Software Architecture

Prof. Bertrand Meyer, Dr. Michela Pedroni

ETH Zurich, February-May 2010

Lecture 2: The software lifecycle
Software lifecycle models

Describe an overall distribution of the software construction into tasks, and the ordering of these tasks

They are models in two ways:

- Provide an abstracted version of reality
- Describe an ideal scheme, not always followed in practice
Lifecycle: the waterfall model

Royce, 1970 (original article actually presented the model to criticize it!)

Succession of steps, with possibility at each step to question and update the results of the preceding step
A V-shaped variant

- Feasibility Study
- Requirements Analysis
- Global Design
- Detailed Design
- Implementation
- Unit Validation
- Subsystem Validation
- System Validation
- Distribution
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.
Merging of middle activities

Feasibility study

Requirements

Specification

Global design

Detailed design

Implementation

V & V

Distribution
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.
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
Lifecycle: “impedance mismatches”

1. As Management requested it
2. As the Project Leader defined it
3. As Systems designed it

4. As Programming developed it
5. As Operations installed it
6. What the user wanted

(Pre-1970 cartoon; origin unknown)
A modern variant
The spiral model (Boehm)

Apply a waterfall-like approach to successive prototypes
“Prototyping” in software

The term is used in one of the following meanings:

1. Experimentation:
   - Requirements capture
   - Try specific techniques: GUI, implementation (“buying information”)

2. Pilot project

3. Incremental development

4. Throw-away development

(Fred Brooks, *The Mythical Man-Month*, 1975: “Plan to throw one away, you will anyhow”).
The problem with throw-away development

Software development is hard because of the need to reconcile conflicting criteria, e.g. portability and efficiency

A prototype typically sacrifices some of these criteria

Risk of shipping the prototype

In the “anniversary” edition of his book (1982), Brooks admitted that “plan to throw one away” is bad advice
Seamless, incremental development

Seamless development:

- Single set of notation, tools, concepts, principles throughout
- Continuous, incremental development
- Keep model, implementation and documentation consistent

Reversibility: can go back and forth

These are in particular some of the ideas behind the Eiffel method
Seamless development

- Single notation, tools, concepts, principles
- Continuous, incremental development
- Keep model, implementation and documentation consistent
- **Reversibility**: go back and forth

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

Prepare for reuse. For example:
- Remove built-in limits
- Remove dependencies on specifics of project
- Improve documentation, contracts...
- Abstract
- Extract commonalities and revamp inheritance hierarchy

Few companies have the guts to provide the budget for this
It seems that the sole purpose of the work of engineers, designers, and calculators is to polish and smooth out, lighten this seam, balance that wing until it is no longer noticed, until it is no longer a wing attached to a fuselage, but a form fully unfolded, finally freed from the ore, a sort of mysteriously joined whole, and of the same quality as that of a poem. It seems that perfection is reached, not when there is nothing more to add, but when there is no longer anything to remove.

(Antoine de Saint-Exupéry, *Terre des Hommes*, 1937)
Reversibility
The cluster model

Cluster 1

Cluster 2
Extremes

“Trickle”

Cluster 1

Cluster 2

“Clusterfall”

Cluster 1

Cluster 2
Dynamic rearrangement

Cluster 1

Cluster 2

Cluster 3

Cluster 4
Order of clusters

Bottom up: start with most fundamental functionalities, end with user interface

Cluster 1

Cluster 2

Cluster 3
Seamless development with EiffelStudio

Diagram Tool
  • System diagrams can be produced automatically from software text
  • Works both ways: update diagrams or update text
    - other view immediately updated

No need for separate UML tool

Metrics Tool
Profiler Tool
Documentation generation tool

...
CMMI: maturity levels*

<table>
<thead>
<tr>
<th>Level</th>
<th>Process Characteristics</th>
<th>Management Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizing 5</td>
<td>Focus is on continuous quantitative improvement</td>
<td>In→Out In→Out In→Out</td>
</tr>
<tr>
<td>Quantitatively Managed 4</td>
<td>Process is measured and controlled</td>
<td>In→Out In→Out In→Out</td>
</tr>
<tr>
<td>Defined 3</td>
<td>Process is characterized for the organization and is proactive</td>
<td>In→Out In→Out In→Out</td>
</tr>
<tr>
<td>Managed 2</td>
<td>Process is characterized for projects and is often reactive</td>
<td>In→Out In→Out In→Out</td>
</tr>
<tr>
<td>Initial 1</td>
<td>Process is unpredictable, poorly controlled, and reactive</td>
<td>In→Out In→Out In→Out</td>
</tr>
</tbody>
</table>

*Slide by Peter Kolb, from our course “Distributed and Outsourced Software Engineering”
Agile/lean methods and extreme programming

- De-emphasize formal process
- De-emphasize design, emphasize refactoring
- Emphasize short-cycled, time-boxed iterative development
- Emphasize the role of tests to guide the development (“TDD”, Test-Driven Development)
- Emphasize the benefit of a second set of eyes: Pair programming
- Emphasize self-organizing teams
- Emphasize customer involvement
Open-source processes

Collaborative, distributed developments

Concentric trust circles

Success with strong project leader (e.g. Linux)

“Given enough eyes, all bugs are shallow”