Lecture 9: Software lifecycle models

Software engineering

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

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
- ...
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"

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

- 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.

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.

The Spiral model (Boehm)

Quality control?

- Analysts
- Designers
- Implementers
- Testers
- Customers
Tasks

Analysts

Designers

Implementers

Testers

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.

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...

Analysis
Design
Implementation
V&V
Generalization

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

Analysis classes

def deferred class VAT
  inherit TANK
  feature
    m_valve, out_valve: VALUE
    is -- fill the vat.
    require
      in_valve.open
      out_valve.closed
    deferred
      in_valve.closed
      out_valve.closed
      is_full
    end
    empty, is_full, is_empty, gauge, maximum, ... [Other features] ...
  invariant
    is_full = (gauge >= 0.97 * maximum) and (gauge <= 1.03 * maximum)
    end
The cluster model

- Mix of sequential and concurrent engineering
- Permits dynamic reconfiguration

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.

Levels of reusability for a software element

- 0 - Usable in some 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

Quality goals: the Osmond curves

- Other qualities
- Functionality
- Desirable
- Common

Nature or nurture?

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

The advice

- Add functionality at constant quality
Complementary material

- OOSC2:
  - Chapter 28: The software construction process

End of lecture