Lecture 9: Software lifecycle models

Software engineering

The collection of processes, methods, techniques, tools and languages for developing quality operational software.
External quality factors

- CORRECTNESS
- ROBUSTNESS
- INTEGRITY
- EASE OF USE
- REUSABILITY
- EXTENDIBILITY
- PORTABILITY
- EFFICIENCY
- ...

- Correctness:
  - The ability of a software system to perform according to specification, in cases defined by the specification.
- Robustness:
  - The ability of a software system to react in a reasonable manner to cases not covered by the specification.

Software quality factors

Product quality (immediate):
- Correctness
- Robustness
- Integrity
- Ease of use
- Ease of learning

Process quality:
- Timeliness
- Cost-effectiveness

Product quality (long term):
- Extendibility
- Reusability
- Portability
- ...

The Software Engineering problem

Developing software systems that are
- On time and within budget
- Of high immediate quality
- Possibly large and complex
- Extendible
Lifecycle models

- Origin: Royce, 1970, Waterfall model
- Scope: describe the set of processes involved in the production of software systems, and their sequencing
- "Model" in two meanings of the term:
  - Idealized description of reality
  - Ideal to be followed

Models and standards

Capability Maturity Model (CMM)

- Characterization of maturity of the software development model of a company
- Five levels
- Popular with defense contractors, outsourcing companies
- Also: ISO 900x quality standards (International Standards Organization)

The anti-process movement

"eXtreme Programming" (XP), "Agile" methods

- Test-driven development
- Recommended practices, e.g. Pair Programming
- Short iteration cycles

"The revenge of the cubicles"
The waterfall model of the lifecycle

- Feasibility study
- Requirements analysis
- Global design
- Detailed design
- Implementation
- Unit validation
- Subsystem validation
- System validation
- Distribution
- V & V

V-shaped

- Feasibility study
- Requirements analysis
- Global design
- Detailed design
- Implementation
- Unit validation
- Subsystem validation
- System validation
- Distribution
- V & V

Arguments for the waterfall

(After B.W. Boehm: Software engineering economics)

- The activities are necessary
  - (But: merging of middle activities)
- The order is the right one.
The waterfall model

- Feasibility study
- Requirements
- Specifications
- Global design
- Detailed design
- Implementation
- V & V
- Distribution

Problems with the waterfall

- Late appearance of actual code.
- Lack of support for requirements change — and more generally for extendibility and reusability.
- Lack of support for the maintenance activity (70% of software costs?).
- Division of labor hampering Total Quality Management.
- Impedance mismatches.
- Highly synchronous model.

Quality control?

Analysts
Designers
Implementers
Testers
Customers
Impedance mismatches

As Management requested it.
As the Project Leader defined it.
As Systems designed it.
As Programming developed it.
As Operations installed it.
What the user wanted.

*Pre-1970 cartoon; origin unknown*

The Spiral model (Boehm)

Figure from: Ghezzi, Jazayeri, Mandrioli, *Software Engineering*, 2nd edition, Prentice Hall.

The Spiral model

M.C Escher: *Waterfall*
Software Architecture

Chair of Software Engineering

Tasks

- Analysts
- Designers
- Implementers
- Testers

Seamless development (as in Eiffel)

Seamless development:
- Single notation, tools, concepts, principles throughout
- Eiffel is as much for analysis & design as implementation & maintenance
- Continuous, incremental development
- Keep model, implementation and documentation consistent

Reversibility: go back and forth
- Saves money: invest in single set of tools
- Boosts quality

Example classes:
- PLANE, ACCOUNT, TRANSACTION...
- STATE, COMMAND...
- HASH_TABLE...
- TEST_DRIVER...
- TABLE...

Analysis classes

defined class VAT
 inherit TANK
 feature
   in_valve, out_valve: VALVE
   fill is -- Fill the vat.
     require in_valve.open
     deferred ensure in_valve.closed
     empty, is_empty, gauge, maximum, ... [Other features] ...
     invariant
       is_full = (gauge <= 0.97 * maximum) and (gauge <= 1.03 * maximum)
     end
**Seamless development**

- Use consistent notation from analysis to design, implementation and maintenance.
- **Advantages:**
  - Smooth process. Avoids gaps (improves productivity, reliability).
  - Direct mapping from problem to solution, i.e. from software system to external model.
  - Better responsiveness to customer requests.
  - Consistency, ease of communication.
  - Better interaction between users, managers and developers.

**Single model principle**

- Use a single base for everything: analysis, design, implementation, documentation...
- Use tools to extract the appropriate views.

**Generalization**

- Prepare for reuse
- **Possible tasks:**
  - Remove built-in limits
  - Reorganize inheritance hierarchy
  - Abstraction (e.g. introduce deferred classes)
  - Improve documentation
The cluster model

Permits dynamic reconfiguration

Mix of sequential and concurrent engineering

Levels of reusability for a software element

0 - Usable in one program

1 - Usable by programs written by the same author

2 - Usable within a group or company

3 - Usable within a community

4 - Usable by anyone

Nature or nurture?

Two modes:
- Build and distribute libraries of reusable components (business model is not clear)
- Generalize out of program elements

(Basic distinction: Program element --- Software component)
Cluster development

- Bottom-up development: from the most general clusters (providing utility functions) to the most application-specific ones.
- Flexible scheduling of clusters – depending on resources, team experience, customer and management demands. Waterfall is one extreme; “trickle” is the other.
- Sub-lifecycle sequencing: specification, design and implementation, validation, generalization.
- Relations between clusters: each cluster may be a client of lower-level ones.

Quality goals: the Osmond curves

The advice

- Add functionality at constant quality
Complementary material

- OOSC2:
  - Chapter 28: The software construction process

End of lecture