CSCU9P5
Software Engineering I

Introduction to Software Engineering

Outline

General principles

The Software Development Life Cycle

Object Oriented Software Engineering

Software Engineering

Why should I learn about Software Engineering?

- Because building LARGE software systems is VERY HARD!

What is Software Engineering?

- Modelling
- Problem-Solving
- Knowledge Acquisition
- A Rationale Driven Approach
- Project management
- Team management
- . . .

Why learn Software Engineering?

A structured and systematic approach is needed for large systems

- Would you use the same approach to building a sky-scraper as you would for a garden shed?

Building large systems involves extensive group work

- A common understanding of the task and proper communication is essential
- Each member of the group needs to understand their task and how it *interfaces* with other tasks
- A method for dividing up the problem into manageable portions is important
- Groups and individuals need to communicate with a commonly agreed *design language*

Why learn Software Engineering?

An understanding of user requirements is vital

- With a small problem, the user requirements are usually obvious
- With a large problem, not all the users requirements will have been stated and some may be contradictory
- Discussing the problem with a client will often lead to a change in requirements
- It is very important that the major requirements are captured early on
- Early feedback and prototype analysis with the client are crucial

Adopting a Software Engineering methodology provides a solid framework that will enable you to solve large problems - it describes the major stages to undertake and guidelines on how to progress through them

SE & Modelling

Software Engineering involves modelling a problem

- Problems with many components are usually complex
- Complexity can often be dealt with through simplification of a system into its main component parts - a model
  - "A model is an abstract representation of a system that enables us to answer questions about the system"
- Models allow us to visualise and prototype a system
- There are many different ways to model a problem
  - » High level - concept
  - » Low level – detailed
  - » Diagrams
  - » Mathematics
- No two people usually model a large problem in exactly the same way

SE & Modelling

Building a model involves

- The Real World, A Problem Domain, A Solution Domain

The Real World

- Environment within which the system must operate
- For example, trading rules within a stock-trading system

The Problem Domain

- Concepts from the real-world that are *relevant* to the users requirements, e.g. stock components, traders

The Solution Domain

- Objects from the problem domain useful in the solution
- Objects that enable problem domain objects to interact

SE & Problem Solving

Software Engineering is about Problem Solving

- Formulate the problem
- Analyse the problem
- Search for solutions
- Decide on appropriate solution
- Specify the solution
- Implement the solution

} Aided by modelling

There is no standard algorithm to follow…

- You often have to try a number of paths (solutions) and then select the one that appears most appropriate
- Previous experience is very useful in selecting a solution
- You can also benefit from other peoples' experience (more later)

SE & Knowledge Acquisition

As you design a system, you acquire more information

- You will very rarely have all the facts at the start
- New knowledge does not always add to prior knowledge
- Sometimes newly acquired information may invalidate your previous ideas
- You need to carefully manage the knowledge acquisition process to ensure you do not miss out crucial information

SE principles help avoid the pitfalls

- e.g. Build flexibility into your design
  - » Keep component dependencies to a minimum
  - » If a major design revision is necessary due to new information, minimal dependencies will reduce the need to start again from scratch.

Outline

General principles

The Software Development Life Cycle

Object Oriented Software Engineering

The Software Development Life Cycle

Aspects of the software production process:

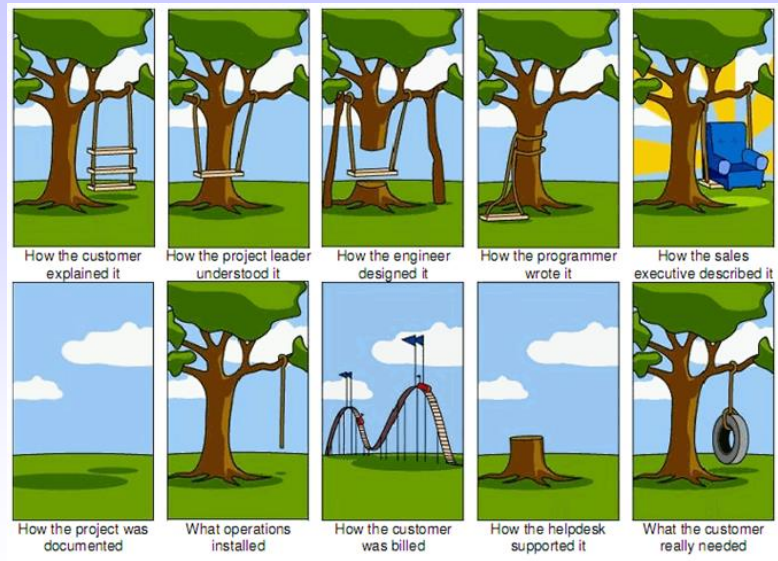
- Requirements (elicitation)
- Analysis (of the problem)
- Design (of the solution)
- Implementation (of the design in code)
- Testing (the code & design)
- Maintenance (upgrades, bug fixes)

Requirements

In this process, we are concerned about what it is the client wants us to produce

- Often the client will not know exactly what they want
- Discussing the problem with them often reveals new requirements that they had assumed would be delivered but had not been in the initial specification document
- The managers who place the contract for a system are often not the end users of that system
- One of your roles is to ensure that the client is getting what they actually want and that you are designing what they want, rather than what you think they want, and sometimes rather than what they say they want!

What to avoid: the classic Tree Swing analogy (this version from tamingdata.com)



Requirements

Early Prototyping Allows Requirements Elicitation

- Building a simple model (even a sketch of the main components) is often a useful method of checking that your understanding and the client's understanding are the same.
- A simple model allows you to run through scenarios and check that all the major parts are there (even if you have no idea about how to make each part work).

Requirements - Example

"Design a resource management system where resources are held in staff offices but are made available via a web based interface. Resources will be booked out via the on-line management system which will also send reminders when items are due for return."

Identify any obvious requirements

- Web Based
- Booking System
- Users
- Resources

The requirements can then be expanded: e.g

- Users: staff, clients, administrators
- Web/Booking system: staff only? Client interface needed?
- Resources: books, journals, CDs? DVDs? rooms?

Interactions should be identified: e.g

- Book resource
- Return resource
- Send reminders
- Resource available enquiry?
- Cancel booking?
- Client status check?
- Ban client?

Analysis

Once we have a proper set of requirements, the next step is the analysis of the problem.

- This involves understanding the scope of the problem
- Identifying the parts in the requirements that will and will not be part of the eventual solution
  - » For example, is the web server component part of our solution?
  - » Does the web site need to do user authentication?
  - » Database: Exists? Off the shelf? Design one?
- Connect them together in an appropriate manner

Design

Analysis of the problem indicates what the major components in the system are, it will not tell us how these components work.

Design involves

- Identification of major component boundaries
- Decomposition of the major components into smaller semi-independent sub-systems
- Design of the interfaces between these major components & sub-systems

Design

Design involves (continued)

- Identification of new components necessary to bridge the gap between objects in the problem domain and the solution domain.
- Identifying if any of these components are potentially re-usable or are available 'off the shelf'
- Flow of control within the system
- Flow of data within the system
- Data management (e.g. central repository or distributed information)

Implementation & Testing

Implementation - The translation of the Design into Source Code

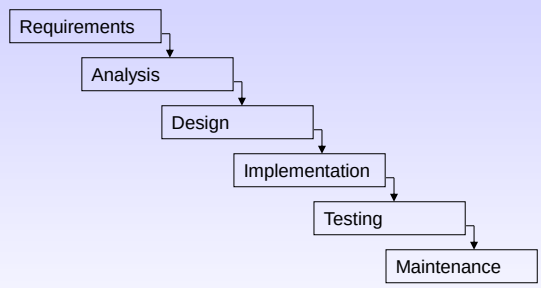
- For each identified component and interface in the design phase, create the source code that will implement it
- Integration of code components such that they perform as one system

Testing

- Check that each element / component does what it is required to do by the design, and each sub-system
- Check system meets the requirements specification
- Check system meets the clients' expectations
- Check system meets the users' expectations

Software Engineering Methods

Following the steps in the order listed here is called the 'waterfall' method of software engineering.

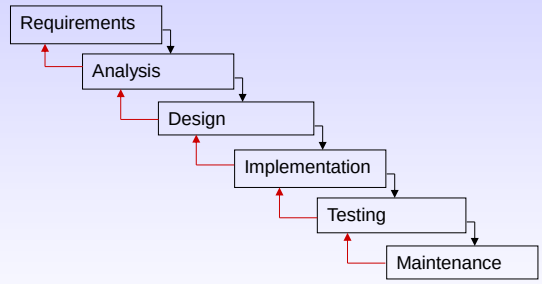


With this **over-simplistic approach**, each step presumes that the previous step was **successfully completed**, however this is *rarely the case*:

- For example, user testing can reveal major design flaws - it would be better to discover this early in the design process
- Approaches that acknowledge **iteration** are more realistic

Software Engineering Methods

For example: we can add iterative cycles to the simplistic waterfall model:



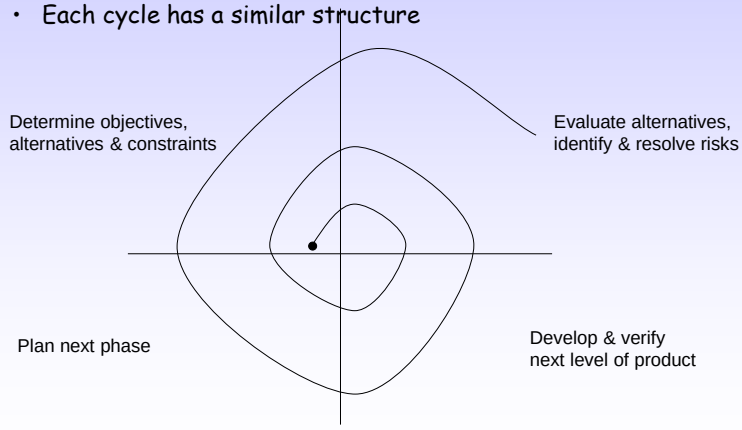
This allows us to return to a previous stage if a problem is discovered

"Longer range" are sometimes included too: e.g. Testing back to Design

Software Engineering Methods

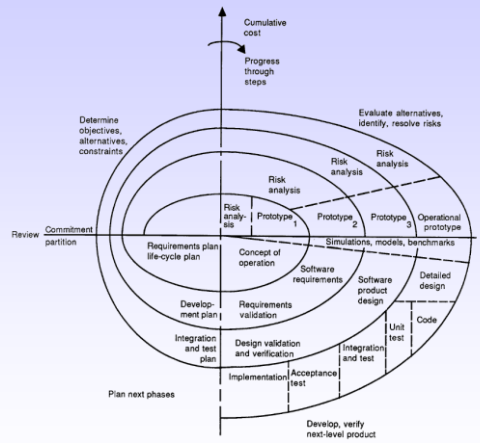
There are many different approaches

- Another example: Boehm's Spiral Model
- The cycles deal with major steps in the development process
- Each cycle has a similar structure



Software Engineering Methods

Boehm's original spiral diagram is very elaborate:

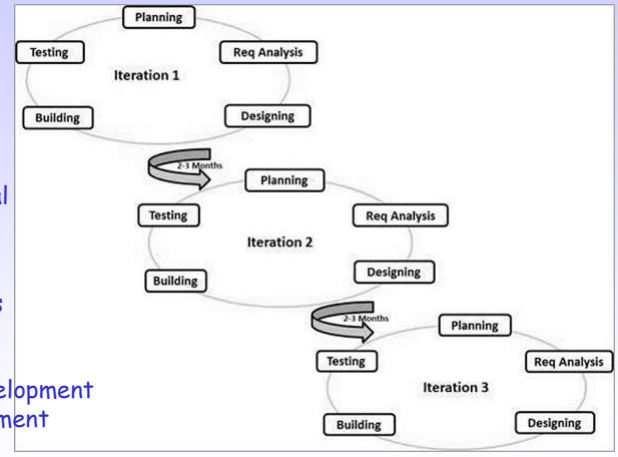


From: https://ieeexplore.ieee.org/document/59/
or http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

Software Engineering Methods

Another approach: The Agile Model:

- Iterative approach
- Working software build delivered after each iteration
- Each build is incremental in terms of features
- Particular methods:
  Rational Unified Process
  Scrum
  Extreme Programming
  Adaptive Software Development
  Feature Driven Development



From: https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm

Outline

General principles

The Software Development Life Cycle

Object Oriented Software Engineering

Software Engineering Methods

We would like a method that allowed us to model our problem and solution at an abstract level

- Abstract model allows early prototyping
- Test various scenarios

The method should support the concept of components, sub-systems and interfaces between components

It would be useful if the approach we adopt was supported by the programming language(s) we will use to implement the solution

- The abstract model could then be more easily converted to a lower level implementation whilst maintaining the core design features

Object Modelling

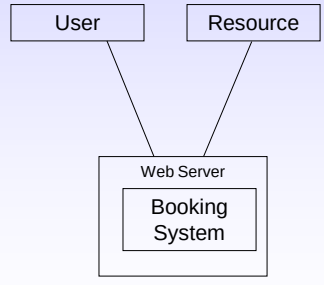
What has all this got to do with object modelling?

- Object modelling is one method of approaching the software development process
- Requirements analysis remains the same
- Emphasis is on the Analysis and Design phases of the software development life cycle
  - » Analysis & Design are concerned with modelling the problem and modelling the solution
- Supports the principle of early prototyping to improve the Requirements and Problem Analysis phases
- Objects identified in the design phase are readily implemented via an OO language

Object Modelling - Analysis

Analysis of the Problem

- Decomposition of elements in the requirements into objects
- We build a diagram of the objects identified in the problem and how they interact

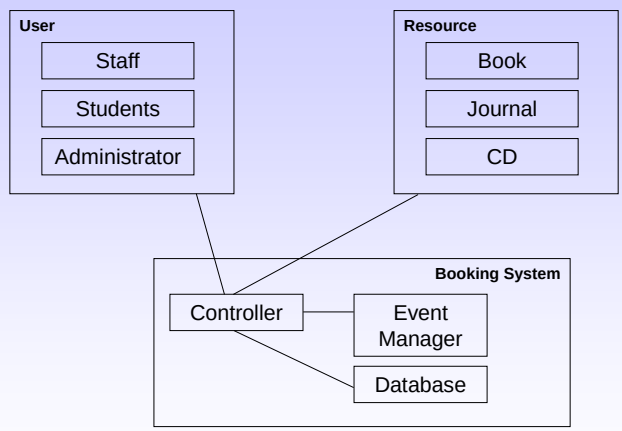


Object Modelling - Design Solution(s)

Design of the Solution

- Further decomposition and creation of new objects to facilitate communication or purely software related features
- There is a transition from our diagram of the problem to our diagram of the solution
- We may initially develop a number of alternative solutions
- This phase is iterative, with continual refinement

Design - Example



The Object Oriented Approach

Implementation - Turn each object identified in the design into a class (a template for an object)

- The class encapsulates the data and operations that define the behaviour of an object, for example:
  - » user objects, staff objects, resource objects
  - » a resource object would contain a borrow operation and data on when it was due back
- The class may contain references to other classes - allowing a hierarchy to be built
- We use programming languages that support object oriented concepts - e.g. Java, C++ or C#.
  - » Implementation process is more automated
  - » It is easier to take the high-level design structure and convert it into the equivalent low-level code due to a direct parallel with the OO programming language structure

End of lecture