Introduction and Process

Lecture 1

Ruzanna Chitchyan, Jon Bird, Pete Bennett
TAs: Alex Elwood, Alex Cockrean, Casper Wang

Purpose of Unit

Previous units aim to improve your knowledge as computer scientists

This unit helps package this knowledge as employability skill-sets

This is achieved by targeting the following industry-relevant issues:

- Transferable skills: Management, Collaboration, Communication
- Platform skills: applying your prior programming language knowledge
- Scalability skills: Development “in the large” at industrial scale

Software Development

But have you heard of ***Software Crisis***

- Software is getting larger and larger
- Software is getting more complex
  - Complex domains (e.g., human behaviours)
  - Systems dependency (e.g., energy and cars)
- Time to market is shorter than ever

And Projects Fail

Ariane 5 Flight 501

Mars Climate Orbiter

Therac-25 Radiotherapy

Why Do Software Projects Fail?

- Bad developers – not the main problem!
- Over budget
  - Poor requirements
  - Over-ambitious requirements
  - Unnecessary requirements
- Contract management
- End-user training
- Operational management

Software Engineering

- Software Engineering
  - Engineering**: cost-effective solutions** to practical problems by applying **scientific knowledge** to building **things** for **people**
    - Cost-effective solutions: process and project management, contracts…
    - Scientific knowledge: modelling, proofs, testing, simulation, patterns...
    - Things=software
    - People: customers and end users

OK, so how do we do *that* ?

Collaboration Tools and Techniques

- UML Designs: Diagramming to reach a consensus on design
- GitHub: Sharing of documents for effective collaboration
- Kanban: Task allocation and progress monitoring
- Test-driven development: Stop other people breaking your code !
- Other techniques are encouraged: Feel free to experiment and explore

Weekly Themes

1. Introduction + Overview
2. Agile Development
3. Requirements Engineering
4. Design
5. Software Quality and Testing
6. [Reading Week]
7. Project Management
8. Human Computer Interaction: Qualitative Analysis
9. Human Computer Interaction: Quantitative Analysis
10. [Easter break: 3 weeks]
11. Advanced Topics in Software Engineering
12. Module Closure

Groupwork

- We can only hope to tackle a real-world scale application...

  ....if we work in teams towards the end goal

- Working together is also a key transferable skill
- In any non-trivial industrial project, you'll be working in a team (maybe summer project!)
- It's not easy – which is why it is essential to practice now!

- Teams of 5 (or 6), assigned randomly

“Tasting Stick” Metaphor

- All of the previously named techniques are large and complex
- We don’t expect you to become experts in everything
- The aim is to give you a “taster” of everything
- So you can at least be able to say that you have tried them
- One way to think of this unit is as a “tasting stick”
- Working in groups, individuals are likely to specialise in some aspects

Software Development Life Cycle Processes:

Which tasks are to be done and in what order?

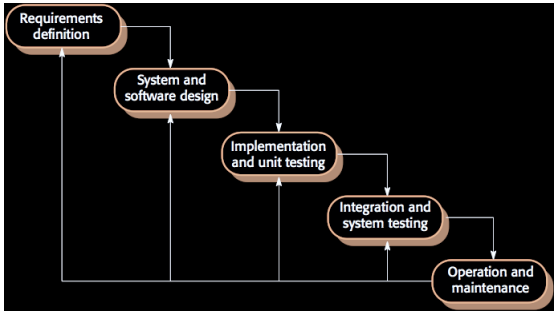
Software Development Tasks

- Requirements Analysis
- Planning
- Design: high level and detailed
- Development
- Testing
- Deployment
- Operation and Maintenance

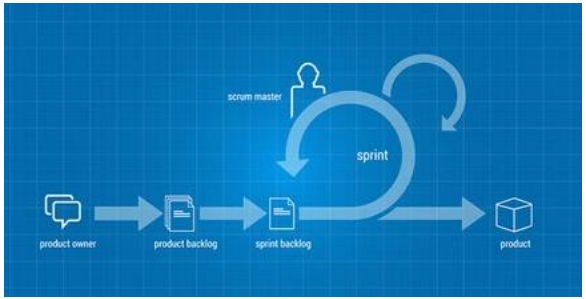
These to-does are combined in various sequences, making up different Software Development Life Cycle Processes

Software Development Life Cycle Examples

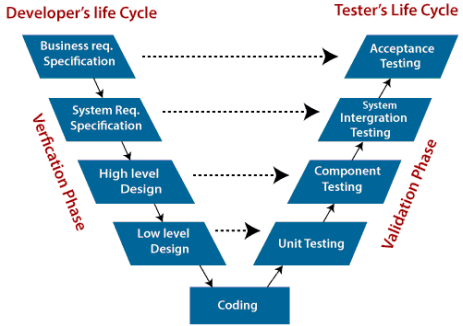
Waterfall



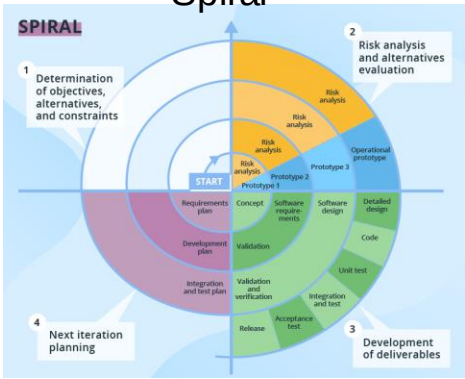
Agile



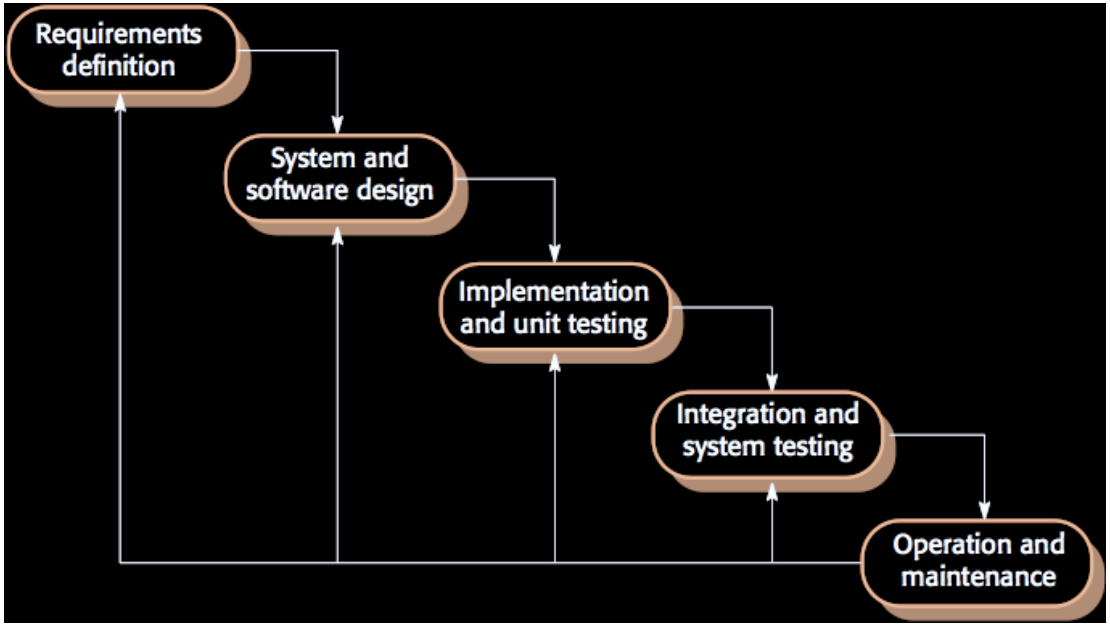
V-Model



Spiral



Waterfall Software Development Life Cycle



Requirements: What to Implement?

- What should the system do ?
- What are the needs of the users ?
- What are the needs of the host organisation ?
- What are interoperability needs of existing systems

We need to think about functional elements: what the system should do?
And also non-functional elements: quality properties of system operation, e.g., security, ease of use, response time etc.

Design: How to Structure Software?

- What objects, databases, servers, services etc. should we create?
- How are these elements structured into a system?

Why to design?

- Helps to decide where to place requirements
- How the parts of the system will be interacting
- Divide up work between team members

Implementation

Not just programming, but also…

- Parallel working and code sharing
- API partitioning and “firewalling”
- Versioning, integration and config management
- Development environments and automated build
- Automated testing
- Docs and training material

Verification and Validation

- Verification: check if the software/service complies with a requirements, constraints, and regulations. ("Are you building it right?")
  - Demonstrates that system meets specifications

- Validation: does this meet the needs of the customers/ stakeholders? ("Are you building the right thing?")
  - Demonstrate that system meets user needs

- Can pass verification but fails validation: if built as per the specifications yet the specifications do not address the user's needs.

- Involves checking, reviewing, evaluating & testing

Waterfall SDLC: Advantages and Disadvantages

- Simple to use and understand
- Every phase has a defined result and process review
- Development stages go one by one
- Perfect for projects where requirements are clear and agreed upon
- Easy to determine the key points in the development cycle
- Easy to classify and prioritize tasks

- The software is ready only after the last stage is over
- High risks and uncertainty
- Misses complexity due to interdependence of decisions
- Not suited for long-term projects where requirements will change
- The progress of the stage is hard to measure while it is still in the development
- Integration is done at the very end, which does not give the option of identifying the problem in advance

Case Study: Insulin Pump Control System

An insulin pump is a medical system that simulates the operation of the pancreas (an internal organ). The software controlling this system is an embedded system, which collects information from a sensor and controls a pump that delivers a controlled dose of insulin to a user.

People who suffer from diabetes use the system. Diabetes is a relatively common condition where the human pancreas is unable to produce sufficient quantities of a hormone called insulin. Insulin metabolises glucose (sugar) in the blood. The conventional treatment of diabetes involves regular injections of genetically engineered insulin. Diabetics measure their blood sugar levels using an external meter and then calculate the dose of insulin that they should inject.

The problem with this treatment is that the level of insulin required does not just depend on the blood glucose level but also on the time of the last insulin injection. This can lead to very low levels of blood glucose (if there is too much insulin) or very high levels of blood sugar (if there is too little insulin). Low blood glucose is, in the short term, a more serious condition as it can result in temporary brain malfunctioning and, ultimately, unconsciousness and death. In the long term, however, continual high levels of blood glucose can lead to eye damage, kidney damage, and heart problems.

Current advances in developing miniaturized sensors have meant that it is now possible to develop automated insulin delivery systems. These systems monitor blood sugar levels and deliver an appropriate dose of insulin when required. Insulin delivery systems like this already exist for the treatment of hospital patients. Many diabetics want to have such systems permanently attached to their bodies.

To Do: Working in Pairs

- Discuss 2 reasons on why Waterfall process could work well for this case study and 2 reasons why it wouldn't work (10 min)

- Class discussion (5 min)

Review

- What is Software Engineering about?
- What is Software Development Life Cycle?
- Can you name any Software Development Life Cycles?
- What are the specific characteristics of Waterfall SDLC?