Martin Glinz

Software-Qualität – Ausgewählte Kapitel

Kapitel 3

Qualität bei evolutionärer Entwicklung

Universität Zürich
Institut für Informatik

Charakteristika evolutionärer Entwicklung

- Charakterisiert durch Entwicklung nach einem Wachstumsmodell
  - System wird unteteilt in eine Reihe von Teillieferungen/Iterationen/ Inkremente
  - Jede Teillieferung ist lauffähig
- Klassisches Wachstumsmodell
  - Entwicklung der Teillieferungen nach einem Phasenmodell
  - Anforderungen und Architektur werden dokumentiert
  - Umfang einer Teillieferung typisch 3 Wochen bis 3 Monate
- Agile Software-Entwicklung
  - Fokus auf Programmierung, rudimentäre Dokumentation
  - Anforderungen durch Tests beschrieben
  - Kontinuierliche Integration
  - Umfang einer Teillieferung typisch 1 bis 6 Wochen

Evolutionäre Entwicklung und Qualität

- Ausnutzen von zwei Gesetzmäßigkeiten:
  Der Bedarf an explizitem Qualitätsmanagement wird reduziert durch
  (1) Kurze Rückkopplungszyklen
  (2) Gelebte Qualitätskultur
- Neues Problem: Re-Validierung/Verifikation
  erfordert Testautomatisierung
- Klassisches Wachstumsmodell / große Inkremente
  - Ausnutzen der kürzeren Rückkopplungszyklen
  - Ansonsten weitgehend klassisches Qualitätsmanagement
- Agile Software-Entwicklung / kleine Inkremente
  - Angepasstes Qualitätsmanagement erforderlich
  - basiert stark auf hoher Qualitätskultur

Agiles Qualitätsmanagement

- Sehr kurze Rückkopplungszyklen
  - Kompetenter Kundenvertreter jederzeit verfügbar
  - Kleine Inkremente
  - Kontinuierliche Integration
- Testen von Beginn an
  - Tests definieren das geforderte Systemverhalten
  - Tests entstehen vor oder zusammen mit dem Code
  - Kontinuierlicher Regressionstest
- Fehlervermeidung an der Quelle
  - Programmieren in Paaren ($\Rightarrow$ kontinuierliche Inspektion)
  - Inspektion des von Einzelpersonen geschriebenen Codes
- Explizite Qualitätsverbesserung
  - Refaktorierung dient ausschließlich der Qualitätsverbesserung

Agiles Qualitätsmanagement – 2

- Konfigurationsverwaltung
  - Agile Entwicklung ist kein JeKaMi
  - Konfigurationsverwaltung ist zwingend erforderlich
- Gelebte Qualitätskultur
  - Qualität vor Funktionalität
  - Projektteam als lernende Organisation
  - Gemeinsame Verantwortlichkeit für die Ergebnisse
  - Ermöglicht optimistische Konfigurationsverwaltung
  - Regelmäßige Retrospektiven
  - Realistische Planung und Arbeitslast
- Intrinsische Motivation
  - Selbstmotivierte Personen arbeiten schneller und besser

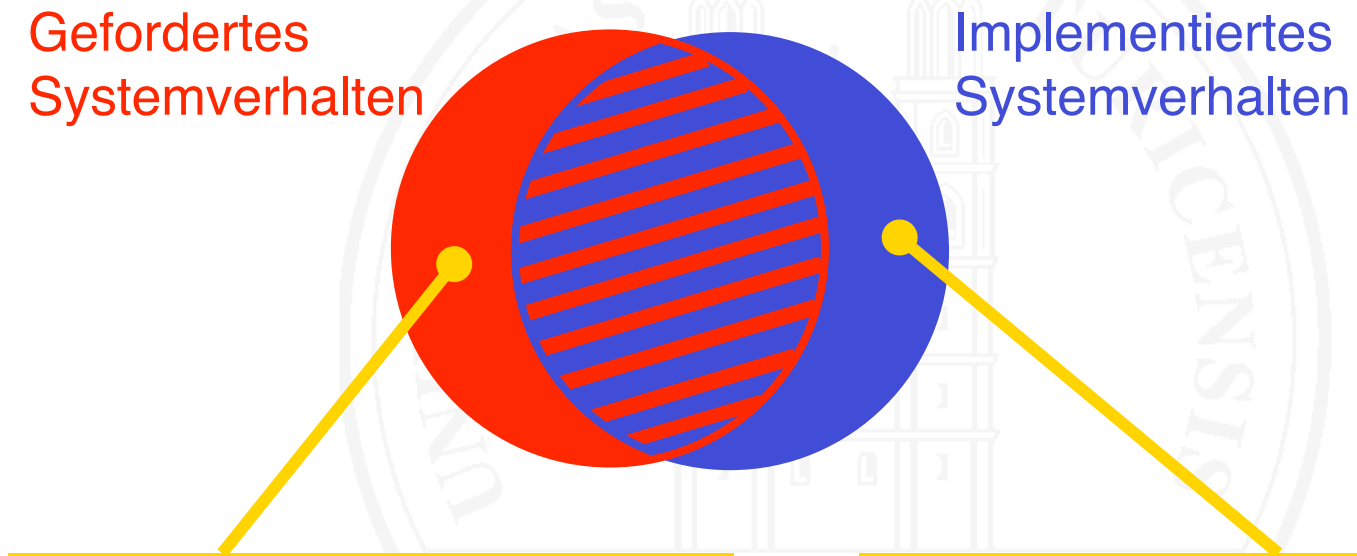
Qualitätsprobleme

- Architektur gewachsen statt gestaltet
  - Gefahr von Komitee-Architekturen
  - Fehler nicht immer mit Refaktorieren kompensierbar

Morris: Lucky Luke – Auf nach Oklahoma

Qualitätsprobleme – 2

- Test einseitig auf gefordertes Verhalten fokussiert



Gefordertes, aber
- nicht implementiertes oder
- falsch implementiertes

Verhalten wird bei agilem Testen meistens entdeckt

Nicht gefordertes, aber implementiertes Verhalten
- kann schädlich sein (Sicherheit!)
- wird bei agilem Testen meistens nicht entdeckt

Werkzeug-Umgebung

- Qualitätsbewusste evolutionäre Entwicklung geht nicht ohne Werkzeuge:
  - Konfigurationsverwaltung, z. B. CVS, SVN
  - Kontinuierliche Integration, z. B. Ant, Maven
  - Testerstellung, z. B. JUnit
  - Regressionstest, z. B. JUnit
  - Problemverwaltung, z. B. Bugzilla, Trac

Literatur

Glinz, M. (2005). *Software Engineering*. Vorlesungsskript, Universität Zürich.

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

Wolf, H., S. Roock, M. Lippert (2005). eXtreme Programming. 2. Auflage. Heidelberg: dPunkt.

Zeller, A., J. Krinke, T. Zimmermann (2003). Open-Source-Programmierwerkzeuge. Heidelberg: dPunkt.