Antrittsvortrag zur Masterarbeit

Arbeitsgruppe Software Engineering
Institut für Informatik

Praktikable Ansätze zur Verbesserung der Entwicklung eines Altsystems

Nataliya Lashchyk

Gliederung

- Fallstudie
- Motivation
- Ziel
- Konzept

Fallstudie

- über 4 Jahre der Entwicklung
- 154.307 LOC
- Java
- 8 - 12 Entwickler
- ~13 % Testabdeckung

Legacy System

Was ist ein Legacy System?

“Der Begriff Altsystem (*engl*. legacy system) bezeichnet in der Informatik eine etablierte, historisch gewachsene Anwendung im Bereich Unternehmenssoftware.”

Wikipedia

“To me, legacy code is simply code without tests”

Michael C. Feathers

“Some crap made by someone else”

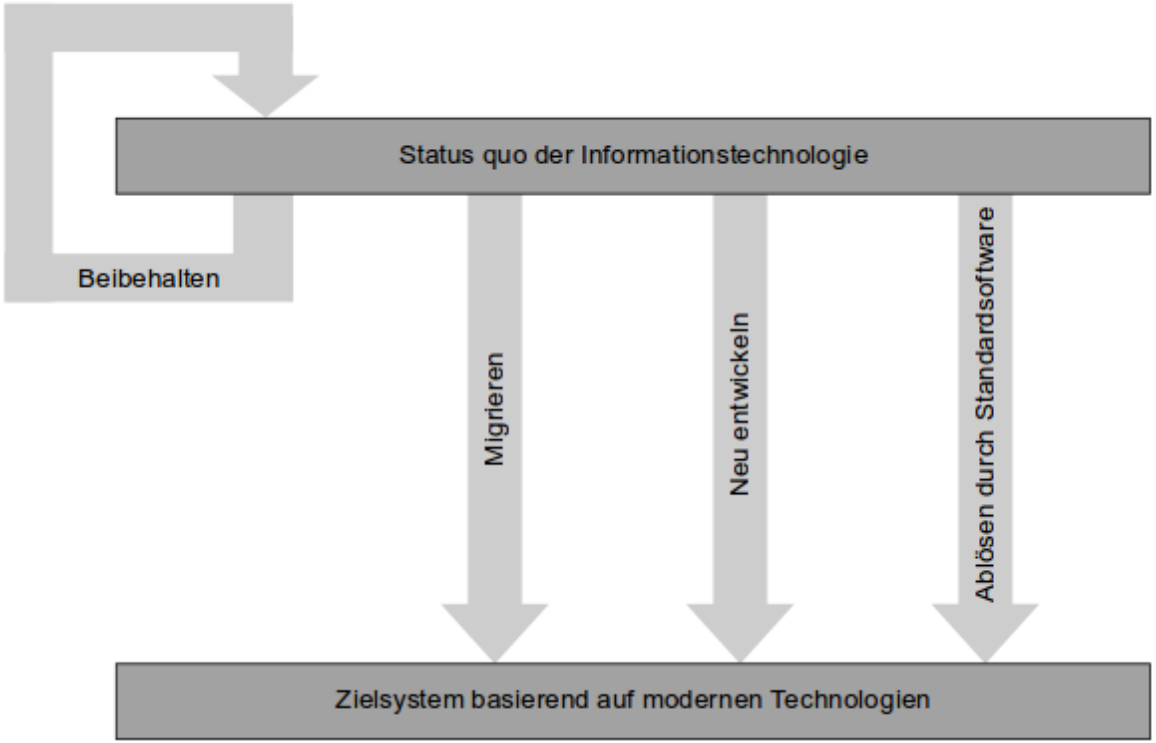
Entwickler, Entwickler, Entwickler

Bestehende Probleme

- Hohe Komplexität
- Unzureichende Dokumentation
- Mangel an Expertenwissen
- Hohe Kosten für Wartung und Weiterentwicklung

Auswahl einer geeigneten Strategie

Umgang mit Legacy Code



Hauptziel der Masterarbeit

Durchführung einer angemessenen Analyse, um die Strategien für Softwareentwicklung ableiten zu können, wie man mit dem vorliegenden System umgehen kann.

Ansätze

(I) Erkennung von Defektmustern im Code anhand Fehlerberichte

(II) Ermittlung von defektanfälligen Komponenten anhand Defekt-Korrekturen

(III) Analyse von Wartungsaufwand von unerreichbaren Codeblöcken

(I) Ansatz

Erkennung von Defektmustern

(I) Motivation & Ziel

- Zusammenhänge zwischen dem Entwicklungsprozess
- Ursprung von Fehlern

(I) Forschungsfrage

Inwiefern können bestimmte Fehlermuster im System erkannt werden?

Konzept

1) Auswahl von Fehlerberichten aus Issue Tracker;
2) Identifizierung von fehlerhaften Codezeilen aus SVN-History;
3) Manuelles Erkennen von Ähnlichkeiten;
4) Ermittlung einer Klassifikation von Defekten.

Mustern -> Probleme -> Konventionen

Klassifikation von Defekten

Fehlerklassifizierungsansätze:

- Orthogonal Defect Classification
- IEEE Standard Classification for Software Anomalies
- ...

automatisch:

- DynaMine

(I) Teilfragen

- Gibt es ähnliche Defekte?
- Welche Arten von Defekten kommen meistens vor? (*Klassifikation*)
- Was sind die Ursachen (*Probleme*) von Fehlermustern?

(II) Ansatz:

Defektanfällige Komponente

Motivation & Ziel

- Identifizierung kritischer Module
- Verständnis von deren Fehlerquellen
- Vermeidung weiterer Defekte

(II) Forschungsfrage

Welche Module haben ein großes Fehlerpotenzial und durch welche Variablen wird dieses beeinflusst?

(II) Konzept

Ermittlung von defektanfälligen Komponenten durch Berechnung der durchgeführten Korrekturen:

- Identifizierung von *Bugfix Links* (Commit-Message + Kommentar aus dem Fehlerbericht)
- Identifizierung der Module mit den meisten Korrekturen

- **Identifizierung von *Defect Insertions*** <- *SZZ Algorithmus (z.B. CVSAnalY)*

Stellen, die häufig korrigiert sind -> Reengineering

(II) Teilfragen

A. Codebereich:
   - Gibt es Module, die besonders defektanfällig sind?
   - Gibt es Module, die besonders anfällig für "*Defect Insertion*" sind?
   - Welche Änderungen (*Fixes*) oder Eigenschfaten von Änderungen führen zu weiteren Problemen?

B. Zeitbereich:
   - Sind bestimmte Tageszeiten/ Wochentagen/Wochen/ Phasen während des Release-Zyklus besonders defektanfällig?
   - Was sind die Aufwanddaten für Korrekturen?

Ansatz III

Wartungsaufwand von unerreichbaren Codeblöcken

(III) Motivation & Ziel

- Analyse von unerreichbarem Code
- Wirkung auf die Wartung

(III) Forschungsfrage:

Wie ist der Wartungsaufwand von unerreichbaren Codeblöcken?

(III) Konzept:

- Ermitteln von unerreichbarem Code mit *SonarQube* Metriken
- Analyse von Methoden, die als “*Unused*” und “*Deprecated*” markiert sind

(III) Teilfragen:

- Wieviele unerreichbare Codeblöcke können aus dem System entfernt werden?
- Wie kann eine Bereinigung ohne großes Risiko durchgeführt werden?

Vielen Dank!