ACM/IEEE Code of Ethics

Institute of Electrical and Electronics Engineers (IEEE)
Association for Computing Machinery (ACM)

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Code of ethics - preamble

- Preamble
  - The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a consistent code.
  - Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Code of ethics - principles

1- PUBLIC
  - Software engineers shall act consistently with the public interest.

2- CLIENT AND EMPLOYER
  - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3- PRODUCT
  - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

Code of ethics - principles

4- JUDGMENT
  - Software engineers shall maintain integrity and independence in their professional judgment.

5- MANAGEMENT
  - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6- PROFESSION
  - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

Code of ethics - principles

7- COLLEAGUES

– Software engineers shall be fair to and supportive of their colleagues.

8- SELF

– Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

The rationality of these principles is discussed in the book.

Dilemma Ex.

- A particularly difficult situation for professional engineers arises when their employer acts in an unethical way. Say a company is responsible for developing a safety-critical system and, because of time pressure, falsifies the safety validation records. Is the engineer's responsibility to maintain confidentiality or to alert the customer or publicize, in some way, that the delivered system may be unsafe?
- The appropriate ethical position here depends entirely on the views of the individuals who are involved…

Dilemma Ex.

- Another ethical issue is participation in the development of military and nuclear systems. Some people feel strongly about these issues and do not wish to participate in any systems development associated with military systems.
- Others will work on military systems but not on weapons systems.
- In this situation, it is important that both employers and employees should make their views known to each other in advance.
- Where an organization is involved in military or nuclear work, they should be able to specify that employees must be willing to accept any work assignment."Safety comes first"

Software Process Models

Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution

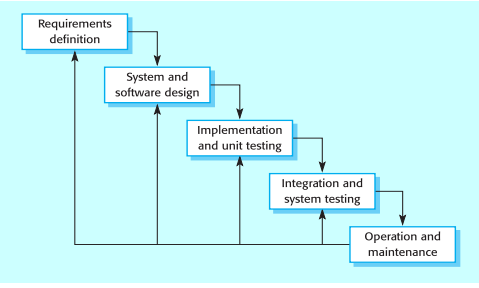
The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design;
  - Validation;
  - Evolution.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based (Reuse Oriented) software engineering
  - The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.

Waterfall model



Waterfall model phases

1. Requirements analysis and definition

- The system's services, constraints, and goals are established by consultation with system users.
- They are then defined in detail and serve as a system specification.

2. System and software design

- The systems design process allocates the requirements to either hardware or software systems
- establishing an overall system architecture

Waterfall model phases

3. Implementation and unit testing

- the software design is realized as a set of programs or program units.
- Unit testing involves verifying that each unit meets its specification.

4. Integration and system testing

- The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met.
- Then software system is delivered to the customer.

Waterfall model phases

4. Operation and maintenance

- 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.
- this is the longest life cycle phase. The system is installed and put into practical use.
- **The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.**

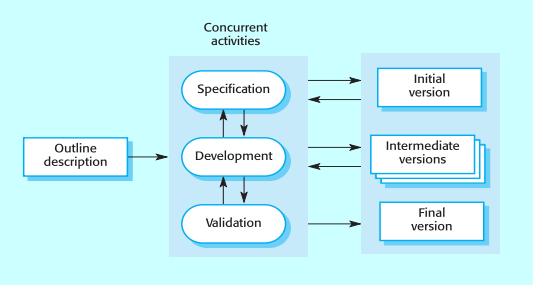
Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

Evolutionary (Incremental) development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

Evolutionary development



Evolutionary (Incremental) development

- Incremental development reflects the way that we solve problems.
- We rarely work out a complete problem solution in advance but move toward a solution in a series of steps, backtracking when we realize that we have made a mistake.
- By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed.
- Each increment or version of the system incorporates some of the functionality that is needed by the customer.

Benefits of Evolutionary Development

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Evolutionary development: Applicable

– For small or medium-size interactive systems;
– For parts of large systems (e.g. the user interface);
– For short-lifetime systems.

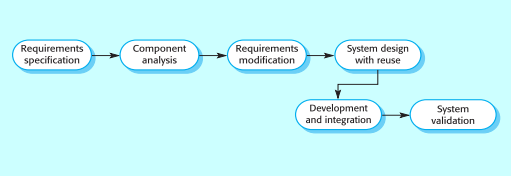
Evolutionary development: Problems

- **Lack of process visibility:** Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- **Systems are often poorly structured**
- **Special skills** (e.g. in languages for rapid prototyping) may be required.
- The problems of incremental development become acute for large, complex, long-lifetime systems, where different teams develop different parts of the system.

Component-based (Reuse-oriented) software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- This approach is becoming increasingly used as component standards have emerged.

Reuse-oriented development



Reuse-oriented Stages

- **Component analysis;** Given the requirements specification, a search is made for components to implement that specification
- **Requirements modification;** the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components.
- **System design with reuse;** the framework of the system is designed
- **Development and integration;** and the components and COTS systems are integrated to create the new system

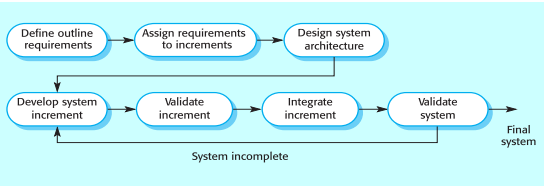
Process iteration

- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
- Iteration can be applied to any of the generic process models.
- Two (related) approaches
  - Incremental delivery;
  - Spiral development.

Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development



Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.