What is software testing? And why is it so hard?

Seminar: Softwareentwicklung in der Wissenschaft

Marvin Heuer

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2018-07-16

Gliederung

Einleitung

- Was ist Testen?
- Wieso bleiben Bugs unentdeckt?
- Testen ist und bleibt schwer nachzuvollziehen
- Wie können wir vorgehen für gutes Testen?

Definitionen im Software Testing

- Softwaretests[1]
  - Ausführung eines Systems/Komponenten unter bestimmten Bedingungen
  - Überwachung des Ablaufs
  - Auswertung hinsichtlich einiger Aspekte
- Softwaretests im Paper[2]
  - Prozess des Ausführens eines Softwaresystems
  - Abgleich der Spezifizierung
  - Abgrenzung zu Code Reviews
- Komponententests[2]
  - Individuelle Komponenten oder Sammlung von Komponenten
  - Testinput nur für Komponenten
  - Häufig mit Debugger

Definitionen im Software Testing

- Integrationstests[2]
  - Verschiedene Komponenten durch eine Reihe von Einzeltests
  - Fokus auf Kommunikation zwischen den Komponenten
- Systemtests[2]
  - Sammlung von Komponenten, die ein Produkt darstellen
  - Eingabedomäne sind alle Komponenten

Testauswahl

- Funktionale Tests[2]
  - black-box test
  - Testauswahl und Testdatenadäquanz basierend auf der Spezifikation
  - „spezifikationsbasierte Tests“
- Strukturelle Tests[2]
  - white-box test oder code-based test
  - Eingaben basierend auf der Struktur des Quellcodes/Datenstrukturen

Vier Phasen des Testens

- Modellieren der Softwareumgebung[2]
- Auswahl von Testszenarien[2]
- Ausführen und Auswerten der Testszenarien[2]
- Testfortschritt messen[2]

Phase 1: Modellieren der Softwareumgebung

- Interfaces erkennen und simulieren[2]
  - Human interfaces
  - Software interfaces
  - File system interfaces
  - Communication interfaces
- Interaktionen außerhalb der Software[2]
  - „Non-repeatable read“
  - Absturz eines Partners
  - Paralleler Zugriff

Einführung zum Beispiel

- Programm, dass die Systemzeit und Datum anzeigt
- Alt+F4 beendet das Programm
- Tab-Taste wechselt zwischen den Feldern

Abbildung: Aktuelle Systemzeit und Datum mit Feldern für neue Werte

Beispielumgebung

- Zwei Eingabequellen
- Verschiedene Eingaben:
  - Gültige Werte
  - Ungültige Werte
  - Unerwartete Werte
- Rahmenbedingungen prüfen (Andere Programme, RAM etc.)

Phase 2: Auswahl von Testszenarien

- Hohe Abdeckung erreichen[2]
  - Jede Zeile mind. einmal ausführen (im Optimalfall)
  - Jedes extern generiertes Ereignis nutzen
  - Fokus auf „input sequences“

Kriterien bei der Auswahl

- Ausführungspfadtest (Execution path test criteria)[2]
  - Jede Anweisung muss einmal ausgeführt werden
  - Jede Verzweigung muss ausgewertet werden
  - Jede Datenstruktur muss genutzt werden
  - „Fault Seeding“
- Testdomaingrößen[2]
  - Jeder physischer Input
  - Jede Interfacekontrollstruktur
  - „discrimination criterion“
  - Typisches ausführbares Verhalten nachahmen
  - Grenzwerte und Äquivalenzklassen[15]

Quellcode im Beispiel

- Pfade aufdecken
- Alle Pfade müssen geprüft werden
- Beispiel hätte 28 Testfälle

```
Input = GetInput()
While (Input != Alt-F4) do
    Case (Input = Time)
        If ValidHour(Time.Hour) and ValidMin(Time.Minute) and
            ValidSec(Time.Second) and ValidAP(Time. AmPm)
        Then
            UpdateSystemTime(Time)
        Else
            DisplayError("Invalid Time.")
        EndIf
    Case (Input = Date)
        If ValidDay(Date.Day) and ValidMonth(Date.Month) and
            ValidYear(Date.Year)
        Then
            UpdateSystemDate(Date)
        Else
            DisplayError("Invalid Date.")
        EndIf
    Case (Input = Tab)
        If TabLocation = 1
        Then
            MoveCursor(2)
            TabLocation = 2
        Else
            MoveCursor(1)
            TabLocation = 1
        EndIf
    EndCase
    Input = GetInput()
Enddo
```

Abbildung: Beispielquellcode

Schnittstellen im Beispiel

- Zeit:
  - 86.400 verschiedene zulässige Eingaben
  - Unzulässige müssen ebenfalls geprüft werden
- Datum:
  - Zulässige und unzulässige Eingaben finden
  - Kombinationen wie Mitternacht 1999 um beides zu testen
- Mehrere Eingaben hintereinander?
- Unendlich viele Testfälle möglich

Mögliche Fälle	While	Case 1	If 1	Case 2	If 2	Case 3	If 3
1	F	-	-	-	-	-	-
2	T	T	T	-	-	-	-
3	T	T	F	-	-	-	-
4	T	F	-	T	T	-	-
5	T	F	-	T	F	-	-
6	T	F	-	F	-	T	T
7	T	F	-	F	-	T	F
8	T	F	-	F	-	F	-

Abbildung: Wahrheitstabelle

Phase 3: Ausführen und Auswerten der Testszenarien

- Vollautomatisierung
- Szenario-Auswertung
- Ansätze zum Auswerten
  - Formalismus
  - eingebetteter Testcode

Regressionstesten

- Wieviel „Retesting“ ist nötig?
- Ein Fix kann:
  - nur das gemeldete Problem lösen
  - das gemeldete Problem nicht lösen
  - das gemeldete Problem lösen, dabei etwas anderes "kaputt"machen
  - das gemeldete Problem nicht lösen, dabei noch etwas anderes "kaputt"machen

Regressionstesten

- Softwaretesttyp
- Bestimmung ob neue Probleme als Ergebnis von Änderungen auftauchen
- Vorgehen ist klar

Phase 4: Testfortschritt messen

- Fortschritt in Zahlen schwer zu messen
- Strukturelle Vollständigkeit:
  - Klassische Programmierfehler?[10]
  - Kompletter Quellcode ausgeführt?[3]
  - Interne Daten initialisiert und benutzt?[5]
  - „seeded errors“[3]
- Funktionale Vollständigkeit:
  - Alle Ausführungspfade durch Tests abgedeckt?[11]
  - Alle Eingaben getestet?[3]
  - Kompletten Zustandsraum entdeckt?[6]
  - Alle erwarteten Szenarien ausgeführt?[12]

„Testability“ and Zuverlässigkeitsmodelle

- „Testability[14]“ als eine Möglichkeit die Komplexität des Testens eines Programmes zu bestimmen
- Zuverlässigkeitsmodelle:
  - mathematische Modelle von Testszenarios
  - sagen Verhalten von Software voraus
  - setzen „operational profile“ voraus

FortranTestGenerator

- Generiert automatisch Komponententests
- Python
- Tests für ICON (letzte Woche)
- Entwickelt von Christian Hovy, Arbeitsgruppe Wissenschaftliches Rechnen

Zusammenfassung

- Testen stellt eine große Herausforderung dar
- Strukturelles Denken und Verständnis vom Code ist gefragt
- Komplexe Natur des Testens
- Wissenschaftliche Genauigkeit in der Praxis gefordert(siehe Testgenerator)

Literatur

- [1]IEEE 730-2014 IEEE Standard for Software Quality Assurance Processes, 3.2
- [2]J.A. Whittaker, "What Is Software Testing? And Why Is It So Hard?", IEEE Software 0740-7459, Vol. 17, No. 1, Jan/Feb 2000.
- [3]G.J. Myers, The Art of Software Testing, John Wiley Sons, New York, 1976.

Literatur

- [4]T.J. Ostrand and M.J. Balcer, “The Category-Partition Technique for Specifying and Generating Functional Tests,” Comm. ACM, Vol. 31, No. 6, June 1988, pp. 676–686.
- [5]S. Rapps and E.J. Weyuker, “Selecting Software Test Data Using Dataflow Information,” IEEE Trans. Software Eng., Vol. 11, No. 4, Apr. 1985, pp. 367–375.
- [6]J.A. Whittaker and M.G. Thomason, “A Markov Chain Model for Statistical Software Testing,” IEEE Trans. Software Eng., Vol. 20, No. 10, Oct. 1994, pp. 812–824.

Literatur

- [7]D.K. Peters and D.L. Parnas, “Using Test Oracles Generated from Program Documentation,” IEEE Trans. Software Eng., Vol. 24, No. 3, Mar. 1998, pp. 161–173.
- [8]D. Knuth, “Literate Programming,” The Computer J., Vol. 27, No. 2, May 1984, pp. 97–111.
- [9]G. Rothermel and M.J. Harrold, “A Safe, Efficient Algorithm for Regression Test Selection,” Proc. IEEE Software Maintenance Conf., IEEE Computer Soc. Press, Los Alamitos, Calif., 1993, pp. 358–367.

Literatur

- [10]B. Beizer, Software Testing Techniques, Van Nostrand Reinhold, New York, 1990.
- [11]J.B. Goodenough and S.L. Gerhart, “Toward a Theory of Test Data Selection,” IEEE Trans. Software Eng., Vol. 2, No. 2, June 1975, pp. 156–173.
- [12]J.D. Musa, “Software Reliability Engineered Testing,” Computer, Vol. 29, No. 11, Nov. 1996, pp. 61–68.

Literatur

- [13]E.J. Weyuker and T.J. Ostrand, “Theories of Program Testing and the Application of Revealing Subdomains,” IEEE Trans. Software Eng., Vol. 6, No. 3, May 1980, pp. 236–246.
- [14]J.M. Voas, “PIE: A Dynamic Failure-Based Technique,” IEEE Trans. Software Eng., Vol. 18, No. 8, Aug. 1992, pp. 717–727.
- [15]Arbeitsbereich Softwaretechnik und -architektur, Skript zur Vorlesung SSoftwaretechnik", SoSe 2018.