Software Maintenance

Rishi Parmar

January 8, 2017

Contents

1 Introduction

First of all, I have no idea why we have an exam for this module. I don't even know what the exam is going to be on so I'm just going to go through some lectures and hope for the best. Anyway, 'software Maintenance is changing software after it has been delivered and is in use'. The majority of software maintenance work includes:

- Fixing coding errors
- Fixing design problems
- Adding additional requirements

Software maintenance can be put into **three** categories:

1. **Corrective Maintenance** $\longrightarrow$ Basically fixing bugs
2. **Adaptive Maintenance** $\longrightarrow$ Adapting the software due to environmental changes, such as laws and updated requirements from the business
3. **Perfective/Performance Maintenance** $\longrightarrow$ Improving the performance of the software, this doesn't change functionality

They mention some shit about how *maintenance* is 'preserving software in a working state' whilst *evolution* refers to improving the software. They talk about how shit code is when dealing with large software and basically a pain in the arse and WHY it is a pain in the arse. The reasons are pretty simple e.g messy and bad commenting. They then give a reminder of some Object Orientation concepts which we need to know/understand. These include:

- **Abstraction** $\longrightarrow$ When you only concentrate on the essential characteristics of the software. Basically removing the need to deal with BS
- **Inheritance** $\longrightarrow$ When one object acquires the properties of another which allows for ez object relationships
- **Encapsulation** $\longrightarrow$ Hiding internal implementation and requiring that user interaction can only be performed via an object's methods
- **Modularity** $\longrightarrow$ When source code for an object can be written/edited independently of the source code for other objects
- **Polymorphism** $\longrightarrow$ When classes can have different implementations of the same methods

For more information on Object Orientation and its concepts, check out my PG-13 notes on the topic.

They list the essentials of software maintenance in a list as follows:

- Understanding the client
- Understanding the code
- Refactoring the code
- Extending the code
- Working as a team
- Managing client expectations
- Managing maintenance process

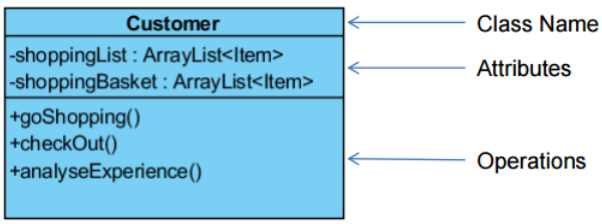
1.1 Understanding the code

They express that with large amounts of code, it is important to understand the structure of the code. Yep, you guessed it, that means class diagrams and other shitty visualisation techniques. I'll give a quick recap of all that garbage.

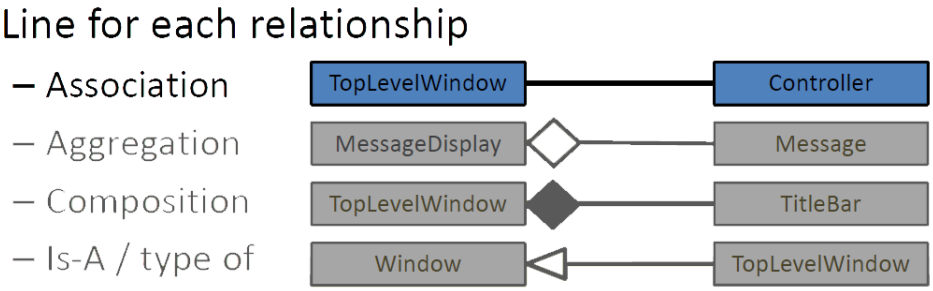
1.1.1 Class Diagrams Review

Classes are blueprints for objects in a software. They contain data and perform operations. Class diagrams represent these blueprints. They are said to address a 'static design view' of a system because they document the main structure of the software. This differs to behavioural diagrams such as use case and activity diagrams which document the dynamic aspects such as the methods and collaborations.
Class diagrams contain three rows, and arrows to represent relationships. The categories are:

1. Class Name
2. Attributes $\longrightarrow$ the variables and arrays etc.
3. Operations $\longrightarrow$ the functions/methods



Class diagrams are good because they provide a simple summary of the classes and relationships. But for larger projects, they can be a pain in the arse. Here's a reminder for the relationships (they don't give this on the slides so you can thank me later or by me a drink).



The speak a bit about how important testing is. They express the importance of testing. As for types of testing, they can be broken down into:

1.2 Testing Introduced

- **Regression Testing** $\longrightarrow$ This is when after you do a bug fix or change, you re-test to make sure that all of the old functionality is still there, and that the bug is fixed
- **Unit Testing** $\longrightarrow$ Automated tests to test the internal workings of the methods. The test is usually made to be small and as isolated as possible and so usually doesn't rely on other resources in the software