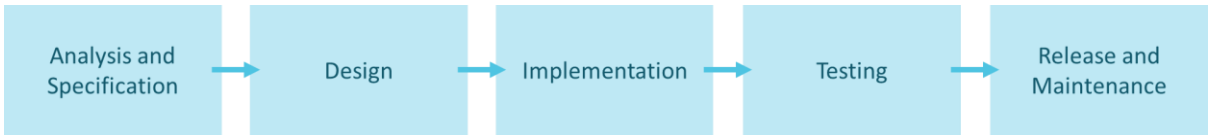
1. Software Engineering Fundamentals

COMP1531 • KC Notes

1.1 Software

- **Software**: a program that tell a computer how and what tasks it needs to perform
    - **Application software**: database, spreadsheet, browser, game
    - **System software**: deals with computer or computer devices, e.g. Windows, anti-virus
- **Software Engineering**: The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches
    - Learn and understand the problem domain (what you build a solution for)
    - Deliver value to the customer

1.2 Software Engineering Life-Cycle



1. **Analysis and Specification**: discovery of the 'system-to-be': a list of features
    - Understand the **problem definition**, including **scope, system's services** (behavioural characteristics)
    - **Abstract** the problem to define a **domain model** (structural characteristics)
    - Identify **functional** (inputs and outputs) **and non-functional** (performance, security, maintainability) **requirements**
    - Done using use-case modelling, user stories
2. **Design**: finding **how to implement all customer requirements**
    - Chunking into symbols and patterns
    - Done using UML symbols (class, component, deployment diagrams)
3. **Implementation**: encoding the design into a programming language to deliver the product
4. **Testing**: Verification that our **system is correct and realises the goals**
    - **Unit tests**, testing individual components
    - **Integration tests**, testing the whole system
    - **User acceptance tests**, verifying the system achieves customer requirements
5. **Operation and Maintenance**
    - Run the system, fixing defects, adding new functionality

1.3 Incremental and Iterative Methods

- Software development is intangible, flexible and complex
    - Can be modified radically during the development process, hard to visualise, composed of many components
- Software Development Method: a prescriptive process that mandates a sequence of tasks
- **Elaborate processes**: rigid, plan-driven, documentation heavy methodologies
- **Iterative processes**: develops increments of functionality and repeated with a feedback loop

1.4 Waterfall Model

- Traditional, sequential life cycle model, plan-driven development
- **Detailed planning**, problem is identified, documented and designed
- Simpler to manage due to **project visibility** – there is a better control over the processes because each phase is completed one at a time
- **Stable requirements and product statement** (e.g. NASA shuttle)
    - 'Mission critical' – strong focus on details
- Can also include **quality gates/completion criteria checks** ensure that the project meets a high and specified standard
- Drawbacks:
    - No working software until late in the lifecycle
    - Not flexible, does not support refinement of customer requirements
    - Incurs a larger management overhead

1.5 Rational Unified Process (RUP)

- Iterative model with four phases, a formal cycle but has concurrent disciplines:
    1. **Inception**: scope the project, identify major players, resources, architecture, risks and costs
    2. **Elaboration**: understand problem domain, analysis, evaluate required architecture and resources
    3. **Construction**: design, build and test software
    4. **Transition**: release software to production
- **Serial in the large and iterative in the small**
    - Phases occur in a serial (in series) manner but is done iteratively (day-to-day)
- RUP is organised in **disciplines**
    - **Development disciplines** – developing models, requirements, engineering the blueprint, testing and releasing
    - **Support disciplines** – configuration and change, project management, environment
    - **Enterprise disciplines** – portfolio management, people, enterprise administration

1.6 Agile

- **Individuals and interactions** > processes and tools
    - Sustainable development, maintain a constant pace indefinitely
    - Team reflects on how to be more effective and adjust behaviour accordingly
- **Working software** > comprehensive documentation
    - Satisfy customer through early, continuous delivery of valuable software
    - Simplicity – maximise the amount of work not done
- **Customer collaboration** > contract negotiation
    - Conveying information done by face-to-face conversation, business people and developers work together throughout the project
- **Responding to change** > following a plan
    - Welcome changing requirements, software delivered frequently
- Drawbacks:
    - Daily stand-up meetings and close collaboration makes outsourcing difficult, difficult and costly for geographically separated clients and developers
    - Does not suit clients who want contracts with firm estimates and timetables
    - Relies on small self-organised teams
    - Lack of documentation means it is difficult to maintain after team changes