TASK

Introduction to the Agile Development Process

Visit our website

Introduction

WELCOME TO THE INTRODUCTION TO THE AGILE DEVELOPMENT PROCESS TASK!

Welcome to the advanced level of your full-stack web development Bootcamp! As you have probably realised by now, full-stack web development involves more than just writing code. A full-stack web developer needs to be able to find out what a system is required to do, plan and design the system, work alone or with a team to build the system and then test, refactor, deploy and maintain the solution. This is quite a process! To better equip you to do this, this task will introduce you to the concept of a software process. We will specifically focus on agile development. In recent years "Agile" has become a buzzword in the business world, but it is not a new concept when it comes to software development. What exactly is agile development? This task aims to answer that question, as well as discuss one of the most popular agile methodologies, Extreme Programming.

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!

A note from the

HyperionDev Team

Another interesting career in the field of software development is Software Architecture. A software architect is a software expert who provides overall direction to a development team. The software architect is able to take a complex system and break it down into a manageable model, by identifying important requirements and constraints. Read more about this role **here**.

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 are 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.

WHY SHOULD YOU CARE ABOUT SOFTWARE PROCESSES?

It is not uncommon for software development projects to cost more and take longer to develop than initially anticipated. Many software systems also don't always deliver the functionality that the users want. People have spent a significant amount of time and effort to find ways to improve this situation which has led to the development of different software process models; steps that teams of software developers can follow so that they can deliver systems **on time, within scope and within budget.**

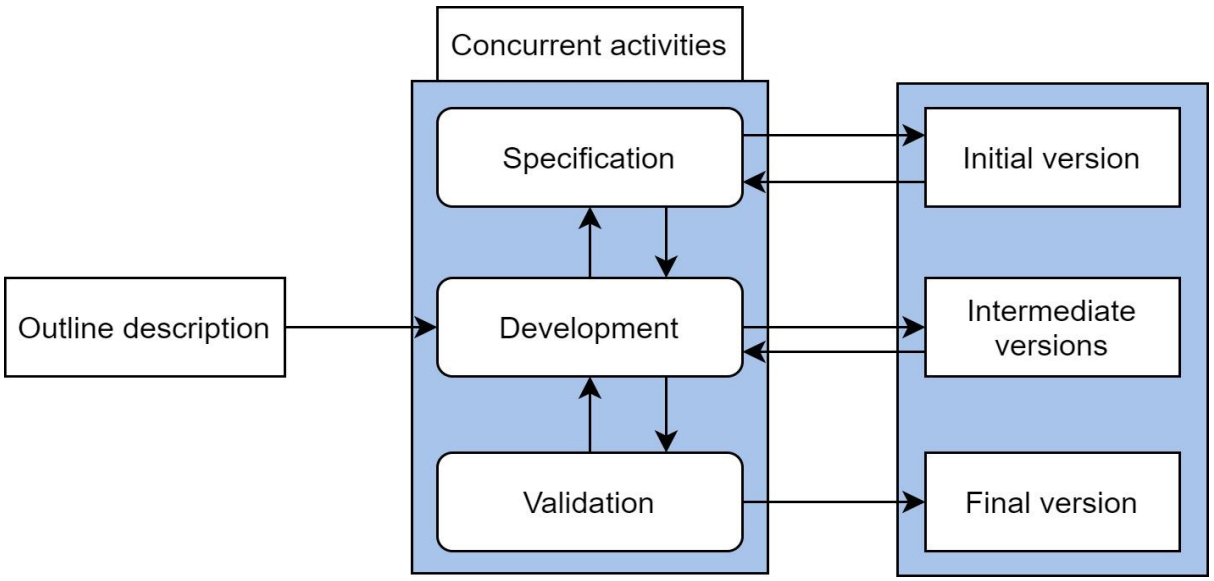
SOFTWARE PROCESS MODELS

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. This task focuses on incremental development.

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 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 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.

SPOT CHECK 1

Let's see what you can remember from this section.

1. What are the 3 advantages of the incremental development model?
2. What are the 3 disadvantages of the incremental development model?

WHAT IS AGILE?

Today, software is part of almost all business operations. This software needs to be developed quickly in order to take advantage of new opportunities and to respond to competitive pressure in a rapidly changing business environment. Therefore, the most critical requirement for software systems nowadays is often rapid development and delivery. Some companies are even happy to trade software quality and compromise on requirements in order to achieve faster deployment of the software.

Agile methods are incremental development methods in which changes are made in small increments. With agile methods, new releases of the system are created and made available to customers every two or three weeks and they involve customers in the development process to get rapid feedback on changing requirements. Agile methods also minimise documentation by using informal communications rather than formal meetings and written documents.

Agile methods allow the development team to focus on the software rather than on its design and documentation. They are best suited to application development where the system requirements usually change rapidly during the development process. Using agile methods allows you to deliver working software to customers quickly. These customers can then suggest new requirements that can be added or suggest how to change requirements later on.

Agile development is a term that is used to refer to a number of different iterative and incremental software development methodologies. Some of the most popular agile methodologies are:

- Extreme Programming (XP)
- Scrum
- Crystal
- Dynamic Systems
- Development Method (DSDM)
- Lean Development
- Feature-Driven Development (FDD)

The best known of these methodologies is Extreme Programming. We will describe this methodology in greater detail later on in this task.

Extra resource

Watch **this short video** to get a quick overview of what Agile Development involves before reading the rest of this section.

All of these agile methodologies share common values and principles which are derived from the Agile Manifesto found below:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Agile methodologies have worked well and were successful for some types of system development such as:

- Product development where a software company is developing a small- or medium-sized product for sale.
- Custom system development within an organisation, where the customer is keen to be involved in the development process and where there are not a lot of external rules and regulations that affect the software.

EXTREME PROGRAMMING

One of the best known and widely used agile methodologies is Extreme Programming (XP). It is called “Extreme Programming” because this methodology pushes recognised good practice, such as iterative development, to “extreme” levels (Beck, 2000 as cited in Sommerville, 2011, p. 64). For example, in a single day, several new versions of a system can be developed, integrated and tested, with XP.

XP takes a client’s requirements for an application, known as *user stories*, and writes them onto a *story card*. The team then breaks down the requirement into smaller tasks. The tasks are then allocated to a pair of programmers who begin by developing tests for each task. Once the tests are created, the programmers can begin tackling their allocated tasks. Because there are multiple pairs of developers all working on different tasks concurrently, several versions of a programme could be created in a day. However, new versions are only released to clients every 2-3 weeks. This release deadline is never missed; if it is looking like there is too much that needs to be done before the deadline, it is the client’s responsibility to decide what functionalities can be postponed to a later release.

STORY CARD:

StoryTag: DocBookToHTML Release: Book Priority: 1

Author: Joanne on: 2/21/02 Accepted: 3/17/02

Description: Make the DocBook files readable and printable.

Considerations: HTML has some drawbacks: Estimate: 4.1
- Printed version is not production quality.
- Footnotes can't appear at end of page.

Who	Task	Est.	Done
Rob	Simple tags: <chapter>, <title>, <para>	1	2/24
Rob	Asymmetrical tags: <attribution>	1	3/3
Rob	Contextually related tags: <title>	1	3/11
Rob	Stateful output: <footnote>	1	3/14
Joanne	Acceptance Test: Print the first chapter	.1	3/17

Example of a story card

As mentioned above, a story card is written up for each new feature that the client requires the developing programme to have. When drawing up a story card, the requirement is broken down into tasks. Each task is given an estimated effort level and resources required. It is usually up to the client to decide which story cards are the most urgent and which should be released with the latest version.

Extra resource

Watch **this short video** to see how user stories are used and what makes a good user story. In addition, **this video** provides some practical examples of story cards.

PAIR PROGRAMMING:

This is one of the key components of XP. As mentioned, developers work in pairs to complete tasks. Team members usually rotate so that all team members will have worked together at some point. One of the main benefits of this process is its promotion of refactoring: if your partner can't understand your code, how can you expect other people to? Similarly, having two people work on the same code creates an incidental code review system because there is always at least one other person who has seen a piece of code. Finally, having multiple people work on the same task creates a sense of collective ownership — every team member is responsible for the end product. This ensures a high-quality product. According to

Cockburn and Williams (2000), pair programming causes the programming process to take 15% longer, but it had 15% fewer defects, which is quite significant. Programmers also reported a rise in job satisfaction through the collaboration.

TESTING IN XP:

The testing process in XP is somewhat unusual, but there is a good reason for it. Unit tests for tasks are developed before the task itself is even started! Why? Firstly, this forces the developers to have an in-depth understanding of the specifications — you can't predict what can go wrong until you have a comprehensive understanding of what the task is meant to accomplish. Secondly, it checks that the implementation of the task is in line with the requirements given. Thirdly, it avoids test lag: when a developer works faster than the test creator. Finally, it prevents the introduction of errors that may have slipped through the cracks in the current version.

Tests are created using automated testing frameworks. These tests need to be standalone, executable components that can simulate submission input and should be able to check that the subsequent output is what is expected.

Clients help to create incremental acceptance tests for the release of the latest version. This includes testing the programme with real data to check if it meets the real needs of the client.

LINKING XP TO THE MANIFESTO:

Looking at the information above, you should be able to see quite clearly how XP aligns with the Agile Manifesto, namely:

1. **Individuals and interactions over processes and tools:** Programming is done in pairs and the whole development team is accountable for the code. The development process also does not involve excessively long working hours.

2. **Working software over comprehensive documentation:** The system is released every few weeks. Requirements are based on user stories recorded on story cards. These guide what features are to be included in the next version of the program. The constant refactoring of code improves code quality and simple designs are used.

3. **Customer collaboration over contract negotiation:** The client is a key member of the development team and is responsible for defining the acceptance tests for the system.

4. **Responding to change over following a plan:** This is done through regular releases and designing tests before code is written. The code is also refactored by the pair programmers often. New functionality is also constantly integrated.

Compulsory Task 1

Follow these steps:

- Create a file called **answers.txt**. Answer the following questions in answers.txt:
    - Reread “Agile Manifesto”. Can you think of a situation in which one or more of the four “values” could get a software team into trouble?
    - Describe agility in your own words.
    - Why do you think requirements change so much in software development projects?
    - Explain why test-first development helps the programmer to develop a better understanding of the system requirements.

Completed the task(s)?

Ask your mentor to review your work!

Review work

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:

Cockburn, Alistair & Williams, Laurie. (2000). *The Costs and Benefits of Pair Programming*.

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

Sommerville, I. (2011). *Software Engineering 9*. 9th ed. Boston: Pearson Education, Inc., pp. 56-81.

SPOT CHECK 1 ANSWERS

1.
    a. It is cheaper to change the system if the customer's requirements change.
    b. It is easier to get feedback from the customers regarding the work that has been done on the system.
    c. Useful software is delivered and deployed more rapidly to the customer, even if all the functionality has not been included.
2.
    a. It is difficult to have accurate documentation that reflects all versions of the system because there are so many different versions.
    b. The structure of the system tends to degrade as new systems are added.
    c. 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. 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.