approaches. Therefore, although it is convenient to discuss these models separately, in practice, they are often combined.

3.1.1. The waterfall model

The waterfall model was the first software process model to be introduced (Figure 3.1.). It is also referred to as a linear-sequential life cycle model. The principal stages of the model represent the fundamental development activities:

1. Requirements analysis and definition. Software requirements specification establishes the basis for agreement between customers and contractors or suppliers on what the software product is to do. Software requirements specification permits a rigorous assessment of requirements before design can begin. It should also provide basis for estimating product costs, risks, and schedules.
2. System and software design. Design activity results in the overall software architecture. Software design involves identifying and describing the fundamental software system components and their relationships. The systems design process partitions the requirements to either hardware or software components.
3. Implementation and unit testing. During this phase, the software design is realised as a set of software components. Components are tested ensuring each component meets its specification.
4. Integration and system testing. The program units or components are integrated and tested as a complete system to ensure that the software requirements have been met. After successful testing, the software system is delivered to the customer.
5. Operation and maintenance. The system is delivered and deployed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and providing new functionalities as new requirements emerge.

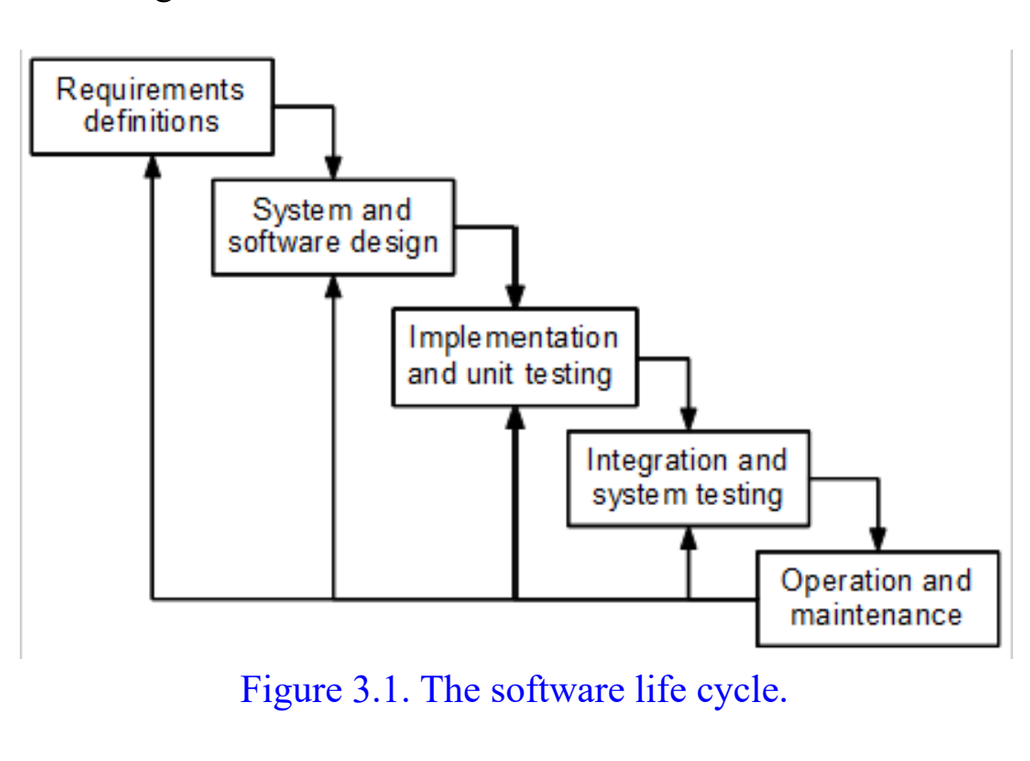

Figure 3.1. The software life cycle.

In principle, the result of each phase is one or more documents. The following phase shall not start until the previous phase has finished. In practice, these stages overlap and feed information to each other. During design, problems with requirements are identified; during coding design problems are found and so on. The software process is not a simple linear model but involves a sequence of iterations of the development activities. Because of the costs of producing and approving documents, iterations are costly and involve significant rework. Therefore, the waterfall model should only be used when the requirements are well understood and unlikely to change significantly during system development.

3.1.1.1. V-model of software process

The V-model represents a software process model that may be considered an extension of the waterfall model. Instead of moving down in a linear way, the process steps are bent upwards after the implementation phase, to form the typical V shape. The V-model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represent time and level of abstraction respectively. Similarly to waterfall model the process steps follow each other in a sequential order but V-model allows the parallel execution of activities.

In the requirements definition phase the requirements of the system are collected by analyzing the needs of the user, and in parallel the user acceptance or functional test cases are also designed. At the level of architectural design the software architecture, its components with their interface are designed at high-level to provide functional requirements of software. The design of integration testing is also carried out in this phase. In the component design phase the units and modules are designed at low-level. The low level design document or program specifications will contain a detailed functional logic of the units and modules. The unit test cases are also developed in this phase. After the implementation phase the development process continues upward by unit, integration and system testing.

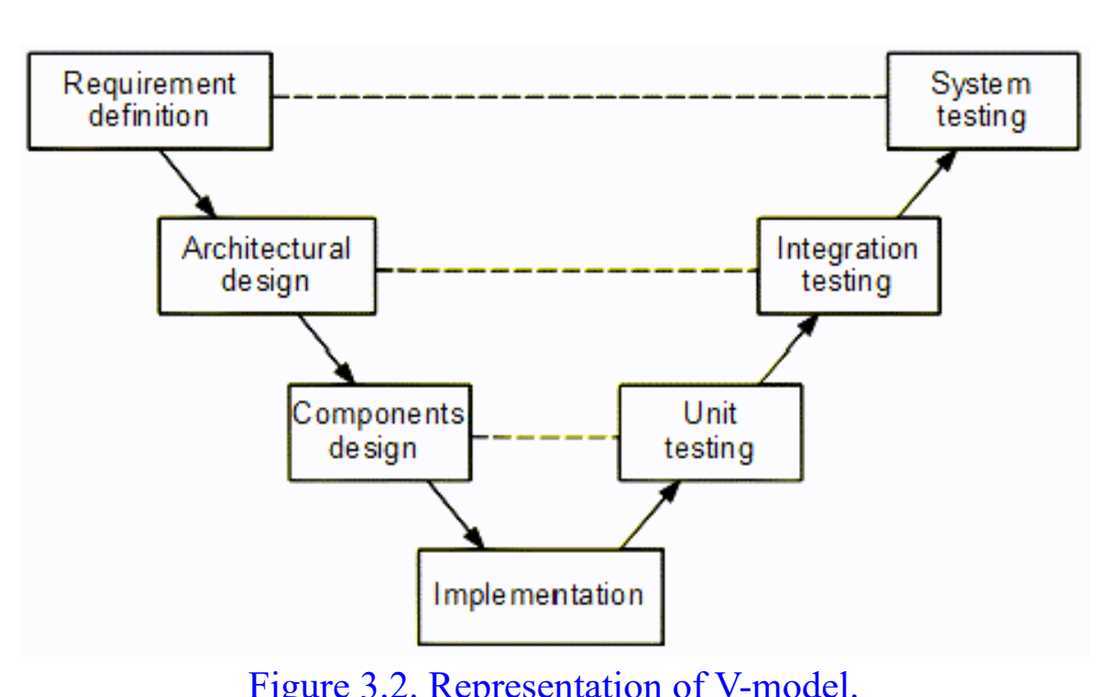

Figure 3.2. Representation of V-model.

3.1.2. Evolutionary development

Evolutionary development is based on the idea of developing an initial implementation, exposing this to user comment and refining it through many versions until an adequate system has been developed (Figure 3.3). Specification, development and validation activities are interleaved with rapid feedback across activities.

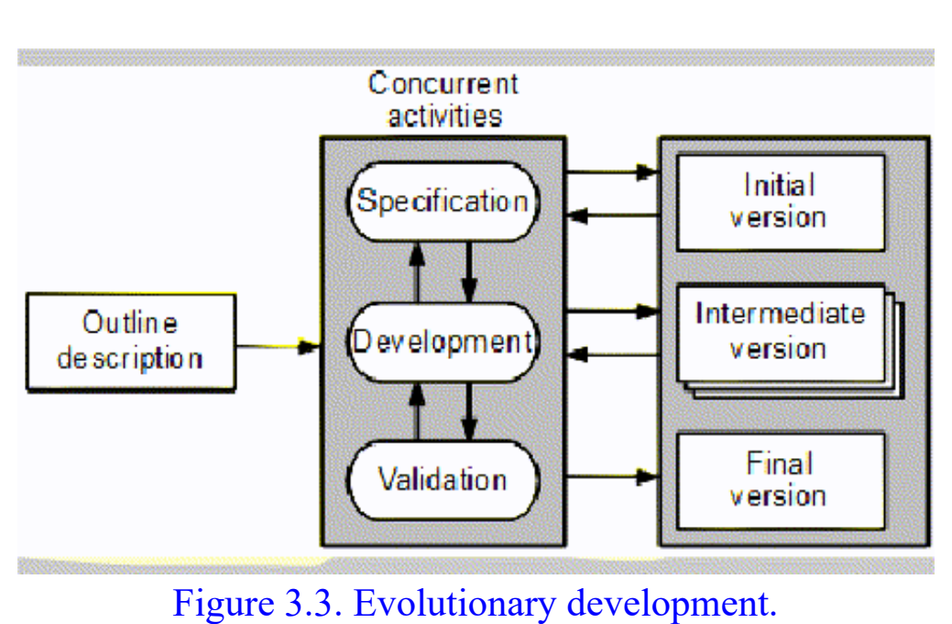

Figure 3.3. Evolutionary development.

There are two fundamental types of evolutionary development:

1. Exploratory development. The objective of the process is to work with the customer in order to explore their requirements and deliver a final system. The development starts with the parts of the system that are well understood. The system evolves by adding new features proposed by the customer.
2. Throwaway prototyping. In this case the objective of the evolutionary development process is to understand the customer's unclear requirements, namely to validate and derive the requirements definition for the system. The prototype concentrates on experimenting with the customer requirements that are poorly understood.

An evolutionary approach to software development is often more effective than the waterfall approach in producing systems that meet the immediate needs of customers. The advantage of a software process that is based on an evolutionary approach is that the specification can be developed incrementally. As users develop a better understanding of their problem, this can be reflected in the software system. However the evolutionary approach has also problems. Regular deliverables are need for monitoring progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system. Continual change tends to corrupt the software structure.

3.1.3. Component-based software engineering

In the majority of software projects, there is some software to reuse. The reusable components are systems that may provide specific functionality for the system. This reuse-oriented approach relies on a large base of reusable software components and some integrating framework for these components. The stages of component-based software process which are different to other processes are the followings:

1. Component analysis. Based on the requirements specification, a search is made for components that can implement the given specification. Usually, there is no exact match, and the components that may be used only provide some of the functionality required.
2. Requirements modification. During this stage, the requirements are analysed using information about the new components. Requirements are then modified to reflect the services of available components.
3. System design with reuse. During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused. Some new software may have to be designed if reusable components are not available.
4. Development and integration. Software that cannot be externally procured is developed, and the components and reusable systems are integrated to create the new system.

Component-based software engineering has the obvious advantage of reducing the amount of software to be developed and so reducing cost and risks. It usually also leads to faster delivery of the software. However, requirements compromises are inevitable and this may lead to a system that does not meet the real (original) needs of users.

3.2. Process iteration

Changes are usually unavoidable in all large software projects. The system requirements change as organization continuously responds to the changing environment and conditions. Management priorities may change. Due to the quick progress in technologies, designs and implementation will change. This means that the process activities are regularly repeated as the system is reworked in response to change requirements. The following two process models have been designed to support process iteration:

1. Incremental delivery. The software specification, design and implementation are broken down into a series of increments that are each developed in turn.
2. Spiral development. The development of the system spirals outwards from an initial outline through to the final developed system.

3.2.1. Incremental delivery

The waterfall model of development requires defining the requirements for a system before design begins. On contrary, an evolutionary development allows requirements to change but it leads to software that may be poorly structured and difficult to understand and maintain.

Incremental delivery (Figure 3.4) is an approach that combines the advantages of these two models. In an incremental development process, customers identify the services to be provided by the software system. They decide which subset of the services is most important and which are least important to them. A number of delivery increments are then defined, with each increment providing a sub-set of the system functionality. The allocation of services to increments depends on the priority of service. The highest priority services are delivered first.

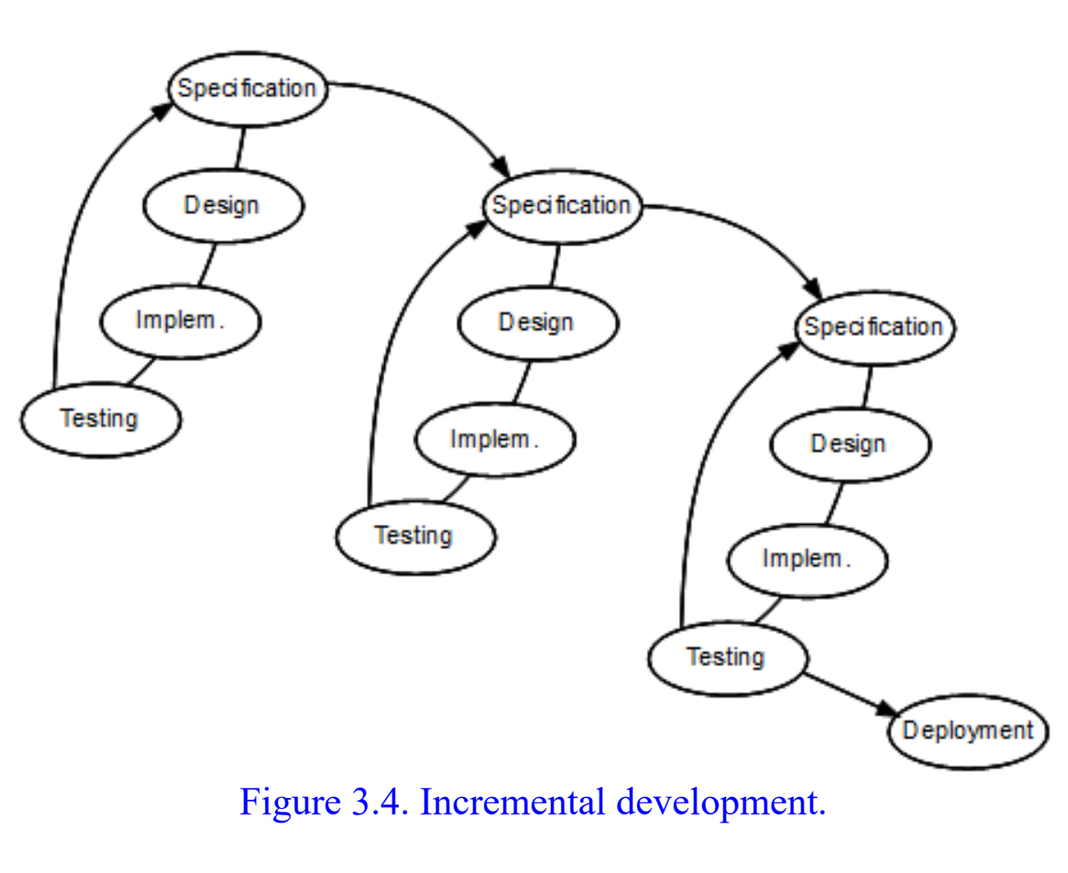

Figure 3.4. Incremental development.

Once the system increments have been identified, the requirements for first increment are defined in detail, and that increment is developed using the best suited software process. In the example given in Figure 3.4. the waterfall model is used to develop the increments in every iteration. As new increments are completed, they are integrated with existing increments and the system functionality improves with each delivered increment. The incremental development process has a number of advantages:

1. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.
2. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments.
3. There is a lower risk of overall project failure. Although problems may be encountered in some increments, it is likely that these will be solved in later versions.
4. As the highest priority services are delivered first, and later increments are integrated with them, it is unavoidable that the most important system services receive the most testing. This means that software failures are less likely to occur in the most important parts of the system.

3.2.2. Spiral development

The spiral model of the software process (Figure 3.5.) represents the software process as a sequence of activities with some backtracking from one activity to another, the process is represented as a spiral. Each loop in the spiral represents a phase of the software process. Thus, the innermost loop might be concerned with system feasibility, the next loop with requirements definition, the next loop with system design and so on.

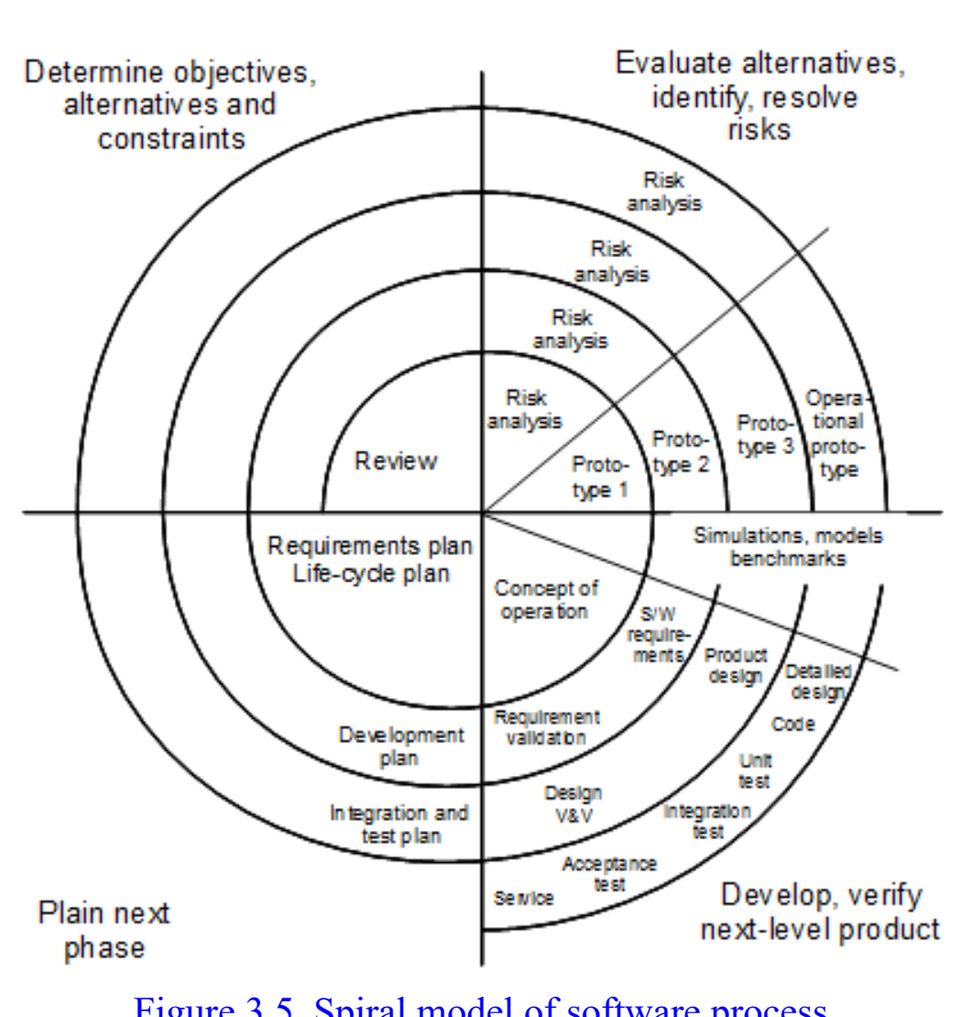

Figure 3.5. Spiral model of software process.

Each loop in the spiral is split into four sectors:

1. Objective setting. Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
2. Risk assessment and reduction. For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk.
3. Development and validation. After risk evaluation, a development model for the system is chosen.
4. Planning. The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

The main difference between the spiral model and other software process models is the explicit recognition and handling of risk in the spiral model.

3.3. Process activities

The four basic process activities of specification, development, validation and evolution are organised differently in different development processes. In the waterfall model, they are organised in sequence, whereas in evolutionary development they are interleaved. How these activities are carried out depends on the type of software, people and organisational structures involved.

3.3.1. Software specification

A software requirement is defined as a condition to which a system must comply. Software specification or requirements management is the process of understanding and defining what functional and non-functional requirements are required for the system and identifying the constraints on the system's operation and development. The requirements engineering process results in the production of a software requirements document that is the specification for the system.

There are four main phases in the requirements engineering process:

1. Feasibility study. In this study an estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-effective from a business point of view and whether it can be developed within existing budgetary constraints.
2. Requirements elicitation and analysis. This is the process of deriving the system requirements through observation of existing systems, discussions with potential users, requirements workshop, storyboarding, etc.
3. Requirements specification. This is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document: user (functional) requirements and system (non-functional) requirements.
4. Requirements validation. It is determined whether the requirements defined are complete. This activity also checks the requirements for consistency.

3.3.2. Software design and implementation

The implementation phase of software development is the process of converting a system specification into an executable system through the design of system. A software design is a description of the architecture of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms used. The design process activities are the followings:

1. Architectural design. The sub-systems of system and their relationships are identified based on the main functional requirements of software.
2. Abstract specification. For each sub-system, an abstract specification of its services and the constraints under which it must operate is defined.
3. Interface design. Interfaces allow the sub-system' services to be used by other sub-systems. The representation of interface should be hidden. In this activity the interface is designed and documented for each sub-system. The specification of interface must be unambiguous.
4. Component design. Services are allocated to components and the interfaces of these components are designed.
5. Data structure design. The data structures used in the system implementation are designed in detail and specified.
6. Algorithm design. In this activity the algorithms used to provide services are designed in detail and specified.

This general model of the design process may be adapted in different ways in the practical uses.

A contrasting approach can be used by structured methods for design objectives. A structured method includes a design process model, notations to represent the design, report formats, rules and design guidelines. Most these methods represent the system by graphical models and many cases can automatically generate program code from these models. Various competing methods to support object-oriented design were proposed in the 1990s and these were unified to create the Unified Modeling Language (UML) and the associated unified design process.

3.3.3. Software validation

Software validation or, more generally, verification and validation (V & V) is intended to show that a system conforms to its specification and that the system meets the expectations of the customer buying the system. It involves checking the processes at each stage of the software process. The majority of validation costs are incurred after implementation when the operation of system is tested.

The software is tested in the usual three-stage testing process. The system components, the integrated system and finally the entire system are tested. Component defects are generally discovered early in the process and the interface problems during the system integration. The stages in the testing process are:

1. Component (or unit) testing. Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.
2. System testing. The components are integrated to make up the system. This testing process is concerned with finding errors that result from interactions between components and component interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements.
3. Acceptance testing. It is considered a functional testing of system. The system is tested with data supplied by the system customer.

Usually, component development and testing are interleaved. Programmers make up their own test data and test the code as it is developed. However in many process model, such as in V-model, Test Driven Development, Extreme Programming, etc., the design of the test cases starts before the implementation phase of development. If an incremental approach to development is used, each increment should be tested as it is developed, with these tests based on the requirements for that increment.

3.3.4. Software evolution

Software evolution, specifically software maintenance, is the term used in software engineering to refer to the process of developing software initially, then repeatedly updating it for various reasons.

The aim of software evolution would be to implement the possible major changes to the system. The existing larger system is never complete and continues to evolve. As it evolves, the complexity of the system will grow. The main objectives of software evolution are ensuring the reliability and flexibility of the system. The costs of maintenance are often several times the initial development costs of software.

3.4. The Rational Unified Process

The Rational Unified Process (RUP) methodology is an example of a modern software process model that has been derived from the UML and the associated Unified Software Development Process. The RUP recognises that conventional process models present a single view of the process. In contrast, the RUP is described from three perspectives [3] [11] [17] [22]:

1. A dynamic perspective that shows the phases of the model over time.
2. A static perspective that shows the process activities.
3. A practice perspective that suggests good practices to be used during the process.

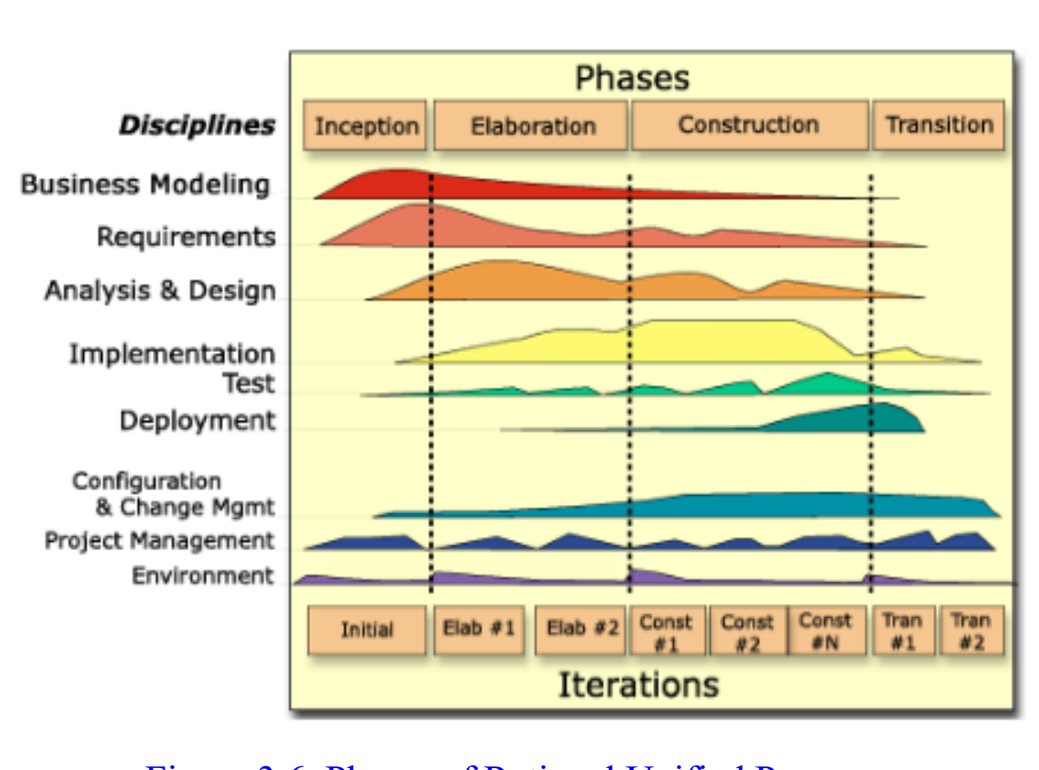

Figure 3.6. Phases of Rational Unified Process.

The RUP (Figure 3.6.) identifies four discrete development phases in the software process that are not equated with process activities. The phases in the RUP are more closely related to business rather than technical concerns. These phases in the RUP are:

1. Inception. The goal of the inception phase is to establish a business case for the system, identify all external entities, i.e. people and systems that will interact with the system and define these interactions. Then this information is used to assess the contribution that the system makes to the business.
2. Elaboration. The goals of the elaboration phase are to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan and identify key project risks.
3. Construction. The construction phase is essentially concerned with system design, programming and testing.
4. Transition. The final phase of the RUP is concerned with moving the system from the development community to the user community and making it work in a real environment.

Iteration within the RUP is supported in two ways. Each phase may be enacted in an iterative way with the results developed incrementally. In addition, the whole set of phases may also be enacted incrementally.

The static view of the RUP focuses on the activities that take place during the development process. These are called workflows in the RUP description. There are six core process workflows identified in the process and three core supporting workflows. The RUP has been designed in conjunction with the UML so the workflow description is oriented around associated UML models. The core engineering and support workflows are the followings:

1. Business modelling. The business processes are modelled using business use cases.
2. Requirements. Actors who interact with the system are identified and use cases are developed to model the system requirements.
3. Analysis and design. A design model is created and documented using architectural models, component models, object models and sequence models.
4. Implementation. The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
5. Testing. Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
6. Deployment. A product release is created, distributed to users and installed in their workplace.
7. Configuration and change management. This supporting workflow manages changes to the system.
8. Project management. This supporting workflow manages the system development.
9. Environment. This workflow is concerned with making appropriate software tools available to the software development team.

The advantage in presenting dynamic and static views is that phases of the development process are not associated with specific workflows. In principle at least, all of the RUP workflows may be active at all stages of the process. Of course, most effort will probably be spent on workflows such as business modelling and requirements at the early phases of the process and in testing and deployment in the later phases.

The practice perspective on the RUP describes good software engineering practices that are recommended for use in systems development. Six fundamental best practices are recommended:

1. Develop software iteratively. Plan increments of the system based on customer priorities and develop and deliver the highest priority system features early in the development process.
2. Manage requirements. Explicitly document the customer's requirements and keep track of changes to these requirements. Analyze the impact of changes on the system before accepting them.
3. Use component-based architectures. Structure the system architecture into components.
4. Visually model software. Use graphical UML models to present static and dynamic views of the software.
5. Verify software quality. Ensure that the software meets the organisational quality standards.
6. Control changes to software. Manage changes to the software using a change management system and configuration management procedures and tools.

The RUP is not a suitable process for all types of development but it does represent a new generation of generic processes. The most important innovations are the separation of phases and workflows, and the recognition that deploying software in a user's environment is part of the process. Phases are dynamic and have goals. Workflows are static and are technical activities that are not associated with a single phase but may be used throughout the development to achieve the goals of each phase.

3.5. Exercises

1. What software development project does waterfall model recommended for?
2. What software development project does waterfall model not recommended for?
3. List application problems related to waterfall development!
4. What is the meaning of requirements validation?
5. What are the main activities of software design phase?
6. What is the different between the system and acceptance testing?
7. Explain the role of prototypes in the evolutionary development!
8. Explain the reason for software developed by an evolutionary development are often more difficult to maintain!
9. What are the support workflows in RUP?
10. How does RUP support the iterative software process?