Freie Universität Berlin

Verbesserung des Entwicklungsprozesses durch testgetriebene Entwicklung und kontinuierliche Integration

Stefan Rossbach
Institut für Informatik
Freie Universität Berlin
07.07.2011

Überblick

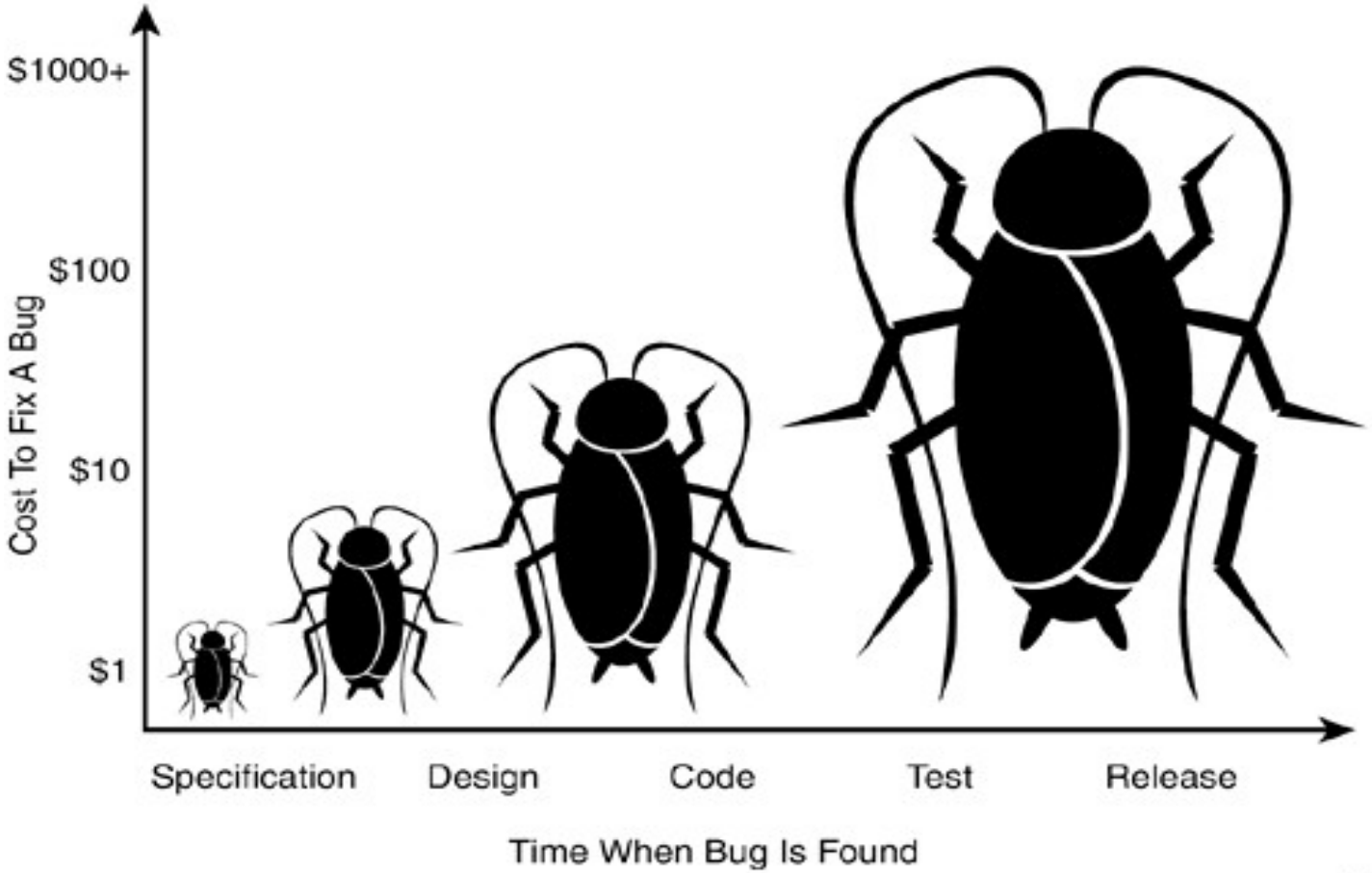
- Testen von Software
- Testgetriebene Entwicklung
- Kontinuierliche Integeration
- Derzeitige Probleme in Saros
- Zeitplan

Testen von Software

- Software wird immer getestet
  - z.B bei der Kompilierung auf Syntaxfehler
  - in C/C++ außerdem beim Linken auf fehlende Bibliotheken, in Java erst während der Laufzeit
  - immer jedoch spätestens durch den Endbenutzer

- Testen von Software
  - dient zur Auffindung und Beseitigung von Defekten
  - am Besten in möglichst frühen Stadien der Entwicklung
  - Tests können allerdings nie die Abwesenheit von Defekten zeigen

Kosten von Defekten in der Entwicklung



[Patton05]

Statisches Blackbox-Testen

- Testen der Spezifikation gegen die Anforderungen
  - wurden die nötigen Standards eingehalten
    - z.b GUI Layout diverser Betriebssysteme
    - Behörden typische Wege z.B im Militär
    - Sicherheitsaspekte
  - Skalierbarkeit (Architektur)
  - Erweiterbarkeit (Architektur)
  - Vollständigkeit der Spezifikation
  - beschreibt die Spezifikation die richtige Lösung
  - existieren keine Mehrdeutigkeiten
  - ist die Implementierung testbar ?

Dynamisches Blackbox-Testen

- Systemtest (Entwickler)
  - testen der Spezifikation des Systems (Verifikation) ggf. Validation
- Akzeptanztests (Kunde, Endbenutzer)
  - testen des Systems ggf. ohne die Spezifikation (Validation)
- Testqualität
  - testen des Systems unter korrekter Eingabe
  - testen des Systems unter falscher Eingabe
  - testen von Grenzwerten
  - testen von nicht spezifizierten Verhalten

Beispiele

- Taschenrechner
  - berechnen der Wurzel von einer negativen Zahl
  - berechnen des Logarithmus von 0

- Bildprogramm
  - laden von falschen Dateien z.B laden einer Audiodatei mit dem Namen MyImage.jpg
  - laden von Dateien mit ungewöhnlichen Dateipfaden z.B

    C:\Benutzer\roßbach\Meine Bilder\Alice + Bob streiten sich.bmp
  - laden von korrumpierten Dateien
  - Entfernung des Mediums während des Ladevorgangs
  - laden von Dateien mit überlangen Pfadnamen
    - die meisten in C/C++ geschriebenen Programme die fopen() benutzen sind unter Windows Systemen auf MAX_PATH (ca. 260 Zeichen) beschränkt

Statisches Whitebox-Testen

- Testen des Codes durch Inspektion unter anderem auf
  - Lesbarkeit und Wartbarkeit
  - Dokumentation
  - numerische Fehler
  - falsche Blockkontrolle
  - falsche boolesche Verknüpfungen
  - Portabilität
  - Mehrsprachigkeitsunterstützung
  - Einhalten der Code Konventionen und Richtlinien
    - Sichtbarkeit von Klassen, Methoden, und Variablen
    - Fehlerbehandlung
    - Loggen
    - Benennung von Klassen, Methoden, Variablen und Paketen
    - verbotene Kontrollstrukturen z.B goto

Dynamisches Whitebox-Testing

- testen von einzelnen oder mehreren Komponenten mit Blick auf den Quellcode
  - ermöglicht zielgenaues Testen von Ausnahmesituation die im realen Betrieb unter Umständen nie vor fallen werden
    - z.B sind die Fehlermeldungen korrekt
    - sind die Meldungen verständlich
  - unter Zuhilfenahme von Code Überdeckungswerkzeugen kann noch nicht getesteter Code eingegrenzt werden
  - ermöglicht das Testen in Isolation ggf. werden Hilfsmittel wie Attrappen (Mock Objekte) benötigt
  - sollten möglichst automatisiert stattfinden
  - aber immer zuerst Tests gegen die Spezifikation erstellen damit sichergestellt wird, dass auch die richtige Funktionalität getestet wird

- Verwendung von vorhandenen Frameworks wie z.B für Java: Junit, PowerMock, EasyMock und Cobertura oder Emma

Code Überdeckung

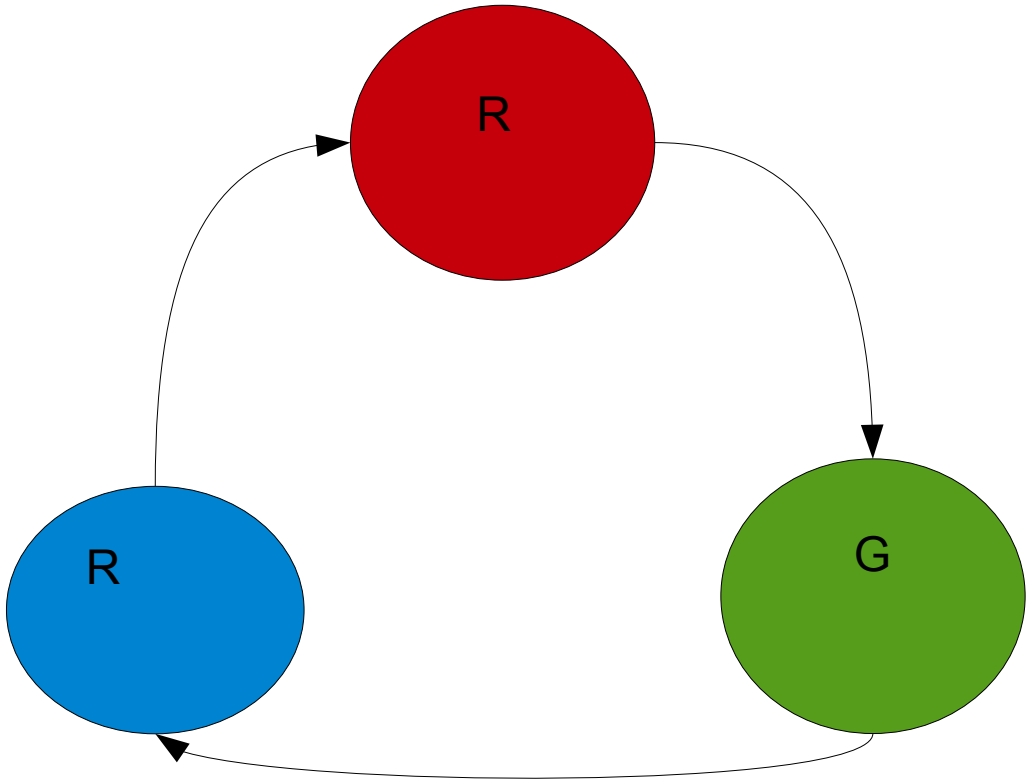
- ist ein Hilfsmittel zum Anzeigen von noch nicht getesteten Code
- Code Überdeckung kann nicht die Abwesenheit von Defekten zeigen
- Unterschiedliche Arten, meisten mit Cx bezeichnet, aber nicht standardisiert
  - C0 Anweisungsüberdeckung
    - z.B x = x +1;
  - C1 Zweigüberdeckung
    - if (x == 0) { x++;} enthält zwei Mögliche Pfade
  - C2 Pfadüberdeckung
    - die Kombination aus allen C1 Möglichkeiten inklusive Schleifenstrukturen
  - C3 Bedingungsüberdeckung
    - if (x && b && z || k) ...

Testgetriebene Entwicklung

- Testgetriebene Entwicklung ist eine Design Strategie
  - zuerst werden die Tests geschrieben
  - danach wird die eigentliche Spezifikation implementiert
- Vorteile
  - da Tests im voraus geschrieben werden müssen entfällt das mühselige Schreiben am Ende der Entwicklungsphase
  - der Code ist testbar
  - ermöglicht einfache Refaktorisierung von bereits existierenden Code
    - Programmierer hat die Sicherheit durch existierende Tests keine Fehler einzubauen
- Nachteile
  - verwirrend für „Neulinge“
  - erfordert diszipliniertes Verhalten der Entwickler

Testgetriebene Entwicklung

schreibe einen Test (der fehlschlägt)



refaktorisiere den Code

schreibe genau soviel Code bis der Test erfolgreich ist

Kontinuierliche Integration

- ist im eigentlichen Sinne keine agile Methode da die Idee schon vor der agilen Entwicklung propagiert wurde
- die Idee ist einfach, nach jeder Änderung an dem System wird es neu gebaut und mit Hilfe von Regressiontests getestet
- ermöglicht durch Software die auf einem Server läuft (CI Server)
- kontinuierliche Integration solle aber nicht nur kontinuierliche Kompilierung enthalten

- CI Server bieten weitaus mehr
  - Konditionale Ausführungen von sogenannten „Builds“
  - erzeugen von Historien
  - Verteilung von „Builds“ auf verschiedene „Slave“ Rechner

- Nachteile
  - erfordert unter Umständen viele Ressourcen wie z.B Speicherplatzbedarf, Rechenleistung, Arbeitsspeicher, Netzwerkkapazität etc.

Derzeitige Probleme in Saros

- Spezifikationen
- Dokumentation
- veraltete Testbeschreibungen in der TestLink Datenbank
- zu wenig Whitebox Testfälle, Code Überdeckung momentan ~ 20%
- Random random = new StfTestCase(){public boolean nextBoolean(){return this.run() == passed;}};
- Architektur ? Momentan viele Einzelstücke verwaltet durch einen Pico Container
- manche Komponenten sind in Isolation gar nicht bzw. nur mit sehr großen Aufwand testbar (schlechtes Design ?)
- zu viel Nebenläufigkeit
- manuelles Testen findet scheinbar nur im Institut statt, verfälscht daher Ergebnisse in Bezug auf die Netzwerkschicht

Zeitplan

- Woche 1 – 4 (06.06.2011 – 03.07.2011)
  Einarbeitung in das STF
  Konfiguration der VM
  Installation des Jenkins CI Server
  Schreiben von Scripts zum Starten des STF
  Konfiguration von Jobs
- Woche 4 – 8 (04.07.2011 – 31.07.2011)
  Korrektur von Defekten und Refaktorisierung des STF
  Erstellen eines Testlink Jobs für den CI Server
  Fertigstellung des automatischen Deployments
  Schreiben von Testfällen
- Woche 8 – 12 (01.08.2011 - 28.08.2011)
  Schreiben von Testfällen
  Schreiben der Arbeit

Literatur

- [Patton05] Ron Patton: Software Testing 2nd Edition 2005
- [Beck02] Kent Beck: Test-Driven Development By Example 2002
- [DuMaG07] Pual M. Duvall, Steve Matyas, Andrew Glover: Continuous Integration. Improving Software Quality and Reducing Risk 2007
- [CC] http://de.wikipedia.org/wiki/Ueberdeckungstest

DEMO

FRAGEN ?

VIELEN DANK !