Software Engineering

Dr. Yih-Ling Hedley
Email: aa0817@coventry.ac.uk

Software Engineering: Introduction

- An **engineering discipline** which is concerned with all aspects of **software** production, including:
  - Software development process
  - Project management
  - Tools, methods and theories adopted
- To adopt a **systematic** and **organised** approach to producing high-quality software
- A quality process to produce a **high-quality product, on time** and **to budget**.
  - To apply appropriate methods and tools to find the solutions to problems and constraints

A Software Process

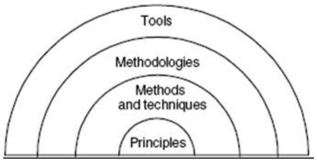
- **A set of activities** to produce a software product
- Generic activities to all software processes are:
  - **Specification** – functionality of software and its development constraints
  - **Development** - production of the software
  - **Validation** - checking that the software is what the customer wants
  - **Evolution** - changing the software in response to changing demands

Software Engineering: Objectives

- To produce software:
  - **on time**
  - **to budget**
  - with **required quality:** e.g.
    - **Maintainability/Modifiability**: Software must evolve to meet changing needs
    - **Dependability/reliability**: Software must be trustworthy, reliable
    - **Efficiency**: Software should not make wasteful use of system resources
    - **Usability/Suitability**: Software must be usable by the users for which it was designed

Software Engineering: Principles 1

- Principles form the basis of methods, techniques, methodologies and tools and become practice through methods and techniques
  - often **methods and techniques are packaged in a methodology**
  - **methodologies can be enforced by tools**



SE: Principles 2

- **Software Engineering Principles** (Ghezzi, C. et al)
  - Principles (rules) of software engineering for the success of a software project.
    - Principle 1. **Separation of Concerns**
    - Principle 2. **Modularity**
    - Principle 3. **Incrementality**
    - Principle 4. **Abstraction**
    - Principle 5. **Generality**
    - Principle 6. **Anticipation of Change**
    - Principle 7. **Rigor and Formality**

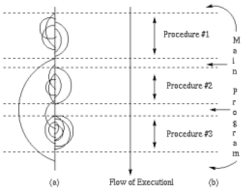
Source: C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering, Second Edition 2002

SE Principles: Separation of concerns

- **Separation of concerns**
  - Refers to a given problem that involves **different areas of concerns** to be identified and separated to deal with complexity, and to achieve required engineering quality)
    - Minimizes interdependence
    - Increases reusability

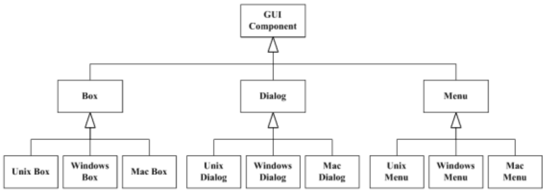
SE Principles: Modularity 1

- **Modularity**
  - a specialization of principle of **separation of concerns**
  - separating software into **components/modules**, according to **functionality and responsibility**
  - ignore details of other modules when dealing with a module



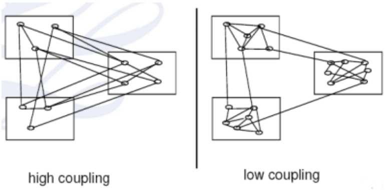
SE Principles: Modularity 2

- **Modularity : Cohesion**
  - the degree to which the internal contents of a module are related



SE Principles: Modularity 3

- **Modularity: Coupling**
  - coupling (the degree to which a module interacts with or depends upon other modules)



SE Principles: Incrementality

- **Incrementality**
  - Process proceeds in increments:
    - delivers **subsets of a system** at early stages or a prototype for feedback from the clients or expected users, from which new features are added incrementally
    - deals with **functionality** first and then considers performance

SE Principles: Abstraction

- **Abstraction**
  - expresses the concept only in terms of what is relevant, everything that's not relevant is hidden from the user
  - identifies the important aspects of a phenomenon and ignore its details
    - The entities or objects in the system have a **high level summary view** (an abstract view) of what data or a method is, not the details.
  - also means that to separate the b**ehaviour of software components (what it does) from their implementation (how it does it)**

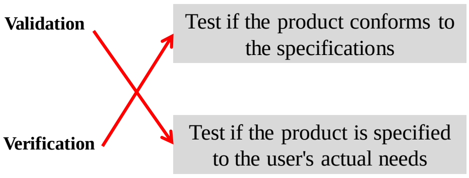
SE Principles:
Generality and Anticipation of Change

- **Generality**
  - To discover an instance of a more general problem when solving a problem, so that the solution can be **reused** in other cases (e.g. use of patterns)
- **Anticipation of Change**
  - Anticipates potential future changes with ability to support software evolution

SE Principles:
Rigor and Formality 1

- **Rigor and Formality**
  - **Rigor**
    - **systematic test data derivation** for a software product;
    - **rigorous documentation of development steps** helps project management and assessment of timeliness for a software process
  - **Formality**
    - software process is driven and evaluated by **mathematical laws**, e.g. mathematical analysis of program correctness for a software product

Validation and Verification:
Activity 1


Validation

Verification

Test if the product conforms to the specifications

Test if the product is specified to the user's actual needs

Validation and Verification:
Activity 1 Feedback

- **Validation**
  - Have we produced the right product? i.e., " is the product specified to the user's needs?
  - E.g. source code inspection (static) for a software product against specific test cases (dynamic).
- **Verification:**
  - Have we produced what we were trying to make? i.e., " does the product conform to the specifications? "
  - E.g. Does the product satisfactorily meet all use cases?

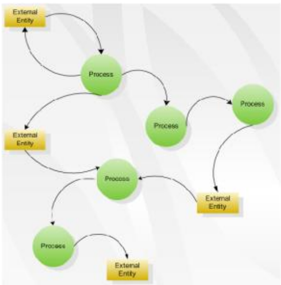
Methodologies 1

- A **methodology** defines an **approach** (i.e. **paradigm**) to be used in software development to produce high-quality and cost-effective software in a systematic manner
- **Software development approach/methodology**: -
  - Structured (function-oriented): 1970s
  - Object-oriented:1990s –
    - First and Second Generations: hybrid (partly structured)
    - Third Generation: integrated – Unified Process (UP), e.g. IBM Rational Unified Process (RUP)
  - Agile: mid 1990s

Source: U. Khan et al., Object-Oriented Software Methodologies: Roadmap to the Future, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 2, 2011

Methodologies 2

- **Structured/function oriented**:
  - **Processes** manipulate **data** and show how they transform data objects that flow through the system
  - Considers the processes and data separately

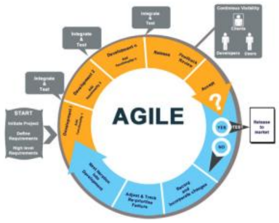


Methodologies 3

- **Object-orientated**
  - Based on the concept of **objects** in which data is encapsulated with the functions that act on the data
- **Unified Process**: (object-oriented)
  - provides a framework for object-oriented software engineering using UML
  - a **use case driven**, **architecture-centric**, **iterative** and **incremental** process
  - **Unified Modelling Language (UML):** a standard language for visualising, specifying, constructing and documenting software artefacts

Methodologies 4

- **Agile**
  - Agile processes use feedback, driven by regular tests and releases of the evolving software.



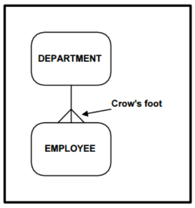
Source: http://www.techjini.com/ourapproach-methodologies.html

Structured Methodology :
Methods and Techniques 1

- **Structured analysis and development methods**
  - **Yourdon structured method** (Constantine and Yourdon): sees **functions** as a design abstraction and identifies the **data-flow** through a system.
  - **Jackson structured programming** (Michael A. Jackson): a design method that concentrated on the structure of **data** and completes the program through continued iterations
  - **Structured analysis** (Tom DeMarco): The function of the system is described by processes that transform the data flows.

Structured Methodology :
Methods and Techniques 2

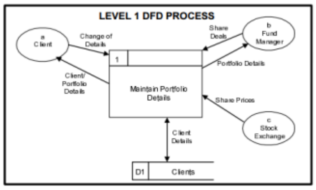
- **Structured methodology - Structured systems analysis and design method (SSADM)** - including:
  - **Logical Data Modelling** : models the data requirements of the system being designed.



Source: P. Biggs A Survey of Object-Oriented Methods

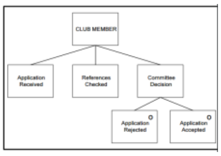
Structured Methodology :
Methods and Techniques 2

- **SSADM** - including:
  - **Data Flow Modelling** : models how data moves around an information system , which examines processes (activities that transform data from one form to another), data flows (routes by which data can flow). data stores and external entities .



Structured Methodology :
Methods and Techniques 2

- **Structured methodology - Structured systems analysis and design method (SSADM)** - including:
  - **Entity Event Modelling**: models the events that affect each **entity and the sequence of the events**, and designs for each event the process to coordinate the sequence of events.



Object Oriented Methodology:

Methods and Techniques 1

- Object Oriented methods
  - **Object-oriented Analysis (OOA):** captures **requirements** around objects, which integrate both behaviours (processes) and states (data) modelled after real world objects that the system interacts with
  - **Object-Oriented Design (OOD):** maps concepts in the analysis model onto the implementation of classes and interfaces. Consider the design of software architectures by applying **architectural patterns and design patterns** with object-oriented design principles

Object Oriented Methodology:

Methods and Techniques 2

- Object Oriented techniques
  - **Object-Oriented Modelling (OOM)**: considered at the analysis level (OOA) and design level (OOD)
  - Models
    - dynamic behaviours, e.g. business logic, processes and use cases
    - static structures e.g. classes and components

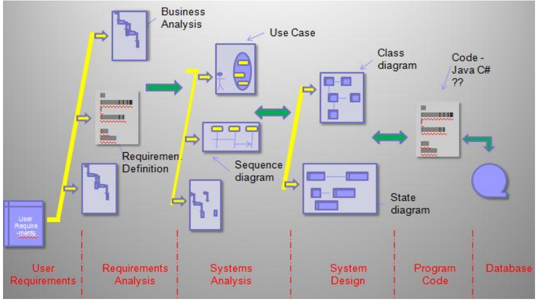
Agile Methodology:

Methods and Techniques

- Agile methods
  - **Adaptive Software Development (ASD)**
  - **Agile Modelling**
  - **Agile Unified Process (AUP)**
  - **Dynamic Systems Development Method (DSDM)**
  - **Extreme Programming (XP)**
  - **Feature-driven Development (FDD)**
  - **Scrum**

Tools: UML Tools

Computer Aided Software Engineering (CASE)



Activity 2

- **Which of the following describes the processes below:**
  - **Decompile executables to get the source code**
  - **Analyse software to produce the models for its design and specification**

Forward Engineering

Reverse Engineering

Re-Engineering

Round-trip Engineering

Activity 2: Feedback

- **Reverse engineering**: in CASE tools
  - **Decompilation**, a type of reverse engineering, to convert executable program code (also object code) into a form of higher-level programming language readable by a human.
  - For source code available, to discover higher-level program poorly documented or no longer valid
  - For no source code available, to discover possible source code , with **clean room design** (a design copy by reverse engineering is recreated without infringing copyrights associated with original design)

Activity 3

- **What functionality in CASE tools that supports the synchronisation of related software artefacts to ensure consistency?**

Forward Engineering

Re-Engineering

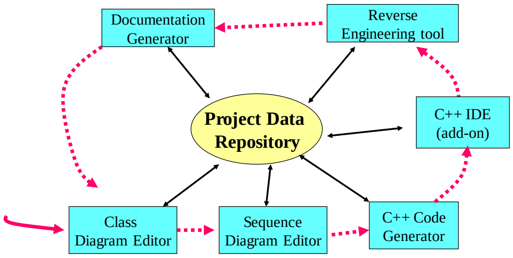
Round-trip Engineering

Activity 3: Feedback

- **Round-trip engineering**: in CASE tools
  - **synchronizes** two or more related software artefacts, such as source code, models, configuration files, and other documents.
    - continuous **alignment between source code and diagram**

CASE: Project Data Repository

- **Round-trip engineering**: in CASE tools



Documentation Generator

Reverse Engineering tool

Project Data Repository

C++ IDE (add-on)

Class Diagram Editor

Sequence Diagram Editor

C++ Code Generator

Activity 4

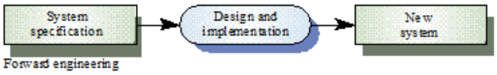
- **Which of the following describes the processes below: 'renovate an existing system from a design that is recovered from its existing source code'?**

Forward Engineering

Re-Engineering

Activity 4: Feedback

- **Forward engineering**: in CASE tools
  - **From design information** recovered from existing source code, based on which to **reconstruct** the existing system to improve its overall quality or performance
    - E.g. Given a UML design model, generate the corresponding code.



Source: Ian Sommerville, Software Engineering

Re-engineering

- **Re-engineering:** Reorganising and **modifying** existing software systems to make them more maintainable, through:
  - **Reverse engineering**: analysing program code to create a high-level program representation
  - **Code restructuring**: analysing source code and revising any violations of structured programming practices
  - **Forward engineering**: from recovered design models by reverse engineering to reconstitute the existing system for improved quality or performance

Source: Ian Sommerville, Software Engineering