Software Engineering notes

Todd Davies

December 22, 2014

Introduction

The development of software systems is a challenging process. Customers expect reliable and easy to use software to be developed within a set budget and to a tight deadline. As we come to depend upon software in so many aspects of our lives its increasing size and complexity, together with more demanding users, means the consequences of failure are increasingly severe. Experience from nearly 40 years of software engineering has shown that programming ('cutting code') is only one of a range of activities necessary for the creation of software systems that meet customer needs. The rest of the time is spent on: planning and acquiring resources for the project; investigating the business and technical contexts for the system; eliciting and documenting user requirements; creating a design for the system; and integrating, verifying and deploying the completed components.

This course unit builds on the programming skills you have gained in the first year, to provide you with an understand-

ing of the major challenges inherent in real-scale software development, and with some of the tools and techniques that can be used in their attainment.

Since software engineering is a subject best learned hands-on, this is a project-based module that involves less traditional lecturing than usual. Instead, relevant skills will be acquired during fortnightly two-hour workshops, supported by a weekly one hour lecture. The understanding gained will be practiced through individual and team project assessments.

Aims

This unit aims to give students an introduction to the principles and practice of analysis, design and implementation in object orientated software engineering. Through experience of building a significant software system in a team, students will further their? experience and understanding of the problems that arise in building such a system. They will develop the analytical, critical and modelling skills that are required by a successful software engineer.

Contents

1 A history of software engineering

In the 60's, when programming projects started to get large enough to warrant their own software develoment strategies. The increased complexity of projects at this time was leading to many of them going over budget, over time, having low quality, not meeting requirements, and being difficult to maintain. This, even at the time, was refferred to as the software crisis.

Dijkstra summed it up in an article in Communications of the ACM:

> The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. (Dijkstra)

1.1 Why do projects fail?

We've seen that projects often failed early on in the history of software development, but why did each project fail? There are a large number of reasons why a specific project could end in failiure, however, there also a number of common reasons for disaster:

- Unrealistic goals
- Inaccurate estimate of project complexity/resources needed
- Badly defined requirements
- Unmanaged risk
- Use of immature technology
- Sloppy devevlopment practices

Now, there are software development practices and methodologies that aim to mitigate the risk of project failiure by defining how a project should be developed. Figure 1.1 shows the a generic view of such a process.

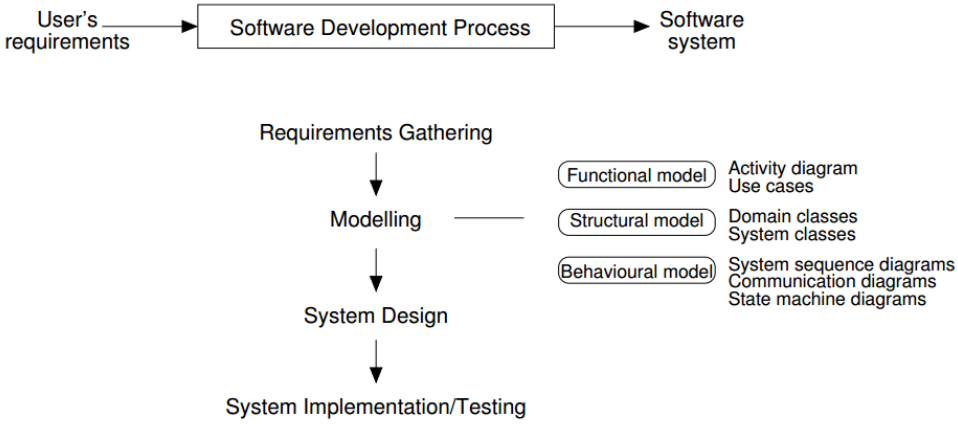

Figure 1: A generic software development process

2 Gathering requirements

Gathering the requirements of a software project is crucial, since if a developer doesn't understand what is being created, how can they create it properly? There are two types of requirements, functional and non functional.

Functional requirements are things that the system should textbfdo. For example, the system should produce both a a4 sized document, and a a5 sized e-readable copy.

Non Functional requirements are things the system should **be**. This could be that the system should be developed with ethical software development practices, or the system should be easily extensible for future development.

When gathering the requirements of the process, it is important to accurately capture the buisiness process that is being encapsulated in the system. It is important to understand the state of the process now, and the desired state of the process at the end of the development time. UML is a way of representing buisiness logic, and the relationship between differnt parts of the system. Activity diagrams can be used to create a logical model of the system.

Activities are composed of *actions*, which are non- decomposable pieces of behaviour. Activity diagrams are composed of activities.