MechEng Software Engineering 3

Lecture 2 : Friday 15th January

Simon Gay
Department of Computing Science
University of Glasgow

2009/10

Software Life Cycle
(Software Development Process)

Ray Welland

ray@dcs.gla.ac.uk

Why do Many Computer Systems Fail to Work as Expected?

- We have all heard of computing systems that fail spectacularly at enormous expense and possible loss of life.
  - Examples: the London Ambulance Service and the failure of Ariane 5.
- Smaller scale projects fail all the time, in a less visible manner.
  - It is often said that 80% of computer systems fail in some way.
- A bug in the code is a relatively rare cause of failure.
  - Most systems are built by people who understand the need for testing, and will do it thoroughly.
  - The Ariane 5 disaster is a rare example of inadequate testing.
- The most common form of failure is building the wrong system.
  - It is difficult to find out and clearly specify customer requirements
  - Systems are not properly validated against customer needs

Software Development

- Software Development consists of a number of activities:
  - Requirements Analysis and Specification (What?)
  - System Design (How?)
  - Implementation and Testing (Build the System)
  - Delivery (Hand over to the customer)
  - Evolution (Maintaining and enhancing the running system)
- Software Development Processes focus on the first four of these, regarding evolution as a separate task

Software Life Cycle

- A software life cycle is the framework within which we carry out the activities from requirements specification to delivery
  - The life cycle may be a simple sequence of phases (analogous to an insect's life cycle)
  - Or it may be *iterative*, we repeat steps within the sequence (or the whole sequence may be repeated, with multiple deliveries/releases)
- Initially we will assume that a simple sequence of activities is sufficient

Describing Systems

- To carry our the early stages of the life cycle we need to be able to define our requirements and describe our system design in some agreed format - we need a modelling language
  - Implementation is defined in terms of a programming language
  - We will use an object-oriented modelling language: **UML**
- UML = the Unified Modelling Language

Why ‘Unified’?

- Object-oriented modelling grew up in the 1980s
  - Grady Booch, 1982, “Object-Oriented Design”
  - Jim Rumbaugh, OMT (object modelling technique)
  - Ivor Jacobsen, Objectory Method
- UML
  - Booch & Rumbaugh combine in 1994, later joined by Jacobsen
  - The Three Amigos
  - Formed The Rational Corporation
    - Absorbed into IBM
- UML now managed as a standard by OMG
  - UML 1.0 in 1997
  - Now UML 2.0

UML's Diagrams

- Analysis
  - **Use Case Diagrams**
  - **Activity Diagrams**
  - **Class Diagrams**
  - **Collaboration Diagrams**
  - **Sequence Diagrams**
- Design
  - **Class and Sequence Diagrams** again
  - **Statecharts**
  - **Package Diagrams**
  - Component Diagrams
  - Deployment Diagrams

UML is not a development method or process

- UML is just a collection of ways (models) of representing a system
- It doesn't include any information about
  - how the representations should be created
  - the order in which the representation should be produced
  - how to determine the acceptability of the representations
- So we need a development process …

An Object Oriented Development Process

- There are many different ways of managing an object oriented project.
- One of the most widely used is the **Rational Unified Process** (see Sommerville 4.4), which has the following steps:
  - Inception
  - Elaboration
  - Construction
  - Transition
    - Transferring the system to its working environment

Inception

- All projects have a starting point
  - A brief description of the reason why the system is needed
  - What its scope will be
  - Roughly how much it will cost.
- The inception phase will generate the business case for undertaking the projected work.
  - How much it will cost
  - What benefits the resulting system will provide
  - How long it will take

Elaboration: Requirements Gathering

- This involves finding out what the customer wants.
- This will initially involve a brief project description.

  The development of a computer system for a community bank. A system is required whereby customers may open accounts and perform the usual transactions on these accounts (credit, debit, obtain balance). The bank is also required to report its total asset value to the government.
- The final result will be
  - A fairly comprehensive list of **use cases**
  - Non functional requirements.
- The project description will also be expanded.

Use Cases

- A ***use case*** is a typical interaction between a user and the system to achieve a simple goal.
- Some typical examples with the community bank example might be:
  - Create a new customer account
  - Credit an account; debit an account; find the account balance
  - Calculate total assets
- As you can see, these use cases are focused on one simple task (from the user's point of view).
- The actual engineering effort can vary greatly.
- An important part of designing a system is to discover all of the potential use cases.
- It will not be possible to get all of them immediately
  - it is necessary to get all of the important ones as early as possible.

Use Cases

- Systems will normally have a large number of small use cases.
  - each one should be as simple as possible, but no simpler.
  - the more complex the use case, the more scope there is for ambiguity.
- Many of the use cases will interact with each other, and these relationships will be described in a use case diagram.
  - there will typically be many different user roles associated with the use cases
  - in the banking example: clerk, customer and manager might be user roles
- Some of the use cases will be clear and unambiguous, whereas other will be difficult to describe accurately.
  - complex use cases are examples of requirements risk (see later).

Domain Model

- The domain model is a “high level” description of the system
- This is the “big picture” of what the system contains, how the system will operate and the context in which it will operate
- Differs from use cases in that it describes **the information and operations that will provide the functionality needed to realise the use cases**
- In object-oriented development, the domain model consists of a collection of interacting objects
- In object oriented design, which is what we are doing in this course, we look for things that make up the domain of interest.
  - in our example, we notice the words ***customer***, ***account*** **and** ***assets***, which could form the basis of our object oriented domain model.
- The domain model comprises conceptual classes that will support all of the use cases in our system, defined in a class diagram.

Elaboration: Risk Analysis

- We obviously want our systems to work, and need to know the areas of risk.
- The following four types of risk are present in most projects.
  1. Requirements Risk
  2. Technology Risk
  3. Skills Risk
  4. Political Risk
- We will look at requirements risk in detail, and so the other topics can be dealt with quickly.
- There will be other areas of risk associated with the software construction process itself, which we will deal with as they arise.

Political Risk

- All projects need funding before they get off the ground
  - Write the project proposal
  - Sell it to the people controlling the money
  - Protect it from threats.
- The amount of politics involved varies depending on whether you work in a small company, large company or government agency
  - Each project needs someone with political skill.
- We won't discuss political risk further.

Skills Risk

- It is common to hear the attitude that once someone learns to program then they can undertake any programming task almost immediately.
- This can lead to the situation where programmers who have trained in Ada using traditional analysis and design must work on a project in C++ using object oriented design, with no additional training.
- The reality is that every programming language has its own techniques which must be mastered before a person can use it well.
  - A project written in C++ in the Ada style will take a lot longer to complete than one written using C++ as it was intended to be used.
- It is true that we learn by making mistakes, but life is too short to make all of them personally. It is better to learn by the mistakes of others.
  - Of course, it is necessary to make a certain number of mistakes personally to learn well.

Technical Risk

- Technology moves fast, and many projects involve new technology.
- The most important technique for dealing with new technology is to design an experiment that uses the new ideas in a simplified way.
  - This will verify that the technology will work as expected.
- There will usually be a choice between several similar technologies, and so an experiment should be designed to find out which one is best.
  - For example – design a GUI interface to a relational database.
- One major form of technological risk is in fitting the components of a design together.
- Questions to ask during a design are:
  - What will happen if this piece of technology does not work?
  - What if we cannot connect these two pieces?

Requirements Risk

- The most common form of failure is to build the wrong system.
  - It may work perfectly, with lots of elegant algorithms, but does not do what is required.
- There are three main causes for this failure.
  - Lack of understanding of business context by the engineers
  - Cultural differences between engineers and customers
  - The fact that natural language is ambiguous, and the same words can mean different things to different people.

Requirements Risk (2)

- From the engineer's perspective, the risk is that we do not know what the customer really wants.
- The solution that we will use on this course involves
  - A more precise technique for describing what the system should do, what the requirements are. These are ***Use Cases***.
  - A way of producing parts of the system quickly, so that the customer can see how the requirements look in practice. This is achieved through ***Prototyping***.