SOFTWARE PROCESS MODELS

Software Process Models

Most process models are based on one of three general models:

- The Waterfall Approach
- Iterative Development
- Component Based Software Engineering

Waterfall Approach

Represents activities as separate process phases:

- Requirements specification
- Software design
- Implementation
- Testing
- etc

After each phase is defined it is 'signed-Off' before the next phase can begin.

Iterative Development

- Interleaves the activities of specification , development and validation.
- Initial system ***rapidly*** developed from abstract specifications
- System refined based on customer input until customer requirements are satisfied
- Only then can the system be delivered

Component Based Software Engineering (CBSE)

- Assumes parts of the system already exist.
- System development focuses on integrating these existing parts as opposed to developing them

Software Engineering Costs

- Software costs often dominate computer system costs.
- The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop.
- For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

- Roughly 60% of costs are development costs, 40% are testing costs.
- For custom software, evolution costs often exceed development costs.
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance, system reliability and security.
- Distribution of costs depends on the development model that is used.

Costs are distributed across the different activities in the process

Distribution depends on the process used and the type of software being developed

Real-time systems require more extensive testing and validation than web based systems

Real-time systems:

- Must guarantee response within strict time constraints
- Real-time response times are in the order of milliseconds and sometimes microseconds

A real-time system may be one where its application can be considered (within context) to be mission critical

The anti-lock brakes (ABS) on a car are a simple example of a real-time computing system.

The real-time constraint in this system is the time in which the brakes must be released to prevent the wheel from locking.

The term Web-Based system refers to those applications or services that are resident on a ***server*** that is accessible using a ***Web browser*** and is therefore accessible from anywhere in the world via the Web.

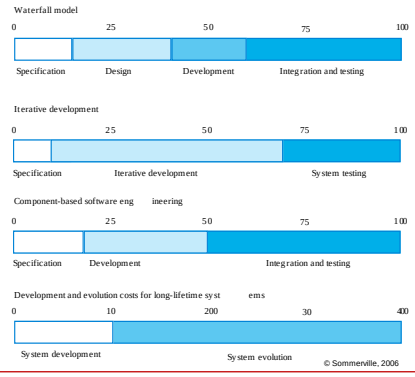
A web information system consists of:

- One or more web applications
- Specific functionality-oriented components
- Information components
- Non-web components.

The Web browser is typically used as a front-end with a database as back-end

Each generic approach to software development has a different profile of cost distribution across the process activities.

The following diagram shows how costs are spent on the different process activities



Costs – Waterfall Model

- Costs for phases are measured separately
  - Specification
  - Design
  - Implementation
  - Integration
- Integration and testing the most expensive development activity
- Normally 40% of total development costs
- Likely to be 50% for critical systems

Costs – Iterative Development

- The interleaving of specification, design and development activities make it hard to say where each activity ends
- Specification costs reduced due to high level specification produced ***prior*** to development
- Specification, design, implementation, integration and testing performed in parallel
- An independent system test is required when implementation complete

Costs – CBSE

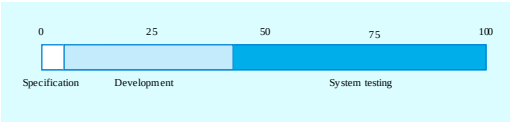
- Only been used for a short period of time
- No accurate figures for cost of different development activities
- Is known that development costs are lower than integration and testing costs
- Integration/testing costs increased as must ensure that components meet their specification and integrate with other components as expected

Evolution costs vary depending on the type of system

For long-life systems (10+ years) evolution costs are likely to be 3 or 4 times the development costs

Smaller systems tend to have a shorter lifetime and reduced evolution costs

For bespoke software (for PC's) this distribution of costs is different:



What are Software Engineering Methods?

A structured approach to software development which aims to produce high-quality software in a cost-effective way

- 1970's: Structured Analysis methods
  Functional components; methods still used
- 1980's/1990's Object oriented
- Both approaches integrated → UML

Includes system models, notations, rules, design advice and process guidance.

- Model descriptions - Descriptions of graphical models which should be produced
- Rules - Constraints applied to system models;
- Recommendations - Advice on good design practice
- Process guidance - What activities to follow.

What is CASE?

Computer-Aided Software Engineering

Software systems that are intended to provide automated support for software process activities.

- Requirements analysis
- System modelling
- Debugging and testing

CASE tool may include a code generator

CASE systems are often used for method support.

- Upper-CASE - Tools to support the early process activities of requirements and design;
- Lower-CASE - Tools to support later activities such as programming, debugging and testing.

Attributes of Good Software

The software should:

- Deliver the required functionality/performance
- Should be maintainable, dependable and acceptable.

Maintainability

Software must evolve to meet changing needs

Change is unavoidable in the business world

Dependability

Software dependability must include reliability, security and safety.

Must be trustworthy. Must not cause physical or economical damage in the event of a system failure

Efficiency

Software should not make wasteful use of system resources e.g. memory.

Responsiveness, processing time, memory utilisation

Usability

Software must be understandable, usable and compatible with other systems.

Must be usable without undue effort by intended user.

Should have appropriate user interface.

Must have adequate documentation/support.

Key Challenges

- Heterogeneity

  Developing techniques for building software that can cope with heterogeneous platforms and execution environments (distributed systems)

- Delivery

  Developing techniques that lead to faster delivery of software;

- Trust

  Developing techniques that demonstrate that software can be trusted by its users.

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- ***Confidentiality***

  Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- ***Competence***

  Engineers should not misrepresent their level of competence. They should not knowingly accept work which is beyond their competence.

- ***Intellectual property rights***

  Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.

- ***Computer misuse***

  Software engineers should not use their technical skills to misuse other people's computers.

  Computer misuse ranges from relatively trivial (game playing on an employer's machine, Browsing, FB) to extremely serious (dissemination of viruses, downloading inappropriate material).

ACM/IEEE Code of Ethics

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.

Sommerville, p.15