G51FSE
"Morning of the exam" revision guide

Craig Knott
Version 1.2

May 25, 2012

0.1 Humble Beginnings

What is software engineering?

"The application of a systematic, disciplined, organised and quantifiable approach to the development, operation and maintenance of software - and the study of these approaches"

When engineering software, we don't do it in an adhoc way. As we are generally working as a team, this would cause all sorts of problems - we need to work together. Thus, we need some sort of agreed structure, or framework.

The structure or framework can be summed up as a software engineering methodology, each of which needs the following properties:

Systematic	Complete certain stages before progressing to others
Disciplined	Each stage has a defined set of goals and objectives to achieve
Quantifiable	A way to check if these goals have been met

The ultimate goal of the application of these methodologies is to produce professional, understandable, relatively bug-free software, that satisfies the requirements of the customer, consumer or stakeholder.

For simple projects, achieving these goals can be very easy. But for complex projects, that are large, have many people, and many independent parts of software, this can be very hard.

0.2 Stages of Software Engineering

There are generally six main stages of the software engineering process. The interaction and order of these sections depends on the methodologies used.

0.2.1 Requirements

A list of things that the software must do
The aim of this section is to produce a list of Requirements that the system must adhere to. This document, the Requirements Documents, should be a detail of what this system does, and **not** how it should do it.

The four main sections are as follows

- Feasibility Study
  Will the new system be able to satisfy the user needs with the available technology and budget?
- Requirements Analysis
  Finding out what the stakeholders require of the system
- Requirements Definitions
  A list of requirements that the system must meet, in a standard language format.
- Requirements Specification
  A list of the requirements in detail

There are two main types of requirements, functional and non-functional.

Functional	Impact the functioning of the software. Must be formally testable. E.g "Pressing left makes the player go left"
Non-Functional	Constraints on the software E.g "The game is fun to play" "The game is finished before June 8th"

0.2.2 Software Design

Representation of the workings of the system
The designs are a representation of the form (look and feel) and function (functionality and workings) of the system.Your designs need to be detailed enough so that someone else would be able to use your them to implement and produce the system you are attempting to specify. The benefits of designs are that, as you are designing, you become aware of limitations, and changes that you will need to make to the system.

Generally designs are as UML (Universal Modelling Language)Diagrams, which are a standard way of displaying information, including things like class diagrams or state-chart diagrams. It's easy to think of the designs as the "Recipe", and the requirements as the "Ingredients" to use in this recipe.

The two main steps to designing are

- Resource Planning
  What components will be needed for the system

Followed by either

- Top-Down Approach
  How the components will fit together. This can sometimes cause the problem of over-engineering, resulting in a rigid, unexpandable model.

Or

- Bottom-Up Approach
  How the components will individually work. Using this method can sometimes cause an unstable architecture.

0.2.3 Implementation

Using the designs to create the system
The implementation is the process of taking a design, and producing a computational system represented by it. This could be done by writing entirely new code, or using pre-existing code to build from the bottom up. There are some practises that should be adhered to when implementing your system.

Version Control Systems	Can be **centralised** or **distributed** Allow you to keep track of changes, help deal with iterative approaches to implementation when you do not have stable requirements and allow multiple people to work on the same piece of code at the same time without overwriting each other's work.
Coding standards	Where to put brackets, how to deal with white space, standard of naming variables Increase code readability and modifiability within the team, help make bugs easier to spot.
Commenting code	Comment any code that isn't self explanatory

When designing an object oriented system, a simple way to help create the necessary objects from a project description is to identify adjectives, nouns and verbs of the description, as these can be converted into different elements of objects, for example:

Identities, names of the object, are **nouns**
Behaviours, methods of the object, are **verbs**
States, variables of the objection, are indicated by **adjectives**

0.2.4 Integration

Combining the code for a full system
In this section, all the individual sections of the code are combined together. During this stage, you should also be testing that the section interaction is what is expected of the system.

0.2.5 Testing

Validation of the system against requirements
After integration testing, the entire system now needs to be tested. It is important that the code and system do what you intended them to do.
The most basic method of testing is to use debugging *printf* statements throughout your code, but this is very limited. A better method is to use an open source debugging program, which allow you to step through the program as it progresses.

There are two classifications of testing

White Box	Where you have access to the code Often used for debugging and unit testing
Black Box	Where you only have access to the system Often used in user acceptance tests

Many commercial developers use a testing harness when testing their system. Which is a framework that allows you to automate the test process.

Debugging

There are several main stages to debugging that you should remember when attempting to fix problems in your code, these are:

Understanding	Look at the information flow up to the bug occurring
Reproduction	Study the system environment when the bug occurs
Hypothesis	Explain the observed behaviour of the bug, and possible causes
Experiment Design	Design something that will falsify your hypothesis
Test	Run this test. If it falsifies your hypothesis, restart this process. However, if it cannot, you may have found the problem.
Implementation	Fix the bug using your hypothesis, and retest the system.

0.2.6 Maintenance

Deployment and upkeep of the system
This is the part of the system development cycle in which the system is to be integrated with the system that the customers are already using (whether that be alongside it, or in place of it). This can be a challenging part of the cycle, as this requires you to work with the infrastructure the customer already has and your code may need to be edited to make sure that everything functions with all the previous systems.
Depending on the terms of your contract, you may also need to continue to update this software. This is why following the system life cycle detailed above is important, so that any updates or modifications can be made with ease.

0.3 Methodologies

What is a methodology?

"Set of methods, principles and rules for regulating the process of creating and deploying software"

0.3.1 Water - Fall method

Description	Start at the beginning of the software life cycle and traverse through each stage until it is as the client specified. Then move onto the next stage.
Advantages	Very common, simple to remember and use
Disadvantages	Any mistakes in the earlier stages will have resonating effects through each of the stages after ward. Simple and linear, not good for changing requirements.

0.3.2 Royce's Model

Description	Similar to the water fall method, but you feedback the changes between each of the stages as you progress. Looking back at the end of each stage, refactoring and refining.
Advantages	Adapts to changes better that the water fall method and each previous section is constantly updated and improved.
Disadvantages	Similar to waterfall, errors can still resonate until they are spotted.

0.3.3 Big Design Upfront

Description	Get the design perfect before moving onto the implementation stage.
Advantages	Everything is planned to the last detail
Disadvantages	Not always possible to predict all possible problems

0.3.4 V-Model

Description	Three main stages of development, design, implementation and testing. All design stages are listed on one side, all testing stages down the other, with implementation at the bottom.
Advantages	Testing is applied at each stage
Disadvantages	Problems in requirements can cause lots of repeated tests and developing.

0.3.5 Spiral Model

Description	Represented as loops in a spiral shape. Where each loop represents a phase in the process and is a mini water-fall. There are no fixed phases or phase order, the spirals in the are chosen depending on what is required.
Advantages	Copes with evolving requirements
Disadvantages	Limited reuse (spirals tailored per project), hard to see the overall design up front.

0.3.6 Evolutionary/Rapid Prototyping

Description	Build a quick, cheap first version of the system, have it evaluated, and then improve it.
Advantages	System is constantly viewed and check by the client
Disadvantages	Some problems with the initial prototype may prove to be difficult to solve in later versions (if at all)

0.3.7 Throw-away Prototyping

Description	A simple first prototype is made, shown to the client, and then thrown away. The best bits are then used in the next model. And so on and so forth until completion.
Advantages	Only the best parts are kept, meaning persistent bugs are eradicated.
Disadvantages	Necessity to reproduce a lot of work at the beginning of each cycle.

0.4 Agile Development

The models and processes in the previous section are good for companies using large teams. However, more and more companies nowadays are trying to work with smaller teams, and adopt agile methods of development.

The Agile Manifesto is a list of twelve points that specify how agile development should be undertaken. The five key points are summarised below

- Deliver early, deliver often
- Welcome changing requirements
- Face-to-face communication is best
- Project deliverables should be in terms of the percentage of software that is actually complete
- Simplicity is essential

There are several ways to employ agile development values, including:

0.4.1 Extreme Programming

In Extreme programming, a few values must be adhered to

- The Customer is part of the team
- Application of "User Stories" over requirements
  Statements with enough detail to show how the customer actually wants to use the product. These stories can easily be converted to tests.
- Make the simplest working element, and optimise this later
- Use of pair programming
  The "Driver" writing the code
  The "Co-Driver" writing the tests and predicting future problems

0.4.2 Test Driven Development

Description	An iterative approach to software development in which the developers firstly write tests that the function/section must satisfy. Following this, they then actually write the code to pass the test, running the test after each new bit of code is added until it is finally successful.
Advantages	More tests are written, and thus the programmer is generally more productive. Debuggers aren't necessary and all code is thoroughly tested. Interface is considered before the implementation.
Disadvantages	Writing all the test can seem laborious and it is difficult to convince management to accept the model. Many successful tests can give the illusion of security and as a result, fewer software testing activities may be used.

0.5 Project Management

0.5.1 Managers and management

Project management is, obviously, a very important part of producing software. A good project manager is key to the success of the project. The best project managers will have good leadership skills, and a knowledge of what they and their developers are doing.

The seven sins of project management

- Gluttony
  Impossible schedules, over the top micro management
- Lust
  Trying to put too many features into a product during the time allowed
- Sloth
  Failing to do high quality work at all times, cutting corners
- Opaqueness
  Obscuring progress, quality or other attributes of the project

- Pride
  Believing to know everything necessary to build the product
- Wastefulness
  Misuse of critical resources
- Myopia
  Not seeing beyond your own work

0.5.2 Charting your progress

A project also needs a way to chart it's progress. The two main methods of doing so are listed below. Note that these charts are not all encompassing, they are to be treated as a rough guide.

Gantt Charts

Tasks are listed down the side and the duration of each task is represented as a bar (the longer the bar, the more time the task requires). Miles stones are added as diamond markers to be adhered to.

The advantage of the Gantt chart is that you can see the tasks as they work in parallel and can plan the critical path. Which is the list of tasks that must be performed before the project is completed, such that extending any task on that list will extend the entire life of the project. With the Critical Path, you can see bottle necks of the project, this allows you to allocate more man power to these parts to ensure their completion on time

The disadvantages of the Gantt chart are that they assume all details of the project are known at the beginning, it's difficult to show iterative stages of development and descriptions of tasks can be vague or sparse.

Milestone Plans

Another method of charting your progress is to use the less common milestone plan. Which focuses mainly on the end-dates of when things need to be completed, or when certain objectives need to be achieved.