Week1 – Software Development Life Cycle Processes

What is Software Engineering

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

Software Development Tasks:

1. Requirement analysis
2. Planning
3. Design: high level and detailed
4. Development
5. Testing
6. Deployment
7. Operation and Maintenance

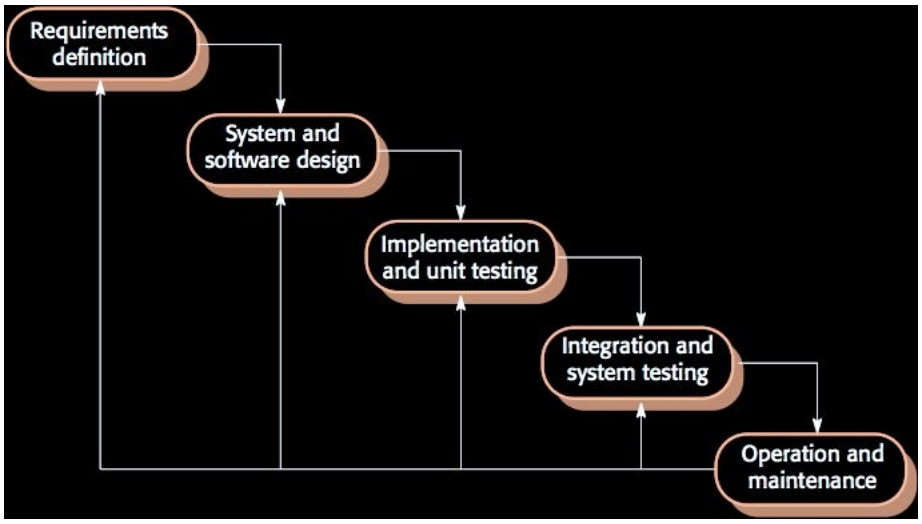
These to-does are combined in various sequences, making up different Software Development Life Cycle (SDLC) models.

SDLC -refers to a methodology with clearly defined processes for creating high-quality software, which is well tested and ready for production use.

Software Development Life Cycle

1. Waterfall
2. Agile
3. V-model
4. Spiral

Waterfall – you must plan and schedule all process activities before start working



1. Requirement
    a. What are the functional elements? What should the system do? And non-functional elements – security, ease of use and response time.
2. Design
    a. What objects, databases, services/servers should we create?
    b. How are they structured?
3. Implementation
    a. Parellel working/code sharing?
    b. API partition and firewalling
    c. Versioning/ integration, config management
    d. Development environments and automated build
    e. Automated testing
    f. Docs and traiing material
4. Verification and Validation (building it right vs building the right thing)
    a. Verification: check the software/service complies with a requirments, constraints and regulations
        i. Demonstrate the system meets specifications
    b. Validation: does this meet the needs of the customers/stakeholders?
        i. Demonstrate the system meets user needs
    c. Can pass the verification but fail validation
    d. Involve checking, reviewing, evaluating & testing
5. Operation and maintenance

    a. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

    b. Making these changes (software maintenance) may involve repeating previous process stages

- **Advantage** vs **Disadvantage**

Simple to use	Software is ready only after the last stage is over
Every phase has a defined result and process review	High risks and uncertainty
Development stages go one by one	Misses complexity due to interdependence of decisions
Perfect for projects with clear and agreed requirements	NOT suited for long term projects where requirement will CHANGE
Easy to determine key points in the development cycle	Hard to measure progress of each stage while still in the development
Easy to classify and prioritise tasks	Integration is done at the very end. Hard to identify problems in advance

Week 2 – Agile Software Development

Agile – a way of thinking about software development – 4 key values. Not to disregard processes, tools and docs

1. Individuals and Interactions **OVER** processes and tools
2. Working software **OVER** comprehensive documentation
3. Customer collaboration **OVER** contract negotiation
4. Responding to change **OVER** following a plan

12 Agile Principles

Satisfy clients' needs	**Satisfy coders' needs**
The highest priority is satisfying the client (by delivering working software early and continuously)	**Work at a steady, sustainable pace (no heroic efforts)**
Embrace change (even late in the development cycle)	**Rely on self-organising teams**
Collaborate every day with the client	**Teams reflect regularly on their performance**
Use face to face communication	**Progress is measured by the amount of working code produced**
Deliver working software	**Continuous attention to**

frequently	**technical excellence**
	Minimise the amount of unnecessary work
	Build teams around motivated individuals

Agile Methods - various approaches that adhere to Agile values and principles

- **Popular agile methods**

1) Extreme Programming (XP) (the two co-creators were signatories of the manifesto)
    - **Ethos**
        - Simple design – use the simplest way to implement
        - Sustainable pace – constant effort and manageable
        - Coding stardards – team follow an agreed style and format
        - Collective ownership – everyone owns the code
        - Whole team approach – everyone is included in everything
    - **Practices**
        - **Pair programming**
            - Helm – keyboard and mouse
            - Tactician – think about implications and potential problems
            - The pair does not own the code, anyone can change, the pairings should evolve at any time.
            - 15% increase in development-time costs but improves design quality, reduces defects, reduces staffing risk, enhances technical skills, improves team communications and is considered more enjoyable at statistically significant levels.
        - **Test driven**
        - Small release – deliver frequently and get feedback from client
        - Continuous integration – ensure the system is operational
        - Refactor the system when things get messy

2) Test-driven development (creator was a signatory)

- Tests are written before any code and they drive all development
- A programmer's job is to write code to pass the tests
- If there's no test for a feature, then it is not implemented
- Tests are requirements of the system
    - Benefits
        - Code coverage – all code has at least one test
        - Simplified debug
        - System doc – tests already describe what the code should be doing

3) Kanban
    - A flexible to do list
    - Issues progress through various states from "To do" to "Done"
        - E.g. 'To Do" -> "In progress" -> "Done" (common to have 3 columns)
    - Example Jira Kanban Board
    - Column = vertical *
    - Swimlines = horizontal*

4) Scrum (the two co-creators were signatories)
    - A project management approach
        - The scrum – a standup daily meeting of the entire team
        - Scrum master -team leader
        - Sprint – a short and rapid development iteration
        - Product backlog – to do list of jobs that need doing
        - Product owner – the client

Problems with Agile

- Hard to draw up legally binding contracts - a full specification is never written in advance
- Good for green-field development when you have a clean slate and are not constrained by previous work. However, it's not so effective for brownfield development which involves improving and maintaining legacy systems.
- Works well for small co-located teams, but what about large distributed development ?
- Relies on the knowledge of developers in the team but what if they aren't around (holidays, illness, turnover) ?

Week 3 –Requirement Engineering

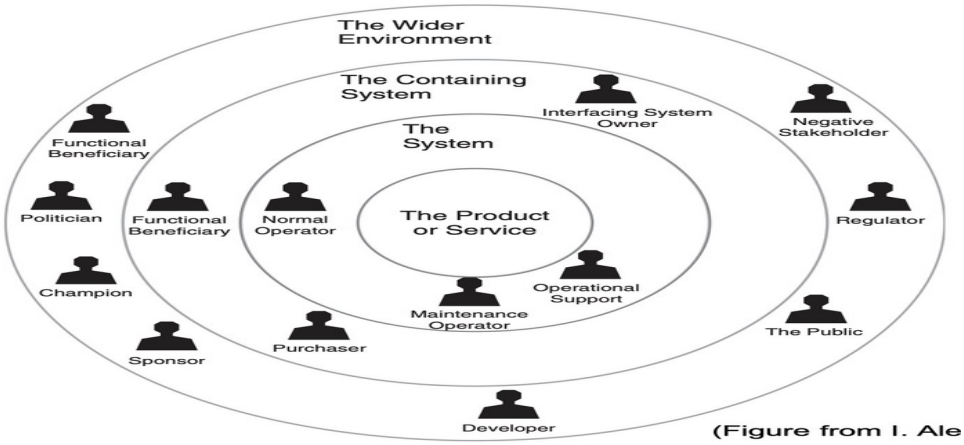
System requirements specify a system, not in terms of system implementation, but in terms of user observation. Requirements record description of the system features and constraints.

- **Functional requirements** specify user interactions with the system, they say **what** the system is supposed to do:
  - Statements of services the system should provide
  - How the system should react to particular inputs
  - How the system should behave in particular situations
  - May also state what the system should NOT do
- **Non-functional requirements** specify other system properties, they say **how** the functional requirements are realised:
  - Constraints ON the services or functions offered by system
  - Often apply to whole system, not just individual features

Requirements are communication mechanism.

Analysing requirements in 4 stages below:

1. Identify stakeholders



(Figure from I. Ale

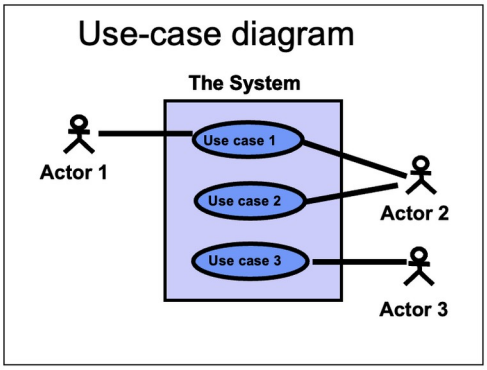
   - Template – onion model
   - Clients
   - Documentation e.g. org chart
   - Anlaysing context of project
   - Surrogate stakhodlers (legal, mass production user)
   - Negative stakholders?

2. Identify top-level user needs (functional requirements)
   - Identify User Stories
     - *As a < type of user >, I want to < some goal >, so that < some reason >*
   - System Behaviour
     - It is how system acts and reacts, comprising actions and activities of a system
     - System behaviour is captured in use cases, which describe the interactions between the system and (part of)its environment

- **Use-case model**
  - Describe the functional requirments of a system in terms of use cases
    - Links stakeholder needs to software requirements
    - Serving as planning tool
    - CONSISTS of ACTORS and USE CASES
      - Use case model comprises of 1) Use case diagram(visual) and Use-case specification (text)

Use-case diagram

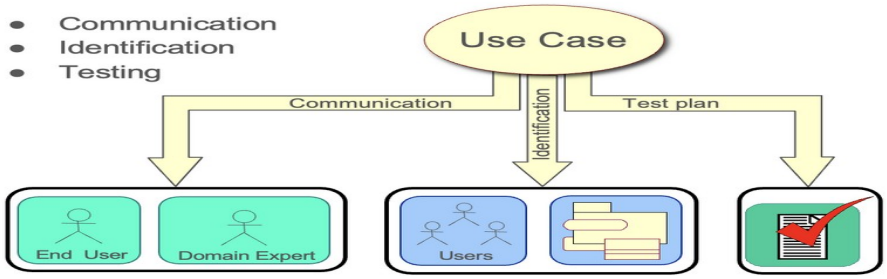
- Shows a set of use cases and actors and their relationships
- Defines clear boundaries of a system
- Identifies who or what interacts with the system
- Summarizes the behavior of the system



- Benefit of Use-Case Model
  - Communication
  - Identification
  - Testing

What Are the Benefits of a Use-Case Model?

- Communication
- Identification
- Testing



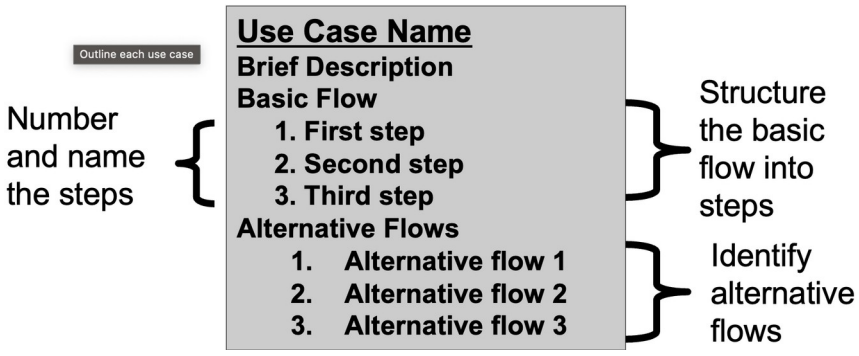
- Concepts in Use Case Modelling
  - Actor: represent anything that interacts with system
    - Represent roles a user of the system can play
      - Human, machine or **another system**
    - They can interchange information with the system
      - Can give or receive information
    - Actors are not part of system, but are EXTERNAL
  - Use case: describes a sequence of events, performed by the system, that yields an observable result of value to a particular actor

- A use case models dialogue between one or more actors and the system
- A use case describes the actions the system takes to deliver something of value to the actor
- A use case is initiated by an actor to invoke a certain functionality in the system.

- Actor and system boundary
  - Everything beyond the boundary that interacts with the system is an instance of an actor

- Extension Relationship in Use case <<extend>>
  - Specify how behaviour of the extension Use Case "e" **can be inserted** into the behaviour of the base use case "b".
- Include Relationship in Use Case <<include>>
  - Specifies how the behaviour of the included Use Case "p" c**ontributes** to the behaviour of the base use case "b".

3. Break down stories into individual steps/refine requirements
   - Use case specification
     i. A requirements document that contains the text of a use case, including:
        1. A description of the flow of events describing the interaction between actors and the system
        2. Other information, such as (Preconditions, Postconditions, Special requirements, Key scenarios, Subflows)
   - Outline each use case
     i. An outline captures use case steps in short sentence, organised sequentially



ii.

   - Flow of events
     i. Is a sequence of steps
     ii. A basic flow = successful scenario from start to finish
     iii. But many alternative flows (regular variants, odd cases, exceptional (error)flows

   - What is a Use Case Scenario

i. An instance of use case
ii. An ordered set of actions from the start of a use case to one of its end point

- Check points for use cases
  i. Each use case is independent of the others
  ii. No use cases have very similar behaviors or flows of events
  iii. No part of the flow of events has already been modeled as another use case

4. Specify atomic requirements
   - Quality Attributes of Requirements
     i. Consistency: Are there conflicts between requirements ?
     ii. Completeness: Have all features been included ?
     iii. Comprehenability: Can the requirement be understood ?
     iv. Traceability: Is the origin of requirement clearly recorded ?
     v. Realism: Can the requirements be implemented given available resources and technology ?
     vi. Verifiability: Can requirements be “ticked off” ?
        1. Unverifiable example:
           a. *The system must be easy to use by waiting staff and should be organised so that user errors are minimised.*
        2. Verifiable example:
           a. *The system must be easy to use by waiting staff and should be organised so that user errors are minimised.*
   - Requirements Elicitation Techniques (gather and define requirements)
     i. Interviews
     ii. Observations
     iii. Surveys
     iv. Current documentation
     v. Similar products and solutions
     vi. Co-design
     vii. Prototyping