University of Portsmouth
BSc (Hons) Computer Science
Second Year

Software Engineering Theory and Practica (SETAP)

M30819
September 2023 - June 2024
20 Credits

Thomas Boxall
up2108121@myport.ac.uk

Contents

Page 1

Lecture - A brief introduction

2023-09-29 11:00 Claudia

1.1 What is Software Engineering Theory and Practice About?

"How to engineer software."

Software Engineering Theory and Practice (SETAP) teaches us how to engineer software, which is different from hacking software. Until now - the majority of development we have done has probably been through hacking software. Engineering implies a process, a set of steps we follow every time in order to be able to replicate what we are doing.

SETAP is not just another programming module. We need to know how to program and be familiar (or willing to learn) a language chosen in our groups but we won't be given programming stuff to learn. The majority of the programming will come in Teaching Block 2, when we implement our applications. TB1 will primarily be for analysis and design of the solution. While a portion of our final mark will come from our programming ability, it will not be the entirety of our final mark.

The module is designed to make us work as part of a team, and improve our skills at that. The thing that is important to remember is that when working in a team, things take longer to complete.

As part of our project, we will be writing documentation, more information to come about this at some point.

1.1.1 Assessments

Working in groups of 5-6 students, we will develop a medium size application. There are two submission points (Friday 15 December 2023 and Friday 10 May 2024) at which different things will need to be handed in. Each submission is equally weighted at 50%.

1.2 Rules

A number of rules have been designed to make it clear how this coursework is assessed.

1. Both submissions and all their components (e.g. code repository and demo) are team submissions
2. An overall mark will be assigned to each submission based on the merits of its content. We will call this `MarkOverall`.
3. Each submission must be accompanied by a *Contributions Table*
4. All team members must agree with and sign off the information provided in the Contributions Table. If someones name does not appear in the table, the assumption is you did not contribute anything.

5. Individual marks will be decided based on the percentages included in the contributions table as follows:

$$\text{IndividualMark} = \text{MarkOverall} - (\text{MaximumContribution} - \text{IndividualContribution})$$

6. If everyone on the team made an equal contribution, all team members should be assigned the same percentage, i.e.

$$\frac{100}{\text{NoOfTeamMembers}}$$

7. Submitting a Contributions Table where the percentages do not add up to 100% will lead to all members of the team being assigned the MarkOverall.

8. Failure to submit a Contributions Table will lead to all members of the team being assigned MarkOverall.

9. Mediation support is available for completing the Contributions Table in cases where any (or more) members of a team does not agree with the information provided in the table. (*Please make sure you contact the module coordinator in good time - at least 10 days before the deadline*).

10. Disagreements in relation to the Contributions Table brought up 2 days before the deadline or later will not be considered for mediation support.

Page 2

Lecture - Software Development Process Models

2023-10-06 11:00 Claudia

2.1 Software Development

Software Development is the process which is undergone to design and build software. With it being a process, there are defined steps which can be repeated time and time again to gain the same results. The structured nature of a proper software development project should mean that at the end of it, you have a good product which meets all the requirements, is well documented and can easily be maintained in the future, hopefully.

2.2 Stages of Software Development

1. Specification
2. Design
3. Implementation
4. Testing & Evaluation
5. Maintenance

2.2.1 Specification

At the end of this stage, there will be a description of what the system should do. This does not include *how* it does it, just what it does. The description will also contain how well it does it, including constraints of the system and non-functional requirements (for example, how long it should take to do a process). The produced document will act as a checklist of deliverables and will often be produced in collaboration between the business and legal teams; it will also form the basis of the contract. Over the duration of writing the document, there may be many iterations and cycles of talking to steakholders to ensure the document is the best it can be when completed so that the design and future phases are as smooth as possible.

A major part of the specification writing phase is gathering user requirements, there will often be lots of back-and-forth with users while gathering these. The specification also includes the system requirements, this features datatypes of data which needs to be gathered - again lots of going back to users in this stage to ensure its gotten right!

2.2.2 Software Design

The software designing stage is about designing the way the application looks, feels and functions. This encompasses: the algorithms (including problem solving and data structures); behaviour modelling (including data flow and exception handling); use case modelling (including actors and (in)valid scenarios); architectural design (including the structure of the system, as components and relations); the interface design (including GUI and how the user interacts); and database design (including the data models).

2.2.3 Software Implementation

The software implementation phase has a few things which have to be done.

This phase includes the actual coding of the system - where the plans made in the design phase are converted to executable code.

Version control is also used here, this allows for management of different versions of the code and tracking changes within it; as well as allowing collaborative coding between multiple people.

Continuous Integration and Continuous Deployment is configured in this phase whereby the deployment of code is automated based off of something such as committing to a branch on GitHub.

Documentation is also produced at this phase. This will generally be of two types - user documentation includes how to guides and instructions for the end users on how to use the software; and code documentation is produced which details how the code works so future developers have some clue what is going on when they get round to looking at it.

2.2.4 Software Testing & Evaluation

This is the final stage done with the initial development of the software. Through testing and evaluating the software, we validate that the right software has been built; verify that the system is being built right; evaluate if the non-functional requirements are met; and check that the client accepts the system.

There are a number of different ways in which this can be done. Before delivery, the entire system can undergo acceptance testing before it is delivered to the client. Different versions of the system get tested throughout development. Throughout the implementation cycles, it is a good idea to have usability testing. Unit test are often done throughout implementation as well as components are tested independently though out implementation.

2.2.5 Software Maintenance

The final stage of software development is the longest - supporting & maintaining the product throughout it's lifespan / the contract's lifespan.

There are a number of different types of maintenance which will get done - perfective (where algorithms, data structures and the system architecture gets optimised); corrective (fixing bugs while the system is in use); preventative (reviewing software quality and upgrading where needed); and adaptive (update to reflect changes in requirements, hardware, storage components and third party software).

There are four stages to each piece of maintenance carried out on software:

1. Specify change, check if there are existing solutions and explore reuse
2. Identifying existing parts of the system related to the change
3. Implement and integrate the change in the existing system

4. Re-test the system: new requirement and all existing requirements.

2.3 Software Development Lifecycle Models

There are a number of well-documented and well-used *Software Development Lifecycle* (SDLCs) Models.

Incremental where one bit of the software is developed at a time

Agile which uses agile development methodology

Waterfall which follows sequential development

Iterative which creates one version at a time

Reuse which reuses code over implementing code.

2.3.1 Iterative Development

This development method consists of a number of cycles - each cycle gets the product closer to the final goal and contains a number of steps including: specification building, design, implementation, and testing. The feedback from each cycle influences the next cycle.

2.3.2 Incremental Development

In this development method, after developing the initial system requirements specification document, a chunk of the design is split off and developed as the first increment. This development includes the design, implementation and testing. Once this increment is completed, another increment would begin - this will contain the same stages except the testing stage will also include testing of the new increment plus all the previous increments.

2.3.3 Reuse Oriented Design

In this development method, you begin with designing the specification. Then a discovery of the currently available software is undertaken, this includes an evaluation of the currently available software and the feasibility of modifying it to use in this project. The development stage comes next, which primarily involves adapting the existing software; including writing new components and configuring the system. Finally, the integration testing is completed where all reused and new software are put together and tested based on the specification.

2.3.4 Waterfall Design

This is the oldest design methodology, where each stage is done to completion before the next one begins. If you have to go 'back up the waterfall' then you have to re-complete all subsequent stages again. The client is often only involved at the first and last stages which is really problematic if a design requirement is misunderstood.

The stages are:

Specification Problem specification and SRS definition complete

Design Provide the complete design of the system

Implementation Convert the design into code. Do not evaluate until implementation complete

Testing Test the system as a whole

Maintenance Deploy and maintain the system. Any issues - go back to the specification.

Page 3

Lecture - Agile

2023-10-13 11:00 Claudia

3.1 History of Agile

The agile development methodology came about when developers were getting tired of the constraints and paperwork requirements of Waterfall and other development methodologies of the day. The methodology was published in February 2001 by a number of extremely experienced developers who said that they value the following during software development

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

Whilst the *Agile Manifesto* doesn't hate on the bracketed alternatives (above) too much, it does justify why it prefers the alternatives:

Responding to change "make a detailed plan for the next week, rough plan for the next 3 months and extremely crude plans beyond that"

Customer Collaboration "the best contracts are those that govern the way the development team and the customer will work together"

Individuals and interactions "a good process will not save a project from failure if the team doesn't have strong players, but a bad process can make even the strongest of players ineffective"

working Software "software without documentation is a disaster. [...] however, too much documentation is worse than too little"

3.2 Agile Development Terminology

Customer the person / group that defines and prioritises features

Release Plan maps out the next six (or so) iterations which has details of the next release

Refactoring rewriting code to change structure but not behaviour

User Stories are tokens of a conversation about a requirement

Iteration Plan the developer selects the number of stories to make into tasks

Pair Programming where pairs of programmers co-author code

3.2.1 Pair Programming

Pair Programming is a commonly used technique used in Agile Development as it allows for quicker peer-reviewing of code to take place rather than having to wait for the code to be submitted and reviewed formally when a feature is complete. In Pair Programming, one programmer types the code and the other watches for errors, these roles change often and the code is marked as being authored by both programmers. The membership of pairs is changed very frequently - sometimes with pairs changing during a single working day (therefore, each programmer is in two different pairs in a day), this leads to every member of the team working with every other member at different points.

3.3 Scrum

Scrum is one of the Agile Software Development methods. Another is Extreme Programming (not covered in this lecture).

Scrum is a efficiency focused method whereby teams meet daily and work in a series of sprints. A sprint will have a small, achievable, goal. Daily "stand up" meetings are held to ensure that the entire team is on the same page with each other. This is lead by a *Scrum Master*. There will be a longer backlog of tasks which get selected from to build the sprints. The *Product Owner* represents the customer and prioritises the requirements. The practices of Scrum are shown below.

Sprint Planning Meeting is attended by all, and the Product Owner (PO) selects tasks from the backlog and defines the goal for the sprint

Sprint Review Meeting is where the team demos the product increments to the PO at the end of the sprint

Sprint Retrospective is where the team and Scrum Master (SM) review the just-finished sprint at the end of it, analysing what went well and what can be improved

Daily Scrum is the 15-minute meeting attended by the team and SM

Product Backlog is the list of all functionality desired for the system; it is written and prioritised by the PO and updated by the SM

Sprint Backlog is the list of functionality planned for the sprint. It is a subset of the product backlog

Sprint burndown chart is a chart which shows progress on a daily basis

3.4 When to use Agile

Due to the nature of Agile development, there are times when it is a good tool to use and times when it is a bad tool to use.

- When the requirements change often
- When the product you are building is small-to-medium sized
- When the team building the product is small-to-medium sized
- When the customer is prepared to be involved as a stakeholder
- When there are a limited number of external rules and regulations which the software has to comply with

3.5 When not to use Agile

- When developing safety critical systems
- When the team is larger or when the team is distributed geographically
- When the customer isn't experienced with Agile projects
- When the customer needs to be involved all the way through as it can be difficult to keep the customer involved in the process
- When it is not possible to get a strict contract drawn-up up front