University of Zurich^UZH

Department of Informatics

Martin Glinz

Software Quality

Chapter 7

Quality in Agile Development

7.1 The Role of Software Life Cycle Models

7.2 Quality in Agile Software Development

Quality and software life cycle models

- Classic software quality management assumes a classic software life cycle model
  - Phased, waterfall-style model with single delivery
  - 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 Quality in Agile Software Development

Quality in agile software development

- Opportunities:
  - Very short feedback cycles
  - Focus on people: quality culture instead of document-based quality management
- Problems:
  - Frequent re-validation required
  - Not all quality problems can be fixed by refactoring

Agile quality management

- Feedback-oriented development
  - Customer representative 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 immediately
  - Pair programming (⇒ continuous inspection)
  - Inspection of code 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?

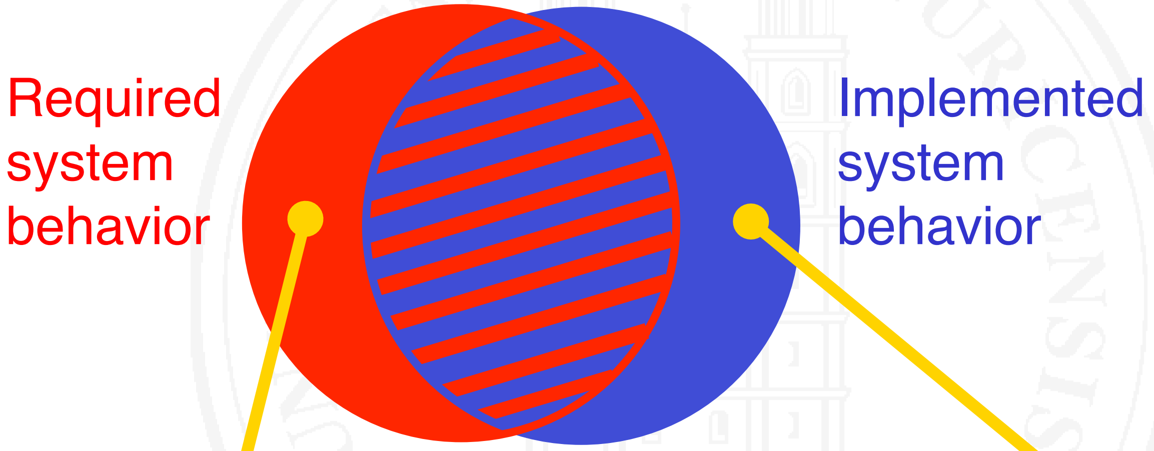
The new city has been built in a rapid and agile fashion – unfortunately, the settlers forgot to reserve space for streets

Die Stadt ist so gut wie fertig – aber etwas wurde vergessen...
Gibt's nicht! Wir müssen ein paar Holzhäuser wieder abreißen und Straßen bauen!
Aber Luke, ist das wirklich nötig? Auf Straßen passieren doch nur Unfälle!
HOTEL
MORRIS
228

Source: Morris: Lucky Luke – Auf nach Oklahoma © Ehapa Verlag

Quality problems – Specification by testing

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



Required, but
- not implemented or
- wrongly implemented

behavior is typically detected in agile testing

Not required, but implemented behavior can be harmful (e.g., for security) and is typically not detected with agile testing

Tooling

Quality-aware agile development is impossible without adequate tools for

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

References

V. R. Basili, A. J. Turner (1975). Iterative Enhancement: A Practical Technique for Software Development. *IEEE Transactions on Software Engineering* **SE-1**(6):390–396.

K. Beck (2002). Test Driven Development by Example. Boston: Addison-Wesley.

K. Beck (2004). *Extreme Programming Explained: Embrace Change*. 2nd edition, Boston: Addison-Wesley.

M. E. Conway (1968). How Do Committees Invent? *Datamation* **14**, 4 (April 1968):28–31.

P. Deemer, G. Benefield, C. Larman, B. Vodde (2010). *Scrum Primer, Version 2.0.* http://www.goodagile.com/scrumprimer/scrumprimer20.pdf

C. Larman, V. R. Basili (2003). Iterative and Incremental Development: A Brief History. *IEEE Computer* **36**(6):47–56.

K. Schwaber (2004). *Agile Project Management with Scrum.* Microsoft Press.

K. Schwaber, J. Sutherland (2012). *Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust.* New York: John Wiley&Sons.

L. Williams, R.R. Kessler, W. Cunningham, R. Jeffries (2000). Strengthening the Case for Pair Programming. *IEEE Software* **17**(4):19–25.

H. Wolf, S. Roock, M. Lippert (2005). *Extreme Programming: Eine Einführung mit Empfehlungen und Erfahrungen aus der Praxis.* (in German). 2nd edition. Heidelberg: dPunkt.