Hyperiondev

TASK

The Software Process

Visit our website

Introduction

Welcome to the Software Process Task!

As a budding software engineer, you are probably eager to dive into developing your own software. However, when creating a software system, it isn't advisable to jump in and start coding. If you would like to produce good software that has all the required functionality in a reasonable time frame, you need to follow a software development process. This task will introduce you to the idea of a software process. It will discuss the concepts of software development processes and the software process models.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

What is a Software Process?

A software process is a set of related activities that leads to the production of a software product. These activities may involve developing software entirely from scratch. However, most business applications are not developed this way. New business software is usually developed by modifying or extending already existing systems or by integrating off-the-shelf software. There are numerous different software processes, but they all must include the following fundamental activities:

1. **Software specification:** The functionality of the software and the constraints on its operation are defined.
2. **Software design and implementation:** The software is designed and programmed to meet the specification.
3. **Software validation:** The software is validated to ensure that it does what the customer wants.
4. **Software evolution:** The software should evolve to meet the changing needs of the customer. The development of software is not finished with deployment.

Software processes are complex and there is no such thing as an ideal process. Most organisations have developed their own software development processes. To take advantage of people's capabilities in an organisation and the specific characteristics of the system being developed, processes have to evolve.

Software Process Models

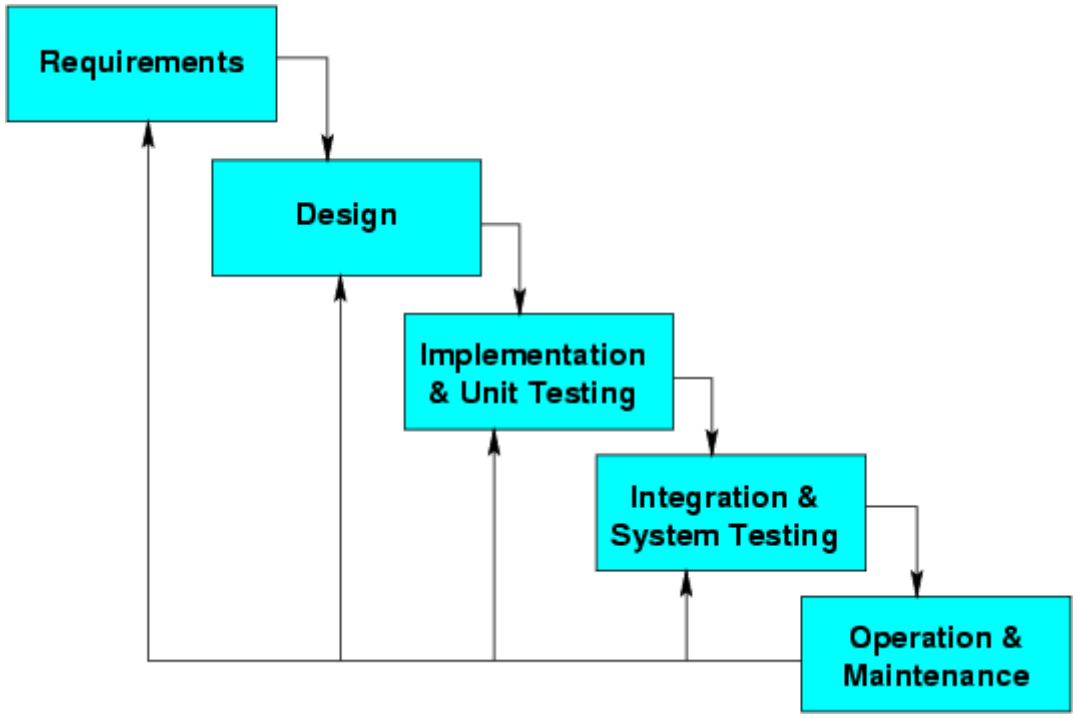
A software process model is a simple representation of a software process. Each process model only provides partial information about the process it represents. We will cover three general process models. These models are not detailed descriptions of software processes but rather a summary of the process.

The process models that we will cover in this task are:

1. The waterfall model
2. Incremental development
3. Reuse-oriented software engineering

The Waterfall Model

The waterfall model is very simple to understand and use. It takes the fundamental activities of specification, development, validation and evolution and separates them into phases, which cascades from one phase to another — hence the name. The waterfall model illustrates the development process in a linear sequential way, meaning the next phase in the process will only begin once the previous phase is complete. With the waterfall model, you must plan and schedule all of the process activities before starting work on them.



The Waterfall Model (cs.odu.edu)

The diagram above shows the different phases of the waterfall model. We will now briefly discuss each phase:

1. *Requirements analysis and definition:* The system users are consulted to establish the system's services, constraints and goals, which are then defined in detail and documented.

2. *System and software design:* The system design helps in specifying hardware and software requirements. It also helps in defining the overall system architecture. Software design provides everything that software

engineers need to know to produce software that implements the required functionality.

3. *Implementation and unit testing:* The software design is realised as a set of program units in this stage. Unit testing verifies that each unit meets the specification.

4. *Integration and system testing:* All program units are integrated and then tested as a complete system to ensure all of the software requirements have been met. The software system is then delivered to the customer.

5. *Operation and Maintenance:* This is often the longest phase. During this stage, the system is installed and put into practical use. Maintenance involves correcting errors that were not picked up in the previous stages and enhancing and improving the system.

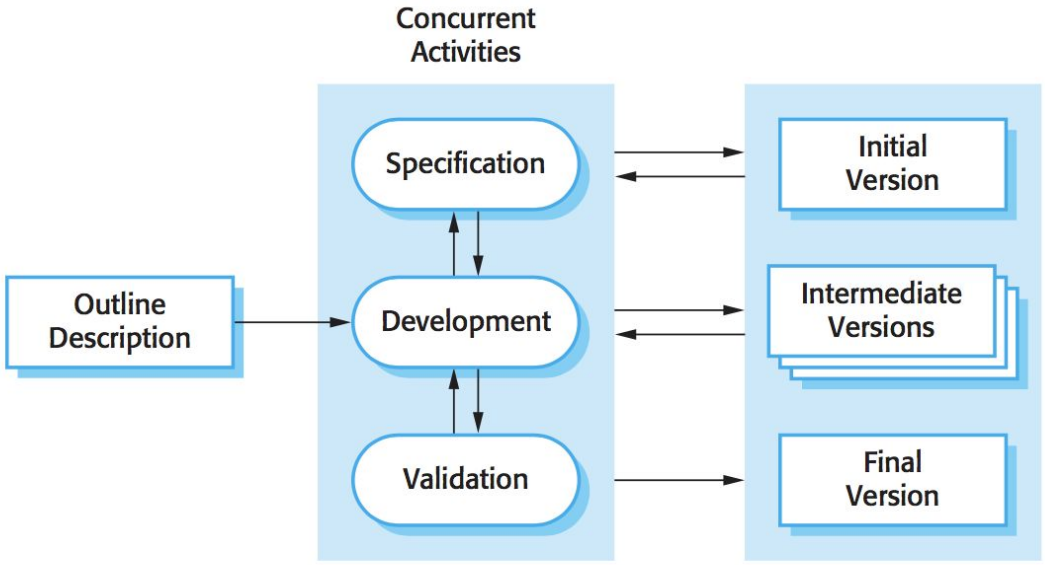
Even though the following phase should not start until the previous phase is finished, in practice, these stages overlap and feed information to each other. For example, during the design phase, problems with requirements can be identified. The software process normally involves feedback from one phase to the other and is not a simple linear model.

A major disadvantage of the waterfall model is its inflexibility. Since you must complete each phase before moving on to the next, commitments must be made at an early stage in the process, which makes it difficult if the customer's requirements change. Therefore, the waterfall model should only be used when the requirements are well understood and unlikely to change during system development. Examples of when this could be practical include when developing safety-critical or security-critical systems.

Incremental Development

The incremental development approach interleaves the fundamental activities of specification, development and validation. The system is developed as a series of versions, or increments, where each version adds more functionality to the previous version.

With incremental development, an initial version of the system is given to the user for feedback. Feedback from the user is used to evolve the system through several versions until an adequate system has been developed.



Incremental Development (Sommerville 2010)

The diagram above illustrates the incremental development model. As you can see the specification, development and validation activities occur at the same time, and not separately like the waterfall model, with rapid feedback across activities.

Incremental development is very similar to how we naturally solve problems. We very seldom work out the whole solution to a problem in advance. Rather, we work towards the solution in a series of steps and backtrack if we make a mistake. Developing software incrementally is cheaper and it is easier to make changes to the software as it is being developed.

Some functionality that is needed by the customer is incorporated into each version of the software. The first versions of the software generally includes the core, most important or most urgently required functionality. Therefore, the customer can test the system at an early stage of development to see if it delivers the most important requirements. Only the current increment has to be changed if it does not deliver what is required.

Three benefits differentiate incremental development from the waterfall model:

1. It is cheaper to change the system if the customer's requirements change.

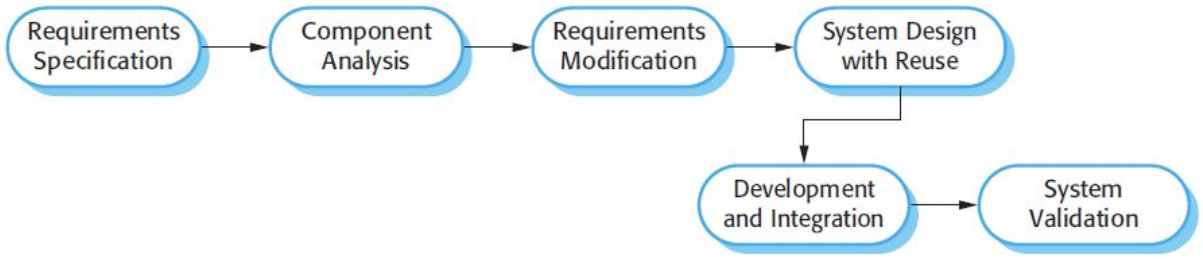
2. It is easier to get feedback from the customers regarding the work that has been done on the system.
3. Useful software is delivered and deployed more rapidly to the customer, even if all the functionality has not been included.

However, a disadvantage of incremental development is that it is not visible. Documentation is needed to measure the progress of development. However, if the system is developed too quickly, it is not cost-effective to produce documents that reflect all versions of the system. Another disadvantage is that as new increments are added, the structure of the system tends to degrade and a lot of time and money will be needed to improve the software.

Large, complex systems compound the problem faced by incremental development. These systems, which usually have a long life span, are worked on by many different teams of engineers. The responsibilities of the various teams need to be clearly defined concerning architecture since large systems require a stable architecture. This has to be planned in advance and not developed incrementally.

Reuse-Oriented Software Engineering

Almost all software projects have some software reuse. This often happens informally. Often, people know of code that is similar to what is required that they then modify and incorporate into the system.



Reuse-Oriented Software Engineering (*Sommerville 2010*)

Above is the reuse-oriented development process model. The requirements specification stage and the validation stage are like other software processes. However, other stages are not.

Below are the stages of the reuse-oriented software development process:

1. *Component analysis:* Components to implement the requirements specification are found. You do not usually find an exact match and the components that you find only provide partial functionality.

2. *Requirements modification:* Using information about the components that have been found, the requirements are analysed. They are then modified to reflect the available components. If no modifications can be made, you re-enter the component analysis activity to search for alternative solutions.

3. *System design with reuse:* The framework of the system is designed or an existing framework is reused. It is organised to cater for components that are reused. If reusable components are not available, some new software may have to be designed.

4. *Development and integration:* Software that cannot be found is developed, and the components and commercial-off-the-shelf (COTS) systems are integrated to create the new system. In this model, system integration is part of the development process and not a separate activity.

In a reuse-oriented process, three types of software components may be used:

1. Web services developed according to service standards.
2. Collections of objects that are developed as a package to be integrated with a component framework.
3. Stand-alone software systems configured for use in a particular environment.

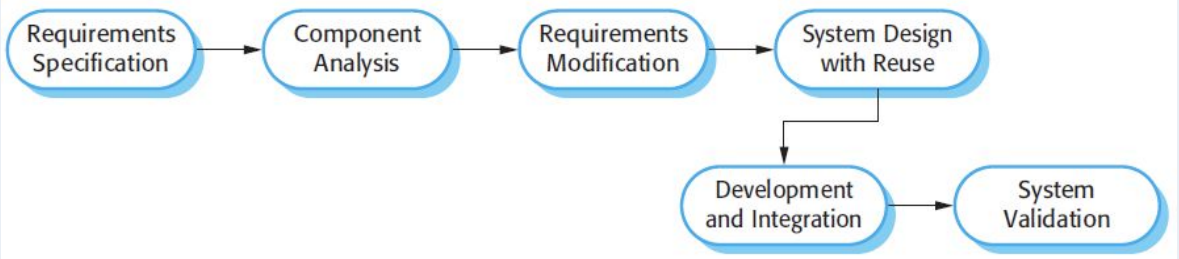
The reuse-oriented development process reduces the amount of software that you need to develop and, therefore, reduces cost and risks. It enables faster delivery of the software. However, you don't always find the exact component that you need and requirement compromises might need to be made. This may lead to a system that does not meet all the needs of the users. Control over the system evolution is also lost as new versions of the reusable components are not under the control of the organisation using them.

Compulsory Task

- Create a file called **answers.txt**. Answer the following questions in **answers.txt**:

1. Suggest which generic software model will be most appropriate for the development of the following systems. Give reasons for your answer.
    a. A system to control anti-lock braking in a car
    b. A virtual reality system to support software maintenance
    c. A university accounting system that replaces an existing system
    d. An interactive travel planning system that helps users plan journeys with the lowest environmental impact

2. Why do you think incremental development is the most effective approach for developing business software systems and less appropriate for real-time systems engineering?



3. Look at the diagram showing reuse-oriented software engineering above. Why do you think it is necessary to have two separate requirements engineering activities in this process?

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.