Technische Aspekte des erfolgreichen Testens von Software in Unternehmen

Tim A. Majchrzak

Westfälische Wilhelms-Universität Münster

Agenda

Agenda

Einführung und Motivation

- Die *Softwarekrise* dauert an
- Erhöhung der Softwarequalität ist weiterhin ein wichtiges Ziel
- Zahlreiche Entwicklungsprojekte scheitern
- Aber: Es wird auch erfolgreich entwickelt
- Ergo:
  - Analysieren, was Softwareentwicklung erfolgreich macht
  - Erkenntnisse auswerten
  - Wissen zur direkten Nutzung zurück in die Unternehmen tragen

Agenda

- Projekt angesiedelt im Münsterland
- Hohe Zahl von Unternehmen, die Software entwickelt
- *Institut für Angewandte Informatik* (IAI)
  - der WWU angeschlossen
  - von der IHK unterstützt
- Generelle Unzufriedenheit mit dem Testen führte zu IAI-Projekt
- Ziele: Status quo identifizieren und Handlungsempfehlungen ableiten

Agenda

Vorgehen und Methodik

- Vorgehen:
  - Unternehmen wurden angesprochen
  - Interviewpartner wurden ausgewählt: Manager *und* Techniker
  - Interviewphase: Klärung des Status quo, Erörterung von Stärken und Schwächen
  - Analysephase: Verdichtung und Herausarbeiten von Empfehlungen
  - Erarbeitung eines Ordnungsrahmens
- Grundsätzlicher Forschungsansatz:
  - Qualitative Analyse
  - Semi-sturkturierte Experteninterviews
  - Keine Konkurrenz zu bestehender Literatur, sondern Ergänzung

Agenda

Generelles

- Viele Empfehlungen sind nicht überraschend
- Zum Teil in der Literatur, zum Teil in der Praxis zu finden
- Praktische Verbreitung ist dennoch insgesamt gering
- Literatur ist unvollständig oder unzugänglich

Moderne Entwicklungsumgebung 1 / 2

- Empfohlen wird die Nutzung moderner Entwicklungsumgebungen
- Sinnvoll sind vor allem integrierte Entwicklungsumgebung (IDE), die sich per Plug-in erweitern lassen
- Nutzen dient nicht primär dem Testen
- Vorteile für das Testen ergeben sich durch Entlastung der Tester

- IDE-Nutzung ist weniger verbreitet als gedacht
- Vor allem in größeren Firmen: mitunter archaische Zustände

Moderne Entwicklungsumgebung 2 / 2

- Vorteile moderner IDEs:
  - Syntaxhervorhebung (syntax highlighting)
  - Empfehlungen während der Eingabe (code completion)
  - Partielle Kompilierung
  - Warnungen (simple Semantik-Überprüfung)
  - Code Rules (über Plug-ins)
  - Trace debugging
- Daher: Moderne IDE nutzen, selbst wenn die Einführung erhebliche Aufwände nach sich zieht

Moderne Paradigmen und Frameworks 1 / 2

- Abstraktion und Modularisierung zur Beherrschung erhöhter Komplexität
- Moderne Paradigmen und Frameworks unterstützen dies
- Modulkommunikation nur über Schnittstellen: viel besser zu testen
- Zudem: Rollenteilung, es kommen spezialisierte Tester zum Einsatz
- Weiterhin: Trennung von Geschäfts- und Darstellunglogik

Moderne Paradigmen und Frameworks 2 / 2

- Alte Programmiersprachen und Legacy: fast immer fehlende Dienstorientierung
- Lösung: Erweiterungen, Wrapper, Umstieg

- Wichtig: Effektive und effizienze Nutzung der Programmiersprache
- In der Praxis zum Teil erhebliche Defizite
- Gleichzeitig: Konflikte etwa zwischen Effektivität und Lesbarkeit

Enge Zusammenarbeit der Entwickler 1 / 2

- Enge Zusammenarbeit führt zu höherer Qualität
- Es muss nicht direkt Paarprogrammierung sein
- In der Praxis häufig: isolierte Modulentwicklung
- Vorteile der Zusammenarbeit:
  - Fehler fallen schneller auf
  - Gegebenenfalls Paarprogrammierung für komplexe Module
  - Einhaltung von Standards fällt leichter
  - Wissens- und Erfahrungsaustausch

Enge Zusammenarbeit der Entwickler 2 / 2

- Besonders erfolgsversprechend: gegenseitige Begutachtung (*code reviews*)
  - Fehler am besten direkt beheben: sehr effektive Arbeitsweise
  - Für jedes Modul gibt es mehrere Ansprechpartner
  - Weniger Fehler nach Entwicklungphase
  - Tester können sich auf das Finden schwerwiegender Bugs konzentrieren
- Verantwortlichkeiten sollten klar geregelt sein

Abstimmung Test- mit Produktivsystemen 1 / 2

- Nachteile moderner Architekturen: Unterschiedliche Entwicklungs- und Produktivsysteme
- Konsequenz: Kompatibilitäts- oder Skalierungsprobleme
- Unterschiedliche Schichten auf Zielsystem
- Systemsoftware häufig anders
- Hardware leistungsfähiger
- Lösung: weitgehende Abstimmung

Abstimmung Tests- mit Produktivsystemen 2 / 2

- Häufig sinnvolle Lösungen möglich
- Z. B. „abgespeckte“ Versionen von Server-Software für PCs
- Nutzung zentraler DBMS (wichtig: separate Datenbank)

- Aufdecken seltener (aber schwerer) Fehler auf Zielsystem (z. B. Speicherlecks, Wettbewerbssituationen)
- Wichtig:
  - Ressourcennutzung muss kontrolliert werden
  - Gefährdung produktiver Daten muss ausgeschlossen sein

Integration der Werkzeuge

- Werkzeuge integrieren sich kaum
- Es gibt keine gemeinsamen Schnittstellen oder Formate
- Folge: Synergien werden verschenkt
- Lösung: Auf Integrierbarkeit achten, gegebenenfalls Eigenentwicklungen
- Erster Schritt: Testdokumentation integrieren
- Zweiter Schritt: Regression automatisieren
- Danach: Umfangreiche Statistik und ähnliches
- Fast unbegrenzte Ausbaumöglichkeiten, z. B. Test-Controlling Mitarbeiterzuweisung, Zeiterfassung, Aufgabenverwaltung, Controlling

Agenda

Fazit

- Ergebnisse eines regionalen Projekts
- Ziele: Steigerung der Softwarequalität, Kostensenkungen
- Austausch von Praxis und Forschung
- Hier: Fünf Technische Handlungsempfehlungen mit Programmierfokus
- Projekt schreitet voran:
  - Veröffentlichung der Teilergebnisse
  - Evaluation der Empfehlungen mit den Teilnehmern
  - Eventuell Folgeprojekte

Vielen Dank
für Ihre Aufmerksamkeit!