Freie Universität Berlin

Seminar zu Ursachen und Vermeidung von Fehlern in der Softwareentwicklung (vol. 2)

Analyse vergangener Defektbehebungen

Philipp Schmidt - phils@inf.fu-berlin.de
Institut für Informatik
FU Berlin
27.02.2006

Überblick

- Warum statische Analyse allein nicht genügt
- Warum vergangene Defektbehebungen analysieren?
  - Charakter von Defektbehebungen
  - Charakter von Änderungen
- Nutzung von Versionsinformationen
  - Statistische Nutzung
  - Verbesserung statischer Analyse
  - Usage Pattern
- Fazit

Warum statische Analyse allein nicht genügt.

Freie Universität Berlin

- Schlechte Anpassbarkeit auf konkrete Projekte
- Hohe False Positive Rate
- Keine bzw. sehr aufwendige Erkennung struktureller Defekte (z.B. double frees)

Charakter von Defektbehebungen

Freie Universität Berlin

- „Ein Fehler kommt selten allein“

- Ein Blick auf behobene Defekte lohnt sich:
  - Die meisten Defekte entstammen ähnlichen Quellen und Gründen
  - Wird ein Defekt behoben, ist höchstwahrscheinlich irgendwo ein ähnlicher zu finden

Warum vergangene Defektbehebungen Analysieren?

Freie Universität Berlin

- Ergebnisse lassen sich für statische und dynamische Verfahren nutzen.
- Ergebnisse von statischer Codeanalyse lassen sich mit Statistischen Daten aus einem Versionskontrollsystem Bewerten
- Neue Verfahren lassen sich mit den erhobenen Daten Entwickeln

Charakter von Änderungen

- Änderungen sind in größeren Projekten durch Versionskontrollsysteme wie CVS gut verfügbar
- Die meisten Defekte sind nur Kleinigkeiten (z.B. falsche Casts, etc)
  - Ihre Behebung zieht meist sehr kurze Checkins nach sich
- Zusammenhängende Konstrukte wie Sperren und deren Aufhebung treten in Commits fast immer zusammen auf
  - Wenn nicht, so hat dies oft einen Defekt zur Folge

Charakter von Änderungen

- Daten über Änderungen am Code sind am besten direkt während der Entwicklung zu erheben
  - Hier werden die meisten Defekte behoben

- Daten aus Releases zu erheben macht wenig sinn
  - Die Defekte gehen in der Masse der Änderungen unter

Noch Fragen?

Statistische Auswertung von Fehlerbehebungen

Freie Universität Berlin

- Auswertung von Fehlerdatenbanken und Code
  - Ziel: Besonders Fehlerträchtige Konstrukte in Entwurf und Code finden ohne diese zu erheben

- Vorgestelltes Verfahren aus:

  Claes Wohlin, Martin Höst, Magnus C. Ohlsson. **Understanding the Sources of Software Defects: A Filtering Approach.** 8th International Workshop on Program Comprehension (IWPC'00), June 10-11, Limerick, Ireland 2000, p. 9

Statistische Auswertung von Fehlerbehebungen

Freie Universität Berlin

- Daten stammen nur aus Code und Fehlerdatenbank, nicht aus einem Versionskontrollsystem
- Alle Daten werden pro Modul erhoben
- Es werden im Design erhoben:
  - Größenmaß: Seitenanzahl
  - Anzahl von Konstrukten (Signal In / Signal Out, etc...)
- Es werden im Code erhoben:
  - Größenmaß: Lines of Code
  - Anzahl an Sprachkonstrukten (if, while, goto, etc...)

Statistische Auswertung von Fehlerbehebungen

Freie Universität Berlin

- Filterung der Ergebnisse – Es werden pro Modul betrachtet:
  - Das Konstrukt dessen Häufigkeit am stärksten mit der Anzahl der Defekte korreliert.
  - Das Konstrukt dessen Häufigkeit am stärksten mit der Anzahl der Defekte korreliert, bereinigt um die Größe der Komponente.
  - Die Konstrukte deren Häufigkeit eine stärkere Korrelation mit Auftreten von Defekten als deren Größe haben
  - Die Konstrukte deren Häufigkeit eine stärkere Korrelation mit Auftreten von Defekten haben als das Auftreten von Defekten mit der Größe.

Statistische Auswertung von Fehlerbehebungen

Freie Universität Berlin

- Interpretation der Ergebnisse
  - Bleibt dem Anwender der Methode überlassen
  - Geeignet z.B. zur Entwicklung von statischen Methoden

- Fazit
  - Recht früh entwickelt
  - Ohne vollen Zugriff auf den Code anwendbar
    - Statistische Code-Daten reichen
  - Keine direkten Erfolge auf die Codequalität

Noch Fragen?

Bewertung statischer Analyse

- Die Ergebnisse von statischen Analyseverfahren werden durch Daten aus der Versionsgeschichte verifiziert und Bewertet
  - Dadurch kann die False Positive Rate gesenkt werden
  - Verfahren mit vielen False Positives werden Praxistauglich

- Vorgestelltes Verfahren aus:

  Chadd C. Williams, Jeffrey K. Hollingsworth. **Bug Driven Bug Finders.** International Workshop on Mining Software Repositories (MSR), May 2004

Bewertung statischer Analyse
Beispiel: Returnvalue-Checker

Freie Universität Berlin

- Ein Returnvalue-Checker prüft ob ein Rückgabewert einer Funktion geprüft oder ungeprüft verwendet wird.
  - Dies ist bei Benutzereingaben sehr wichtig
  - Bei Hilfsroutinen kann dies aber unnötige Rechenzeitverschwendung sein

- Daten aus dem Versionskontrollsystem ermöglichen es automatisch abzuwägen
  - Werden die Rückgabewerte of geprüft, ist es wahrscheinlich ein Defekt sie nicht zu prüfen.
  - Wurde dies in der Vergangenheit an anderer Stelle korrigiert ist ein Defekt sehr wahrscheinlich.
  - Werden die Rückgabewerte ansonsten nie oder nur selten geprüft, ist das wahrscheinlich kein Defekt.

Bewertung statischer Analyse
Beispiel: Returnvalue-Checker

Freie Universität Berlin

- Beispiel: Returnvalue-Checker wurde auf das CVS-Repository von Apache httpd angewandt:
  - Ausgangs False Positive Rate: 99%
  - False Positive Rate mit beiden Verfahren: 78%
  - False Positive Rate, wenn man nur hinzugefügte Returnvalue-Checks beachtet: 74%

- Fazit:
  - False Positive Rate lässt sich drastisch senken
  - Es bleibt fraglich on das Verfahren auf andere statische Tests übertragbar ist.

Noch Fragen?

Usage Pattern zur Defekterkennung

Freie Universität Berlin

- Idee: Häufige gemeinsame Verwendung von Code Mustern wird anhand der Änderungen im Versionskontrollsystem erkannt
  - Verletzungen sind wahrscheinlich Defekte.

- Vorgestelltes Verfahren aus:

  V. Benjamin Livshits and Thomas Zimmermann. **DynaMine: Finding Common Error Patterns by Mining Software Revision Histories.** In Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 2005), September 2005

Usage Pattern zur Defekterkennung: Was sind Usage Pattern?

Freie Universität Berlin

- Usage Pattern sind mehrere Konstrukte, und ihre Beziehung zueinander (z.B. <{(a,b)}, {a=>b}>)
  - Dies ist absichtlich recht offen gehalten

- In DynaMine werden nur Paare von Funktionsaufrufen betrachtet wie z.B. `i.hasNext()` und `i.next()`.
  - Paare haben unter dem Aspekt das sie später durch Data Mining gewonnen werden den Vorteil das sie mit akzeptablem Aufwand zu erkennen sind.

Usage Pattern zur Defekterkennung: Mining Usage Pattern

Freie Universität Berlin

- DynaMine verwendet den Apriori-Algorithmus
  - Dieser Algorithmus wird auch in Webshops für das „Kundinnen die diesen Klappspaten gekauft haben, haben auch die Gumolka Kettensäge gekauft“ verwendet.

- Es werden ausschließlich CVS Checkins betrachtet
  - Diese kommen den Transaktionen für den Apriori-Algorithmus am nächsten.
  - Die CVS Checkins enthalten alle nötigen Informationen und sind kurz genug um sie sinnvoll auszuwerten.

Usage Pattern zur Defekterkennung: Wie benutzt man Usage Pattern?

Freie Universität Berlin

- Je nach Wahrscheinlichkeit eines Pattern wird eine im Code gefundene Verletzung als Kandidat für eine Defekt präsentiert.

- Kandidaten werden zur Laufzeit verifiziert.
  - Die Codefragmente mit den Defektkandidaten werden um die Aufzeichnung der Fehlerkandidaten erweitert.
  - Es wird Überprüft ob Patternteile
    - in tief im Aufrufstack doch eingehalten werden
    - in Dead Ends liegen

Usage Pattern zur Defekterkennung: Fazit

Freie Universität Berlin

- Das Verfahren ist sehr komplex
- Das Verfahren wirkt fast produktionsreif und ist als Eclipse-Plugin verfügbar.
- Das Verfahren ist nur für große Projekte geeignet
  - nur solche bieten eine genügend große Datenbasis für das Mining der Usage Pattern
  - als Beispiele wurden JEdit und Eclipse herangezogen
  - bisher keine Untersuchungen ab welcher Projektgröße das Verfahren funktioniert

Noch Fragen?

Fazit

- Auf der Versionsgeschichte von Projekten basierende Verfahren zur Defekterkennung haben Zukunft!
  - Der Fortschritt in diesem Berecht ist anhand der Artikel deutlich zu erkennen.
- Je umfangreicher Projekte sind, desto besser sind die Verfahren geeignet
  - Je Umfangreicher die gesammelten Informationen, desto geringer die False Positive Rate
- Für kleine Projekte ungeeignet

Anregungen / Kritik

Vielen Dank!