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 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?

What is involved in the process of making tea? You boil the kettle, take a cup out of the cupboard, and put the teabag in the cup. Once the water is boiled, you add it to the cup, then add milk and sugar, stir, and take the teabag out. Voila! Tea!

You'll notice that each activity leads to our desired goal of having a cup of tea. The same is true of a software process: it is a set of steps and activities that need to be completed, which will lead to a final end product. There is a myriad of potential software processes to choose from — each with their own advantages and disadvantages. Some companies even develop their own custom software processes to deal with their specific needs. However, all contain these activities (Sommerville, 2011):

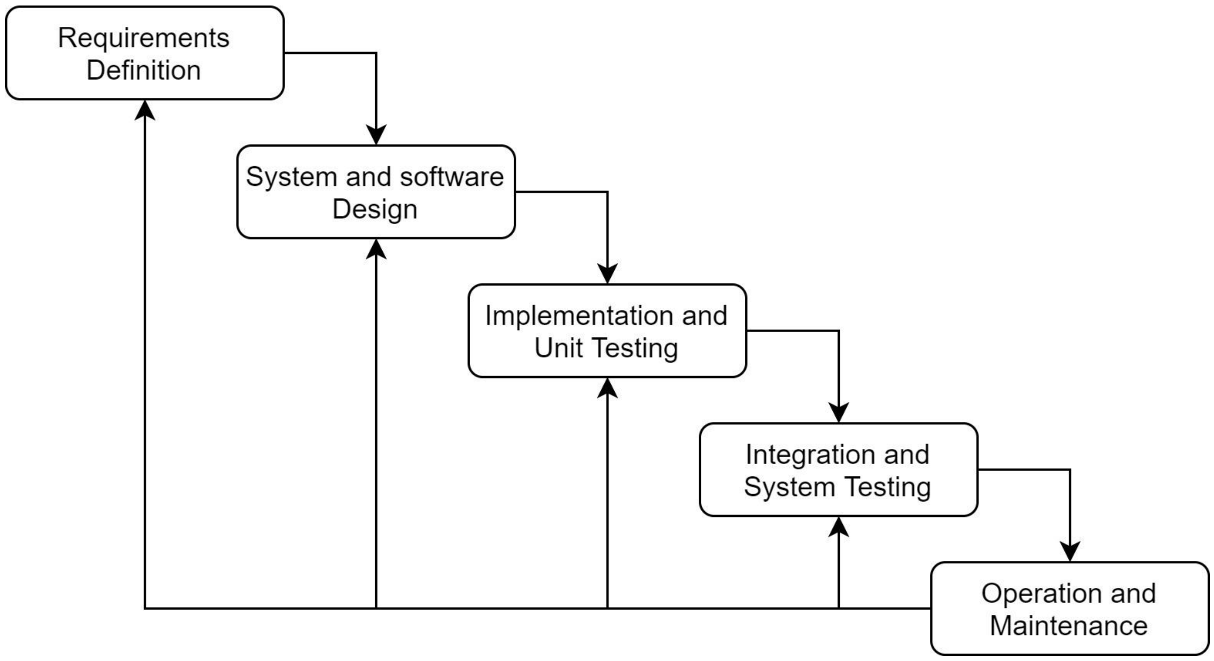
1. **Software specification:** The functionality of the software and the constraints on its operation are defined. Simply put, software specifications define what the program will do and what it won't do. These include the requirements discussed in the previous task.
2. **Software design and implementation:** 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 PROCESS MODELS

A model is used to represent the software process. to help someone understand how a particular concept or process works. The model is often in the form of a diagram or flowchart. In this task, we will be looking at the **Waterfall Model**, the **Incremental Development Model**, and the **Reuse-Oriented Software Engineering Model**. You will notice that each model begins specifying what the software is required to accomplish. This is where the previous task's requirements specification fits in.

THE WATERFALL MODEL

The waterfall model is probably the most intuitive model of the three: just as a waterfall flows downwards, the activities in this model flow from one to the other. The golden standard of this model is that you cannot move onto the next activity until the current one is completed (you can't pour the boiling water into a cup until you've taken your cup out the cupboard!). This means that planning is vital — before you start the process, you need to lay out clearly the plan of which activities will be done when. This makes the Waterfall Model the ideal model for software that focuses on safety or security.


Requirements Definition
System and software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

The Waterfall Model (adapted from Sommerville, 2011, p. 30)

However, it is not always practical for the process to flow only in one direction. As you can see from the arrows in the diagram above, the activities often overlap and backflow — for example, testing could lead to a change in the software design. The problem with this, however, is that due to this model's rigidity, changes can be expensive and time-consuming. This is because the need for change is often only noticed after implementation (when the user is unhappy with the product), which means that a great deal of work might need to be re-done.

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 with specifying hardware and software requirements. It also helps with 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 units. Unit testing verifies that each unit meets the specification.

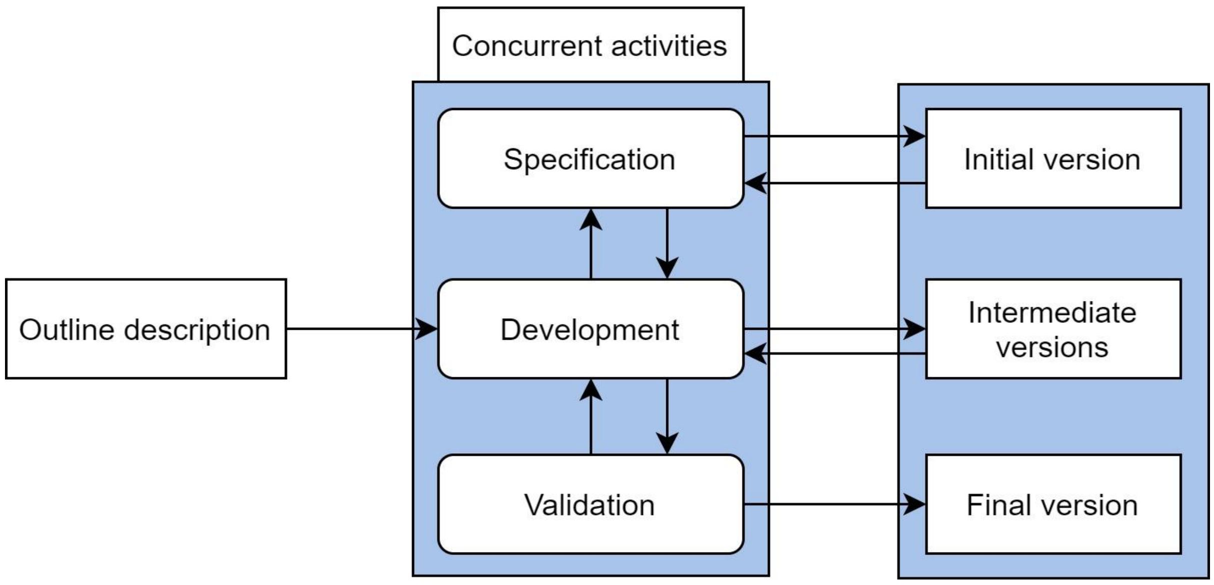
4. **Integration and system testing:** All program units are integrated and then tested as a complete system to ensure all of the software requirements have been met. The software system is then delivered to the customer.

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.

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 Model (adapted from Sommerville, 2011, p. 33)

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 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 include the most important or most urgently required functionalities. 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.

According to Sommerville (2011), the three benefits that differentiate incremental development from the waterfall model include:

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, disadvantages of this model include:

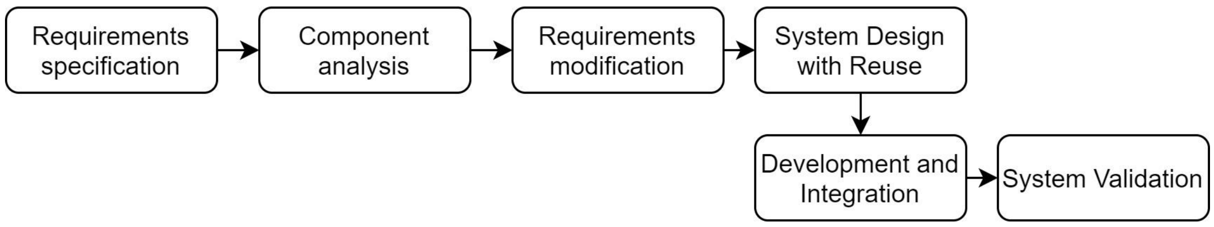
1. It is difficult to have accurate documentation that reflects all versions of the system because there are so many different versions.

2. The structure of the system tends to degrade as new systems are added.

3. Incremental systems can become large and complex due to the nature of the model. Therefore, it is key to create a stable architecture right at the beginning through well-defined team roles before things become complicated (you will learn more about system architecture in the next level). A major risk with incremental development is scope creep, where adding a single small extra feature at every cycle eventually leads to a huge number of 'small extra features' that didn't need to be added. Because they were added in later, they might not be properly integrated into the designs or documents, which eventually causes the whole thing to turn into a house of cards no-one understands properly.

REUSE-ORIENTED SOFTWARE ENGINEERING

Sometimes it's not necessary to reinvent the wheel. Oftentimes, we can use code that already exists in a new programme. This can be done informally (e.g. code that has been written for the business) or formally (e.g. commercial-off-the-shelf (COTS) systems). In this model, that existing code is then taken and adapted to fit with the requirements of the project.


Requirements specification
Component analysis
Requirements modification
System Design with Reuse
Development and Integration
System Validation

The Reuse-Oriented Software Engineering Model (adapted from Sommerville, 2011, p. 35)

The diagram above shows the different phases of the Reuse-Oriented Software Engineering Model. We will now briefly discuss each phase:

1. **Requirements specification:** as with the previous two models, the process begins with defining what the product is required to do.

2. **Component analysis:** in this step, research is done to identify existing components that could be used to achieve the above requirements.

Oftentimes the match is not exact, but the component should at least provide partial functionality.

3. **Requirements modification:** here, the requirements themselves are adjusted to fit with the available components. If nothing can be modified, the process reverts back to step 2.

4. **System design with reuse:** in this step, a system design framework is created, or an existing one is adapted, to cater to the components found in step 2. New software can also be designed in this step if there are no existing components that would be appropriate to use.

5. **Development and integration:** the new software is created and the components found are integrated into the new system.

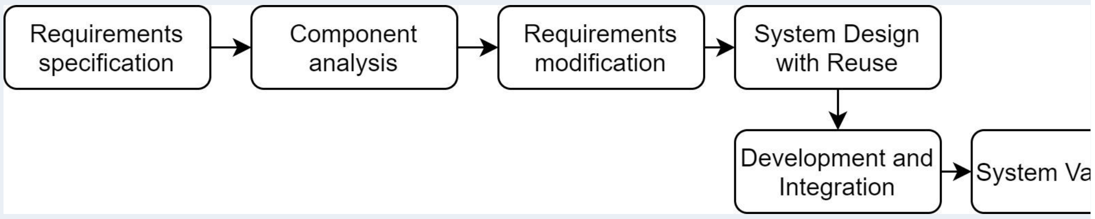
6. **System validation:** finally, the system is validated and finalised to be used for its intended purpose.

This model has obvious benefits: because you’re not wasting time with work that has already been done, it can be cost-effective, save time, and reduce the risks that accompany newly built components. However, the one major disadvantage is that the software is limited by the components available and so room for innovation is limited.

Compulsory Task

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

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

- Why do you think incremental development is the most effective approach for developing business software systems and less appropriate for real-time systems engineering?
- 
Requirements specification
Component analysis
Requirements modification
System Design with Reuse
Development and Integration
System Va

- 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

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

Think that 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.

References:

Sommerville, I. (2011). Software Engineering 9. 9th ed. Boston: Pearson Education, Inc., pp. 27-55.