5. Test Prinzipien

Testen ist eine wichtige Aktivität in unterschiedlichen Phasen der Software Entwicklung, wie wir bereits gesehen haben.

Verifikation und Validation haben mit Testen nur am Rande zu tun:

- Verifikation heisst : prüfen, ob zum Beispiel alle Aktivitäten einer Phase durchgeführt wurden
- Validation heisst : prüfen ob das System das tut, was es tun soll, gemäss Spezifikation

5.1. Qualitäts-Fragen

Qualität wird oft falsch verstanden, speziell im Zusammenhang mit der Software Entwicklung. Im Bereich der Konsumgüter versteht man unter Qualität oft mehr als "das Produkt ist gut"; ein Qualitätsprodukt ist in der Regel überdurchschnittlich gut. Im Bereich der Software ist man in der Regel damit zufrieden, wenn die Software in etwa das tut, was sie tun soll. Überragende Software gibt es demnach kaum?

5.1.1. Software Qualitäts-Sicherung

Die primäre Aufgabe einer Software Qualitätssicherung ist zu garantieren, dass die Software so funktioniert, wie man erwartet, dass sie funktionieren sollte.

Eigentliche Software-Qualitätssicherung geht jedoch weiter: ein Ziel ist es, die Qualität des Software Entwicklungs-Prozesses zu verbessern, wie wir im Zusammenhang mit CMM kennen gelernt haben.

Die SQA (Software Quality Assurance) legt oft auch Entwicklungsrichtlinien und Standards fest, zum Beispiel nach welchem Phasenmodell die Entwicklung zu geschehen hat.

5.1.2. Unabhängigkeit der SQA

Da die Aufgabe der SQA in der Regel als unangenehm angesehen wird, muss garantiert werden, dass sie auch ausgeführt wird (da sonst die Software vom Kunden eventuell nicht akzeptiert wird). Oft wird deswegen die SQA als Abteilung einer andern Organisation als der Software Entwicklung angegliedert.

Einige Software Standards legen diesbezüglich explizit fest, dass SQA unabhängig sein muss, weil sonst der Abteilungsleiter, der für die Entwicklung zuständig ist, die SQA eher als störend empfindet und deswegen auch nicht supportet.

Zum besseren Verständnis

Der Begriff "Qualität" bedeutet "gemäss der Spezifikation", ist also kein Luxusattribut!
Zum Beispiel:
der Qualitätsmanager bei Coca Cola steht dafür grade, dass jede Flasche, die ausgeliefert wird, den Coca Cola Standards gemäss produziert wurde. Qualität ist also kein Luxus und hat nichts mit Exzellenz zu tun.

5.2. Testen ohne Programmausführung

Die Person, welche die Spezifikation schreibt, ist genau sowenig geeignet, die Spezifikation zu prüfen, wie der Programmierer, der seine Programme auch nicht selber testen sollte.

Oft wird ein Review Team zusammen gestellt, welches zum Beispiel mit Hilfe eines sogenannten Walk Through durch die Spezifikation oder die Dokumentation "hindurch spaziert".

Walkthroughs und Code-Inspection sind Techniken, die Tests durch führen, OHNE dass das Programm ausgeführt wird.

Wenden wir uns zuerst den Walkthroughs zu

© J.M.Joller

5.2.1. Walkthroughs

Ein Walkthrough Team besteht normalerweise aus 4-6 Team-Mitgliedern. Das Team setzt sich typischerweise wie folgt zusammen:

- Ein Vertreter des Design Teams : dieser kennt die ursprünglich verlangten Anforderungen an das System
- Der Projektleiter der Spezifikationsphase
- Ein Kundenvertreter
- Ein Vertreter des Teams, welches für die nächste Phase verantwortlich ist.
- Ein Vertreter der SQA Gruppe

Alle Mitglieder eines Walkthrough Teams müssen erfahrene Mitarbeiter sein, sonst ist die Wahrscheinlichkeit, dass Fehler gefunden werden eher klein.

Jedes Mitglied muss bereits im voraus die Dokumente gründlich studiert haben und offene Fragen notiert haben. Zudem muss jedes Mitglied eine Liste der Punkte zusammen stellen, die aus seiner Sicht besprochen, reviewed werden müssen.

5.2.2. Ablauf des Walkthroughs

Der SQA Vertreter übernimmt den Vorsitz des Walkthroughs. Der Grund liegt darin, dass zum Beispiel die Person, die für die vorherige Phase zuständig war, natürlich möchte, dass seine Phase so schnell wie möglich abgenommen wird.

Alle entdeckten Fehler werden schriftlich kurz festgehalten. Es wird beschlossen wer, bis wann, welche Fehler zu beseitigen hat.

Sicher werden nicht alle Fehler entdeckt. Aber in der Regel werden einige gravierende Fehler aufgedeckt.

Der praktische Ablauf des Walkthroughs sieht wie folgt aus:

- Entweder präsentiert jedes Team Mitglied seine Liste der offenen Fragen und der Punkte, die diskutiert werden sollten
  oder
- Ein SQA Mitglied präsentiert eine Vorgehensweise, die andern Teammitglieder stellen ihre Frage an der entsprechenden Stelle des Reviews

Das Vorgehen wurde in einem Standard der IEEE fest gelegt: IEEE 1028, 1988 Standard for Software Reviews.

Der Leiter des Walkthroughs muss primär ein Facilitator sein, das heisst, er muss Fragen stellen und einen konstruktiven Dialog einleiten.

5.2.3. Inspektion

Inspektion als SQA Aktivität wurde 1976 eingeführt und besteht aus folgenden fünf Teilen / Phasen:

1) Overview : Ein Mitglied des Entwicklungs Teams präsentiert eine Übersicht über den Projektstand
2) Preparation : jedes Mitglied versucht individuell die Dokumentation zu verstehen, inkl. einer Liste der aktuell bekannten und beseitigten Fehler
3) Inspection : Schritt für Schritt besprechen der Dokumente, die abzunehmen sind
   Innerhalb eines Tages legt der Leiter des Inspektions Teams einen Zwischenbericht vor
4) Rework : alle Fehler auf der Fehlerliste müssen aufgearbeitet werden
5) Follow Up : verfolgen, ob auch alle gefundenen Probleme beseitigt werden

In der Regel besteht ein Inspektions-Team aus 4 Mitgliedern:

1) Einem Moderator
2) Einem Mitglied des Design Teams
3) Einem Mitglied des Implementations Teams
4) Einem Mitglied des Test Teams

© J.M.Joller

Wichtig bei der Inspektion ist das schriftlich festhalten aller Fehler und aller Änderungen. Diese Informationen weren unter anderem zur Erarbeitung von Metriken, zugeschnitten auf das Entwicklungs-Team, gebraucht.

Eine Studie zeigte folgende Ergebnisse:
1) 82% der Fehler wurden bereits in der Design oder Implementations- Phase gefunden
2) die Produktivität des Integrations- Teams nahm signifikant zu, weil weniger Tests nötig waren als vorher
3) insgesamt wurden 25% Entwicklungszeit eingespart, obschon extra Zeit für die Inspektion eingeplant werden musste

5.2.4. Vergleich von Inspektion und Walkthrough

Inspektions Teams arbeiten in der Regel mit Checklisten. Walkthrough Teams erstellen eigene Listen (offene Frage, Punkte, die besprochen werden sollten).

Walkthrough besteht aus zwei Schritten: der Vorbereitung und der Team-Analyse.

Die Inspektion besteht aus fünf Schritten, wie wir oben gesehen haben. Inspektion dauert in der Regel auch wesentlich länger.

Für die Entwicklung machen sich beide Methoden in der Regel bezahlt, weil Fehler in späteren Phasen dadurch vermieden werden können.

5.2.5. Metriken

Eine der gängigen Metriken für diese Art Tätigkeit ist die *Fehlerdichte*, die Anzahl Fehler pro 1000 LOC (KLOC).

Ein weiteres Mass ist die Anzahl Fehler, die pro Stunde gefunden werden, die *Fehlerentdeckungsrate*.

Beiden Metriken lassen sich nur sinnvoll einsetzen, wenn mehrere Projekte abgewickelt werden und somit Vergleichszahlen erarbeitet werden können.

5.3. Ausführung Basiertes Testen

Im Englischen hat sich eine etwas differenzierte Terminologie eingeführt:

Fault : das was man unter einem Bug versteht, also ein Fehler
Failure : ein Ausfall, ein Bug , der zum Ausfall führte
Error : ein Kodierfehler

Testen ist nur unvollständig möglich. Es ist auch immer nur möglich zu zeigen, dass ein Fehler sich zeigt. Es ist nicht möglich Fehlerfreiheit zu zeigen. Das vollständige Testen eines Programmes wäre viel zu umfangreich: bis alle möglichen Zustände und alle möglichen Programm-Pfade getestet wären, würde das Programm schon lange nicht mehr im Einsatz sein.

Man muss sich also genauer überlegen, was man konkret testen sollte.

© J.M.Joller

5.4. Was sollte getestet werden?

Einer der "Test-Gurus" [Goodenough 1979] schreibt:
Testen muss

- In der zukünftigen Einsatzumgebung getestet werden
- Mit definierten Eingaben zu definierten und vor den Tests festgelegten Ergebnissen führen

Der Tester muss also die Spezifikation des Systems kennen. Er muss wissen, was das System liefern muss.
Das System muss mit definierten Tests, die auch dokumentiert werden und häufig bei neuen Versionen wieder als Tests dienen (Regression Testing).

Die Tests müssen in einem praxisnahen Umfeld, mit praxisrelevanten Daten stattfinden.

Beispiel:
Eine Sensor-Software, die das Wachstum von Gletschern überwachen soll, nützt wenig, wenn sie nur auf PCs läuft, die mindestens 10 Grad Celsius plus benötigen.

Eine Echtzeit Software, die Daten innerhalb einer zehntel Sekunde erfassen und bearbeiten muss, kann nicht von Hand getestet werden. Dafür muss also entweder ein reales System zum Testen zur Verfügung stehen, oder es muss ein entsprechender Simulator gebaut werden.

Welche Aspekte kann man testen?

1. Nützlichkeit
2. Zuverlässigkeit
3. Robustheit

5.4.1. Nützlichkeit

Wieviel "bringt" die Software dem Benutzer? Welche seiner Probleme kann er damit lösen?
Ein korrektes Produkt sollte auch noch korrekt im Sinne der Spezifikation sein, nicht nur korrekt im Sinne der Software Korrektheit.

Ist das Produkt nützlich und preiswert? Lohnt sich dessen Einsatz?
Nützlichkeit wird in der Regel an die Wirtschaftlichkeit gekoppelt. Ein Produkt, welches zwar nützlich aber nicht bezahlbar ist, wird kaum eingesetzt!

5.4.2. Zuverlässigkeit

Wie zuverlässig ist das Software Produkt?

Zuverlässigkeit wird im Falle der Hardware in der Regel gemessen als Ausfälle pro Zeiteinheit, oder mittlere Zeit zwischen zwei Ausfällen, MEAN TIME BETWEEN FAILURES (MTBF).

Bei Software ist diese Zahl nur schwer bestimmbar. Bei der Hardware ist sie eine Zahl, die mit dem Alterungsprozess der Komponenten zusammen hängt (Standard- Beispiel: wie viele Glühlampen leuchten länger als x Tage?)

Ein anderer Zuverlässigkeit- Massstab ist die MEAN TIME TO REPAIR (MTTR), die Zeit, die nötig ist, um das System nach einem Absturz wieder produktiv in Betrieb nehmen zu können.
Diese Grösse ist nur über einen längeren Nutzungszeitraum zu ermitteln. Aber für viele Firmen ist es DIE entscheidende Zahl, da sie kritisch vom Funktionieren des Systems abhängen

© J.M.Joller

5.4.3. Robustheit

Viele Systeme wurden für den Einsatz unter Bedingungen entwickelt, die später nicht eingehalten werden.

Beispiel:
Eine Überwachung Software wurde gemäss Spezifikationen so entwickelt, dass sie pro Sekunde 10'000 Ereignisse erfassen und bearbeiten kann. Was passiert, wenn drei Jahre später die Datenquelle 25'000 Ereignisse pro Sekunde liefert?

Es wird sich kaum mehr jemand an die Spezifikation erinnern und die Software still schweigend einsetzen. Der Effekt dürfte im schlimmsten Falle sein, dass jeder glaubt, die Hardware sei nicht schnell genug!

Robuste Software, das heisst Software, die entweder bei der Dateneingabe bereits Prüfungen durch führt, ob die Eingabe Bedingungen erfüllt sind. Oder Software, die auch noch ausserhalb ihres eigentlichen Einsatzbereiches brauchbare Ergebnisse liefert.

Robustheit lässt sich zum Beispiel so testen, dass eher untypische Daten eingegeben werden und beobachtet wird, wie das System darauf reagiert.

Beispiel:
Für die Berechnung des Ausfallverhaltens von Kernkraftwerken werden Berechnungen grosser Matrizen benötigt. Diese müssen zum Beispiel invertiert werden. Es ist bekannt, dass bei der Inversion numerische Probleme auftreten können, speziell wenn die Elemente der Matrix zum Beispiel 0.0001, also fast Null, oder wenn viele Elemente 1 oder 0.99999 sind (wieder können UNDERFLOW Probleme auftreten, also Probleme, die mit den zu verwendenden Algorithmen zusammen hängen).

5.4.4. Performance / Leistungsfähigkeit

Performance eines Software Systems wird zum Teil im Vertrag mit dem Kunden garantiert : "90% der Arbeiten am Computer weisen eine Reaktionszeit des Rechners von unter 0.2 sec auf, sind also schneller als 0.2sec ".

Wesentlich sind Performance Tests speziell im Bereich der Echtzeit Software, wie die Beispiele und dem Heading "Robustheit" gezeigt haben.

Die Performance eines Software Systems zu testen ist nicht immer einfach. Speziell muss darauf geachtet werden, dass die Performance vom Workload auf dem Rechner abhängt: ist der Rechner nur für mich aktiv, oder muss die Leistung unter der Voraussetzung erbracht werden, dass ein typischer Applikations-Mix auf dem Rechner läuft?

Im letzteren Fall muss garantiert werden, dass diese Workload reporduzierbar ist.

5.4.5. Korrektheit

Um 1970 hat man angefangen zu untersuchen, wie und ob überhaupt Software als korrekt bewiesen werden kann. Die Theorie dazu gilt als recht schwierig. Ich enthalte mich der Stimme, da ich mich damit einige Zeit befasst habe!

Auf jeden Fall muss man folgendes beachten:
Selbst wenn ich beweise, dass ein Programm korrekt ist, beweise ich damit eigentlich noch recht wenig:
Mein Beweis kann falsch sein
Die Voraussetzungen des Beweises können falsch sein
Der Beweis ist korrekt, der Algorithmus ist aber falsch
...

Ein Beispiel für die obigen Bemerkungen:
1969 hat Naur die Korrektheit eines 30 zeiligen Programmes bewiesen. Im Laufe der nächsten drei Jahre wurde gezeigt, dass das Programm 8 Fehler aufweist, die mit der Ausführung zusammen hängen. Zum Beispiel mit der Ausgabe, die in Sonderfällen nicht korrekt waren.

© J.M.Joller

Schauen wir uns ein sehr einfaches Beispiel an:

Eingabe Spezifikation	p: array mit n Integers, n>0;
Ausgabe Spezifikation	q : array mit n Integers, wobei gilt: q[0] <<= q[1] <= ... <= q[n-1]

Frage : ist diese Spezifikation korrekt?
Auf den ersten Blick würde man sagen : sicher!

Also schreiben wir ein einfaches Programm trickSort(int p[ ], int q[ ]), welches genau das leistet:

```
void trickSort(int p[ ], int q[ ])
{
        int i;
        for (i=0; i<n; i++)
                q[i] = 0;
}
```

Das Programm leistet offensichtlich genau das, was spezifiziert wurde. Das Ergebnis dürfte aber nicht das sein, was wir erwartet haben.
Die Spezifikation ist offensichtlich nicht korrekt! Wir haben vergessen anzugeben, dass die Elemente im Array q die selben Elemente sind, wie jene im Array p.

Eine verfeinerte Spezifikation könnte wie folgt aussehen:

Eingabe Spezifikation	p : array von n Integers, n>0;
Ausgabe Spezifikation	q : array von n Integers, so dass q[0]<=q[1]<=q[2]<=...<=q[n-1] Die Elemente des Arrays q sind eine Permutation der Elemente des Arrays p. p[ ] bleibt unverändert.

5.5. Testen versus Programm Korrektheit

Programm Korrektheit Beweise sind in der Tat in der Regel recht abstrakt und mathematisch anspruchsvoll.
Aber unabhängig davon zeigen die Beispiele, die wir eben erwähnt haben, dass sich der Aufwand kaum lohnt!

Wann sollte Software Korrektheit bewiesen werden?
Immer wenn Menschenleben davon abhängen!

In der Weltraumfahrt hat man sich entschieden einen anderen Weg zu gehen:
Eine Berechnung wird auf unterschiedlichen Rechnern in der Regel mit Hilfe unterschiedlicher Algorithmen oder / und Programm berechnet.
Anschliessend wird das Ergebnis verglichen und eine Mehrheitsentscheid gefällt: das Ergebnis, welches am meisten erscheint, wird als das korrekte angenommen. Die Korrektheit wird also "statistisch" bewiesen.

5.6. Wer sollte Programme dynamisch testen?

Das Vorgehen ist heute ziemlich standardisiert:
- Der Entwickler testet seine Module
- Eine unabhängige Stelle (SQA) testet die Integration

Alle Tests müssen festgehalten werden, damit sie auch reproduzierbar sind.

Bei neuen Versionen, neuen Releases, müssen wir zum Teil die alten Test nochmals durch führen, weil die neue Version bestimmte Probleme der alten behebt und hoffentlich keine neuen Probleme einführt.

© J.M.Joller

Aus Benutzersicht gibt es nichts Frustrierenderes als: "das hat aber vorher bereits funktioniert; jetzt habt ihr die Software geflickt und das läuft nicht mehr?".

5.7. Wann kann man mit Testen aufhören

Ganz einfach: DANN WENN DIE SOFTWARE NICHT MEHR EINGESETZT WIRD!

© J.M.Joller

5.8. Zusammenfassung

Wir wissen, dass Testen nicht eine separate Phase ist: Testen muss man parallel zur Entwicklung.

"Papier Tests" : Walkthroughs und Inspektion liefern häufig handfeste Hinweise auf Probleme und potentielle Fehler

"dynamische Tests" : zeigen das Verhalten des Systems in seinem natürlichen Einsatz Umfeld.
Dabei müssen folgende Aspekte getestet werden:

- Robustheit
- Performance
- Zuverlässigkeit
- Braucbarkeit
- Korrektheit

Dabei spielt Korrektheit eine spezielle Rolle:

- Ein korrektes Programm tut noch lange nicht das, was ich eigentlich von ihm erwarte
- Korrektheitsbeweise sind für die Praxis in der Regel nicht durchführbar.
- Korrektheitsbeweise verlangen vom, der den Beweis aufzeigen soll, gute bis sehr gute theoretische Kenntnisse

© J.M.Joller

5.9. Selbstestaufgaben

1. Sie haben 11 Tage Tests mit einem Software Paket durch geführt und haben zwei Fehler gefunden. Was vermuten Sie : gibt es noch viele Fehler in der Software?
2. Wie unterscheiden sich Walkthrough und Inspektion? Was haben Sie gemeinsam?
3. Sie arbeiten bei der Software Firma L&X ("lasy and extreme") in der Software Entwicklung. Die Firma hat 12 Entwickler und 2 Personen in der SQA Gruppe. Sie schlagen Ihrem Vorgesetzen, dem Leiter der Entwicklung vor, dass Walkthroughs eingeführt werden.
   Ihr Vorgesetzter antwortet Ihnen, dass er nicht einsehe, was vier Leute an einem Tisch besser machen könnten als ein Software Entwickler, der doch die Software sowieso viel besser kenn.
   Was antworten Sie ihm?
4. Sie müssen Software testen, die zur Steuerung von Raketen-Triebwerken eingesetzt wird. Welche Aspekte müssen Sie besonders testen?
5. Entwickeln Sie Tests für die Case Study Software:
   wie wollen Sie nach deren Fertigstellung deren Funktionalität und Brauchbarkeit testen?

© J.M.Joller

5.10. Literatur

Lesen Sie folgenden Artikel:
D. Gelperin , B. Hetzel , "The Growth of Software Testing" ,
Communications of the ACM **31** (June 1988) pp. 687 - 695

Die Autoren vertreten die Ansicht, dass die Zeit seit 1988 eine "Prävention orientierte Periode" ist. Inwiefern stimmt das Ihrer Ansicht nach, auf Grund des obigen Textes?

Alternative:
Lesen Sie den Artikel Lowell Jay Arthur :" Quantum Improvements in Software System Quality"
Communications of the ACM **40** (June 1997) pp. 46 -52

© J.M.Joller

© J.M.Joller