7.1 The Role of Software Life Cycle Models

7.2 Agile Software Quality
Quality and software life cycle models

- **Classic** software quality management assumes a classic software life cycle model
  - Phased, waterfall-style model with single delivery, or
  - Iterative, evolutionary model with incremental delivery; typical delivery cycle > 6 weeks

- Focus on comprehensive documentation

- Testing and integration are **phases** in the development cycle

- Upfront quality planning
Quality in evolutionary software development

- Exploiting the benefits of shorter feedback cycles
- Less upfront planning required
- Can adapt to changing quality needs
- Otherwise: classic software quality management
Agile development is different

Agile software development is characterized by

❖ Iterative development in fixed-length cycles
❖ Cycle length typically 1-6 weeks
❖ Focus on programming
❖ Little documentation
❖ No or little upfront planning; focus on refactoring
❖ Requirements specified by stories and test cases
❖ Continuous testing and integration
7.1 The Role of Software Life Cycle Models

7.2 Agile Software Quality
Quality in agile software development

❖ Opportunities:
  − Very short feedback cycles
  − Focus on people: quality culture instead of document-based quality management
  − Early prototypes

❖ Problems:
  − Frequent re-validation required
  − Not all quality problems can be fixed by refactoring
  − Real stakeholders have to be represented by product owner or on-site customer representative
Agile quality management

❍ Feedback-oriented development
  ● Customer representative or product owner on site
  ● Small increments – rapid feedback
  ● Continuous integration
  ● Regularly held retrospectives

❍ People-focused quality culture
  ● Quality over functionality
  ● Realistic planning and workload
  ● Joint responsibility for results
  ● Team as a learning organization
  ● Intrinsically motivated developers work faster and better
Agile quality management – 2

- Testing from the very beginning
  - Tests define required system behavior
  - Tests are written prior to coding or in parallel with coding
  - Continuous regression testing

- Catching faults early
  - Inspection of code prior to committing
  - Pair programming (⇒ continuous inspection)
  - Unit testing prior to committing

- Explicit quality improvement
  - Quality improvement refactorings
Quality problems – Architecture

- Growing a system into an architectural mess
- Structure follows people structure instead of problem structure (Conway’s law*)
- Major architectural mistakes cannot be fixed by refactoring

* Conway (1968): How Do Committees Invent?

The new city has been built in a rapid and agile fashion – unfortunately, the settlers forgot to reserve space for streets

Source: Morris: Lucky Luke – Auf nach Oklahoma © Ehapa Verlag
Quality problems – Specification by testing

- Specification by testing
  - focuses on required behavior
  - neglects unwanted behavior

<table>
<thead>
<tr>
<th>Required system behavior</th>
<th>Implemented system behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required, but</td>
<td>Not required, but implemented behavior can be harmful (e.g., for security) and is typically not detected with agile testing</td>
</tr>
<tr>
<td>• not implemented or</td>
<td></td>
</tr>
<tr>
<td>• wrongly implemented behavior is typically detected in agile testing</td>
<td></td>
</tr>
</tbody>
</table>
Quality problems – regulatory compliance

The need for regulatory compliance (for example, in the healthcare or transportation domains) may

- require a full requirements specification
- require classic system testing
- confine agility to agile design and coding
Tooling

Quality-aware agile development is impossible without adequate tools for

- Configuration management
- Continuous integration
- Test automation
- Problem report management
References


