Grundbegriffe

Ch. Steindl

Häufige Fehleinschätzungen

Programmieren erfordert Fachleute, aber Testen kann jeder.

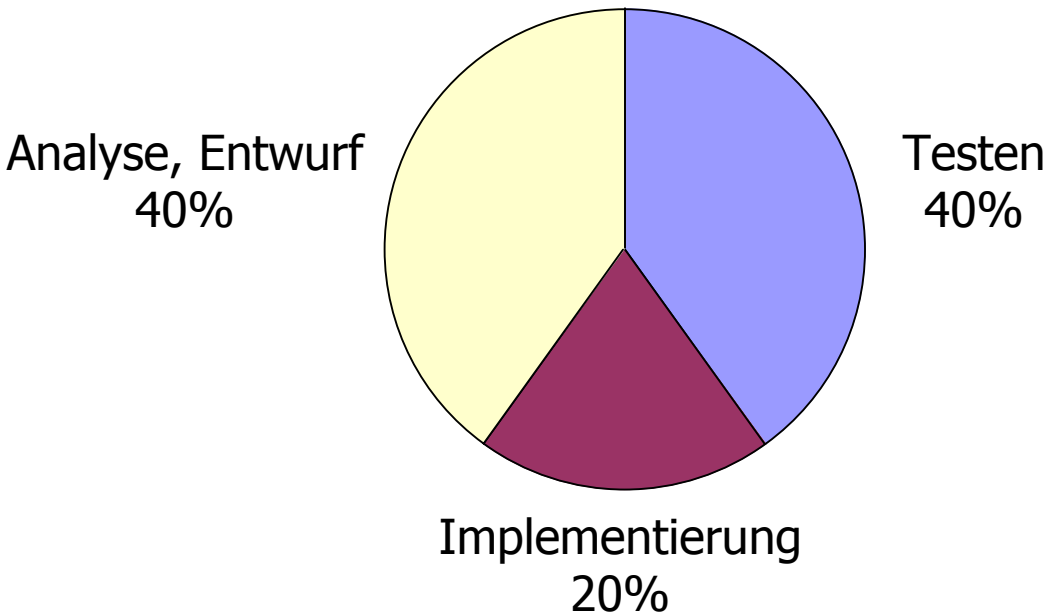
Es gibt ohnehin keine systematischen Testmethoden.

Testen erledigen wir zum Schluss.

Wenn wir sauber entwickeln, ist Testen fast überflüssig.

Anteil des Testens an der Projektdauer

40 - 20 - 40 Regel

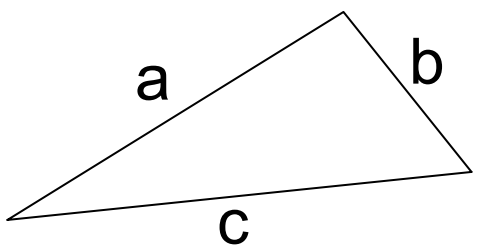


Ein Beispiel

Ein Programm liest 3 ganze Zahlen *a*, *b* und *c* und soll feststellen, ob sie die Seiten

- eines gleichseitigen Dreiecks
- eines gleichschenkeligen Dreiecks
- eines rechtwinkeligen Dreiecks
- eines sonstigen gültigen Dreiecks

bilden.



Mit welchen Eingaben würden Sie es testen?

Testfälle

Gültige Dreiecke

- gleichseitig: 3, 3, 3
- gleichschenkelig: 5, 5, 3
- rechtwinkelig: 3, 4, 5
- sonstiges: 3, 5, 7
- Permutationen davon
  (3,5,5), (5,3,5)
  (3,5,4), (4,5,3), (4,3,5), (5,3,4), (5,4,3)
  (3,7,5), (5,7,3), (5,3,7), (7,3,5), (7,5,3)

Ungültige Dreiecke

- a + b < c: 3, 3, 7
- a + b = c: 3, 4, 7
- negative Seitenlänge: 3, 4, -5
- 0, 0, 0
- nur 2 Seitenlängen: 3, 4
- Permutationen davon
  (3,7,3), (7,3,3)
  (3,7,4), (7,4,3), …
  (3,-5,4), (-5,3,4), …

Wie viele dieser Testfälle haben Sie?
Haben Sie überall das erwartete Ergebnis niedergeschrieben?

Definition des Begriffs Testen

Testen wird oft falsch definiert:

Testen ist der Prozess, der zeigen soll, dass ein Programm keine Fehler enthält.

Testen soll zeigen, dass ein Programm seine Spezifikation erfüllt.

Testen ist der Prozess, der das Vertrauen erzeugt, dass ein Programm das tut, was es soll.

Zitate zum Thema Testen

Mit Testen kann man nur zeigen, dass ein Programm Fehler enthält, niemals, dass es keine enthält.

Edsger W. Dijkstra

Der kluge Programmierer entwickelt ein Programm mit der Überzeugung, es von Anfang an korrekt entwickeln zu können …
und testet es dann im Wissen, dass es Fehler enthalten muss.

David Gries

Richtige Definition von Testen

Testen heißt:
ein Programm mit der Absicht auszuführen,
Fehler zu finden

- Ein Test ist erfolgreich, wenn er Fehler findet.
- Ein Test ist erfolglos, wenn alles gut geht.

Psychologie des Testens

- Testen ist ein destruktiver Prozess!
  - Jedes Programm hat Fehler – Ziel ist es, sie zu finden.
  - Entwickler ist zu wenig destruktiv,
    daher sollte der Tester nicht der Entwickler sein.
  - Richtige Einstellung: versuchen, das Programm zu Fall zu bringen.
- Wert des nächsten gefundenen Fehlers:
  - Mit jedem gefundenen Fehler wird das Programm wertvoller!
  - Korrektheitsbeweis ist unmöglich => Aufgabe, die den Tester frustriert.
  - „Noch einen Fehler zu finden" ist eine kleinere Aufgabe, der sich der Tester gewachsen fühlt.

Entwickler versus Tester

- Entwickler soll nicht testen!
  - Er bringt zu wenig destruktive Einstellung auf.
  - Missverständnisse beim Implementieren wiederholen sich beim Testen.
  - Der Entwickler gibt sich mehr Mühe, wenn er weiß, dass jemand anderer sein Programm testet.
- Tester soll Fehler nicht korrigieren!
  - Fehler nur aufdecken geht schneller => kann mehr Fehler finden.
  - Tester kennt sich im Programm nicht so aus => kann neue Fehler einführen.
  - Korrektur bedeutet Arbeit => Motivation sinkt, weitere Fehler zu finden.

Testen versus Debuggen

- Testen
  - Fehler finden
  - *Destruktive Tätigkeit* => Aufgabe des Testers
- Debuggen
  - Fehlerursache lokalisieren
  - Korrektur überlegen
  - Folgen der Änderung überlegen
  - Änderung durchführen
  - *Konstruktive Tätigkeit* => Aufgabe des Entwicklers

Verifikation versus Validierung

- Verifikation

  „Bauen wir das Produkt richtig?“

  d.h. Test gegen die Spezifikation

- Validierung

  „Bauen wir das richtige Produkt?“

  d.h. ist die Spezifikation überhaupt richtig, vollständig, …

Testprinzipien

1. Gehe davon aus, dass das Programm Fehler enthält.
2. Lass deine Programme von anderen testen.
3. Definiere für jeden Testfall die erwarteten Ergebnisse.
4. Vergleiche Testergebnisse gründlich mit erwarteten Ergebnissen („Das Auge sieht, was es sehen möchte“).
5. Verwende besondere Sorgfalt auf das Testen ungültiger Eingaben.
6. Bewahre alle Testfälle (Testprogramme) auf.
7. Je mehr Fehler man in einem Programmstück findet, desto mehr verstecken sich dort noch!

Testen im Software-Life-Cycle

1. Test der Spezifikation (Requirements Testing)
   - Klarheit, Vollständigkeit, Widerspruchsfreiheit, …
   - Spätere Tests müssen gegen dieses Dokument testen
2. Modultest (Unit Testing, Funktionstest)
   - Modul (Klasse) für sich testen
   - Testumgebung (Testtreiber) schaffen, die Modul aufruft
3. Integrationstest (Integration Testing)
   - schrittweises Zusammenfügen von Modulen zu Subsystemen
4. Systemtest (System Testing)
   - Test des Gesamtsystems unter realen Bedingungen (speziell der nichtfunktionalen Anforderungen)
5. Abnahmetest (Acceptance Test)
   - Test beim Kunden (über längeren Zeitraum)

Statisches vs. dynamisches Testen

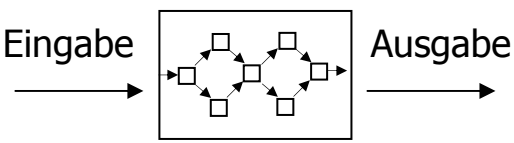
- Statisches Testen
  - Manuelle oder automatische Analyse des Programms, ohne es laufen zu lassen
  - Kann schon einsetzen, bevor das Programm fertig ist
  - Verifikation
  - Code Review / Walkthrough
  - Überprüfung von Codierstandards, Programmierstil und Komplexität
- Dynamisches Testen
  - Laufenlassen des Programms
  - Geht erst nach der Implementierung
  - Unit Test / Integrationstest / Systemtest / Abnahmetest
  - Profiling und Performance-Analyse

Black-Box- versus White-Box-Test

Black-Box-Test

- Source-Code des Moduls wird ignoriert;
  Es wird nur gegen die Spezifikation getestet.
- Testfälle werden so gewählt
  - 1 Testfall für jede gültige Eingabe (-klasse)
  - möglichst viele Testfälle für ungültige Eingaben
- Tester spielt mit dem Testobjekt und versucht,
  es zu Fall zu bringen.

White-Box-Test



- Source-Code des Moduls wird analysiert.
- Testfälle werden so gewählt, dass
  - jede Anweisung zumindest einmal ausgeführt wird.
  - jede Bedingung zumindest einmal true / false ergibt.
  - jeder Pfad zumindest einmal ausgeführt wird.

Testfälle und Testsuiten

Pro Testfall eine Testfunktion

```
boolean case17_1() {
    result = callMyFunction(100, "Mayer");  // Eingabe, Ausgabe
    return (result == expectedResult);      // erwartetes Ergebnis
}
```

Testsuite: Folge von Testfällen

```
void  suite17() {
    if ( ! case17_1()) println("17.1 failed");
    if ( ! case17_2()) println("17.2 failed");
    if ( ! case17_3()) println("17.3 failed");
    …
}
```

Testfälle und Testsuiten sollten schon vor der Implementierung des Testobjekts geschrieben werden.

Assertionen

Plausibilitätsprüfungen

Sind Parameterwerte korrekt?

Sind Datenstrukturen konsistent?

Gelten die gemachten Annahmen?

=> fördern Robustheit

Beispiel

```
void insert (int pos, String s) {
    if (checkAssert) {
       assert(textLen >= 0);
       assert(0 <= pos && pos <= textLen);
    }
    …
}
```

```
void assert (boolean cond) {
    if (! cond) System.exit(99);
}
```

Sollten als "Wachhunde" auch im fertigen Programm bleiben!

Hilfsdrucke

Alternative zum Debugger

Hilfreich zur Visualisierung
- großer Datenstrukturen
- zeitlicher Abfolgen (Methodenaufrufe, Änderung eines Variablenwerts, ...)

```
static boolean[] debug = new boolean[20];
…
void foo (int x, String y) {
        if (debug[3]) System.out.println("x=" + x + ", y=" + y);
        …
}
```

Am besten dynamisch ein/ausschaltbar

java MyProg -debug "3 5 17"

Sollten auch im fertigen Programm bleiben!

Literatur

[Bei90] Boris Beizer: *Software Testing Techniques* (2nd edition). Thomson Computer Press, 1990.

[Buz97] Tony Buzan, Barry Buzan: *Das Mind- Map- Buch. Die beste Methode zur Steigerung ihres geistigen Potentials*. Moderne Verlagsgesellschaft, 1997.

[Bei95] Boris Beizer: *Black-Box Testing*. John Wiley & Sons, 1995.

[Bou97] Kelley C. Bourne: *Testing Client/Server Systems*. McGraw-Hill, 1997.

[Gar99] Stewart Gardiner: *Testing Safety-Related Software, A Practical Handbook*. Springer, 1999.

[Kit95] Edward. Kit: *SoftwareTesting in the Real World*. Addison-Wesley, 1995.

[Ku+98] David C. Kung, Pei Hsia, Jerry Gao: *Testing Object-Oriented Software*. IEEE Computer Society, 1998.

[Mar95] Brian Marick: *The Craft of Software Testing*. Prentice Hall, 1995.

[McG96] John. D. McGregor: *Testing Object-Oriented Components*. Clemson University, Tutorial 9, ECOOP 96.

[Mye79] Glenford J. Myers: The Art of Software Testing, John Wiley, 1979.

[PaR89] Norman Parrington, Marc Roper: *Understanding Software Testing*. Ellis Horwood, 1989.

[PeR97] William E. Perry, Randall W. Rice: *Surviving the Challanges of Software Testing*. Dorset House Publishing, 1997.

Konferenzen

International Symposium on Software Testing and Analysis (ISSTA)

Theory, academic, leading edge practitioners. Sponsored by ACM and ACM's SIGSOFT.

International Conference on Software Engineering (ICSE)

Spring, world-wide. Technical. Primary source after ISSTA for leading edge results.

Quality Week (QW)

Annual, San Francisco Bay Area. Biggest Conference on Testing and QA.

Typically 700+. Many vendors. Good balance between technical/theoretical and practitioners.

Very broad base. Workshops. Sponsored by Software Research Institute

Quality Week Europe (QWE)

is held in Brussels, Belgium in November.

Software Testing, Analysis, and Review (STAR)

QAI International Software Testing Conference

More of a tutorial/workshop than a conference. Newbie orientation.

Zeitschriften

IEEE TSE (Transactions on Software Engineering)

Monthly. The most prestigious journal for testing stuff.

ACM TOSEM (Transactions on Software Engineering Methodology)

Quarterly. Relatively new journal (1992). Somewhat more theoretical than IEEE TSE.

ACM SIGSOFT Software Engineering Notes (from the Special Interest Group on Software Engineering)

Monthly. Goes back to May 1976.

ISSTA conference proceedings

Annual.

IEEE Software

Six times a year. Rarely the latest stuff.

More like surveys and overviews once a subfield has become established.

Refereed, generally high standards. Mostly overviews, but occasionally new stuff.

ACM Computing Surveys

Quarterly. Not specific to testing and QA, but contains the most prestigious survey articles in the field, typically only when a subfield is well established. The authors are usually authorities. Articles are long and comprehensive. When an ACM Survey on a topic appears, it usually means that the field has matured to the point where it is possible to write meaningful books.

Communications of the ACM

Monthly. Survey articles and overviews. Sometimes (rarely) original stuff. More academic and fundational/theory oriented than IEEE Software, but generally the same level.