10 Nicht-funktionale Testmethoden

Wie wir in den früheren Kapiteln bereits gesehen haben, beschreibt ein Qualitätsmodell wie z.B. das von McCall unter Anderem die Qualitätskriterien einer Software. Das wichtigste Qualitätskriterium ist in den meisten Fällen die Funktionalität. Darüber hinaus existieren auch weitere, die beim Test genauso eingeplant und durchgeführt werden müssen. Dieses Kapitel behandelt die grundlegenden Eigenschaften von nicht-funktionalen Testmethoden und beschreibt einige solche Methoden anhand von Beispielen.

10.1 Eigenschaften nicht-funktionaler Testmethoden

Während beim Test der Funktionalität relativ leicht feststellbar ist, ob eine Funktion vorhanden und korrekt ist, erweist sich der Test der anderen Qualitätsmerkmale zum Teil als sehr schwierig. Dies liegt hauptsächlich daran, dass die meisten dieser Merkmale kaum quantifizierbar sind.

Bei funktionalen Tests definiert das Überdeckungsmaß einen gewissen Prozentsatz an „Erfüllung“ des Merkmals Funktionalität. Doch welches Maß soll zum Beispiel für das Qualitätsmerkmal Sicherheit herangezogen werden? In dieser Hinsicht unterscheiden sich die nicht-funktionalen Testmethoden sehr stark von den funktionalen. Zu Beginn der Planung eines nicht-funktionalen Tests steht die Frage, welche Grade der Erfüllung des Kriteriums realisierbar und auch testbar sind (bei funktionalen Tests entspricht diese Überlegung der Planung einer gewünschten Testüberdeckung).

Diese Erfüllungsgrade sind je nach Qualitätskriterium unterschiedlich. Bei der Sicherheit wird man beispielsweise verschiedene Szenarien heranziehen, welche den Grad an Sicherheit beschreiben. Diese Szenarien können zum Beispiel folgendermaßen definiert sein:

- Niedrigstes Sicherheitsniveau: ein Benutzer innerhalb des Betriebes kann versehentlich die Sicherheitsvorkehrungen umgehen.
- Ein Benutzer innerhalb des Betriebes kann mit Absicht die Sicherheitsvorkehrungen umgehen.
- Ein Benutzer kann von außen versehentlich die Sicherheitsvorkehrungen umgehen.
- Ein Benutzer kann von außen mit Absicht die Sicherheitsvorkehrungen umgehen.

Es gibt allerdings keine eindeutige Weise, wie die Erfüllungsgrade für ein Kriterium definiert werden sollen. Beispielsweise könnte man Sicherheitsgrade auch über die Zeit festlegen, welche Benutzer (oder Hacker) unterschiedlicher Erfahrung zur Umgehung einer Sicherheitsvorkehrung benötigen. Manche Entwicklungsmodelle wie z.B. Rational Unified Process ([RUPIBM], [Kruchten, 2000]) beschreiben auch die Methoden zum Test von nicht-funktionalen Qualitätsmerkmalen, inklusive Vorschlägen zur Definition der Erfüllungsgrade.

10.2 Ausgewählte nicht-funktionale Testmethoden

In den folgenden Kapiteln werden dynamische Testmethoden für nicht-funktionale Qualitätsanforderungen beschrieben. Für viele dieser Testmethoden gibt es auch Werkzeuge zur Unterstützung des Tests. Gute Aufstellungen über aktuell am Markt erhältliche Testwerkzeuge aller Art sind im Web zu finden, z.B. [APTEST].

10.2.1 Verwendung von Checklisten

Für manche der nicht-funktionalen Qualitätsanforderungen kann ein dynamischer Test nicht durchgeführt werden. Zum einen können solche Tests sehr teuer sein, vor Allem, wenn man sie in Beziehung zur Relevanz des Merkmals und zum Nutzen des Tests setzt. Als Beispiele sei hier das Qualitätsmerkmal Sicherheit genannt: in den wenigsten Anwendungen ist die Sicherheit wirklich ausschlaggebend, und die Schwierigkeit eines wirklich verlässlichen Tests macht einen dynamischen Test teuer.

Zum anderen ist ein dynamischer Test für manche Qualitätsmerkmale nicht durchführbar oder nicht sinnvoll. Hier sind Wartbarkeit und Testbarkeit repräsentative Beispiele. Die Ausführung eines Programms gibt keinerlei Aufschluss über Aspekte der Wartbarkeit wie zum Beispiel Inline-Dokumentation oder Modularität. Testbarkeit kann zwar an sich dynamisch getestet werden – man merkt schnell, ob etwas leicht zu testen ist oder nicht – aber dann ist es schon zu spät, um rechtzeitig gegenzusteuern.

In solchen Umständen empfiehlt es sich, eine statische Testmethode anzuwenden, nämlich die Verwendung einer Checkliste, welche die wichtigsten Aspekte eines Qualitätsmerkmals berücksichtigt. Checklisten spiegeln die gesammelte Erfahrung eines Betriebes oder eines Forschungsbereiches wieder, und sie ermöglichen einen strukturierten und objektiven Test. Es empfiehlt sich, anfangs Checklisten aus Testmodellen wie z.B. TMap [Pol et al, 2000] zu verwenden, und diese im Laufe der Zeit mit der eigenen Erfahrung anzureichern.

Checklisten sollten so aufgebaut sein, dass sie leicht abgearbeitet und nach dem Test auch ausgewertet werden können. Dazu ist es nützlich, die Checkliste in verwandte Themenbereiche einzuteilen und Checkboxen (Ja/Nein, oder Immer/manchmal/nie, etc.) zu verwenden, welche der Tester einfach abhaken und nachher zählen kann. Dadurch werden auch Tests verschiedener Testzyklen oder Projekte miteinander vergleichbar. Außerdem sollten sowohl Faktoren angeführt werden, welche das Qualitätsmerkmal positiv beeinflussen, als auch solche, die einen negativen Einfluss haben.

Eine Checkliste für das Qualitätsmerkmal „Testbarkeit“ könnte beispielsweise folgende Punkte beinhalten (Auszug aus [Pol et al, 2000]):

Funktionale Systemarchitektur:			**Notizen**
+	Ist die Aufteilung der Teilsysteme begründet?	☐	
+	Liegt eine zugängliche, konsistente und aktuelle technische Dokumentation vor?	☐	
-	Besteht eine starke Interaktion bzw. Abhängigkeit zwischen den unterschiedlichen Funktionen?	☐	
Technische Systemarchitektur:			**Notizen**
+	Werden bei der Entwicklung Standards eingesetzt?	☐	
+	Sind Kontrollprozesse („watchdogs“) vorhanden?	☐	
+	Kann sich der Benutzer Statusinformationen verschaffen?	☐	
+	Ist die Datenverarbeitung in Teilverarbeitungen aufgeteilt?	☐	
+	Werden Programmmodule verwendet?	☐	
+	Sind Eingabe, Verarbeitung und Ausgabe gesondert implementiert?	☐	
+	Sind mögliche Geräteabhängigkeiten in gesonderten Modulen implementiert?	☐	
+	Sind die wesentlichen Funktionalitäten in gesonderten Modulen untergebracht?	☐	
+	Sind I/O-Vorgänge in gesonderten Modulen untergebracht, um die Datenkonzepte von der Datenverarbeitung zu trennen?	☐	
+	Sind die Programme transparent und strukturiert aufgestellt?	☐	
+	Liegt eine zugängliche, konsistente und aktuelle technische Dokumentation vor?	☐	
-	Sind die Teilsysteme auf verschiedenen Rechnern verteilt?	☐	
-	Sind die Algorithmen optimiert?	☐	
-	Wird die Verarbeitung der Daten (z.B. aus Sicherheitsgründen) doppelt ausgeführt?	☐	

Tabelle 10.1: Checkliste Testbarkeit (Auszug aus [Pol et al, 2000])

10.2.2 Massentest

Bei einem Massentest prüft der Tester das Systemverhalten bei Verarbeitung von großen Datenmengen und/oder über lange Zeiträume hinweg. Dadurch kann er Instabilitäten im System finden, die beim Test einer einzelnen Funktion nicht auftreten würden, wie beispielsweise Memory

Leaks. Daher werden Massentests verwendet, um die Zuverlässigkeit und das Verbrauchsverhalten eines Systems zu testen.

In den meisten Fällen wird eine konstante oder langsam steigende Arbeitslast simuliert, doch manchmal ist auch eine dem Echtbetrieb möglichst ähnliche Last erwünscht. Für eine solche, szenariobasiert genannte Arbeitslast werden zufällig generierte Testfälle, oder solche, die aus einem Pool zufällig ausgewählt werden, herangezogen und en masse ausgeführt. Abbildung 10.1 zeigt eine typische szenariobasierte Arbeitslast für Webanwendungen, bei der die maximalen Lasten am Vormittag und Nachmittag erreicht werden und die Minimallast in der Nacht, mit einer kleinen Einsenkung in der Mittagspause.

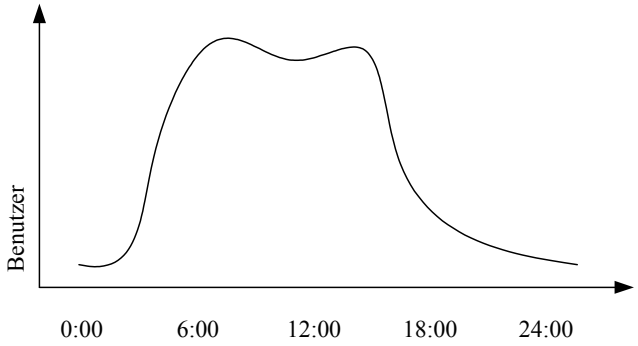

Abbildung 10.1: Szenario der Arbeitslast einer Webanwendung [Asböck, 2001]

Sobald sich bei einem solchen Test Fehler zeigen, kann die Fehlerursache genauer eingeschränkt werden. Zum Test während der Entwicklung können sodann unnatürliche Lasten eingesetzt werden, welche nur diejenigen Eigenschaften haben, welche Fehler hervorrufen, z.B. einen plötzlichen Anstieg oder Abfall der Last.

10.2.3 Lasttest

Das Ziel von Last- oder Stress-Tests ist es, Fehler zu finden, die bei paralleler Ausführung von Funktionen auftreten. Das System wird unter Last gesetzt, das heißt es muss mit einer großen Zahl von gleichzeitigen Anfragen zurechtkommen. Im Allgemeinen versucht ein Stress-Test, die maximale Zahl der Benutzer oder der parallelen Benutzeranforderungen für ein System zu ermitteln. Wenn Echtlasten für den Test gefordert werden, ist die Vorgehensweise gleich wie bei Massentests.

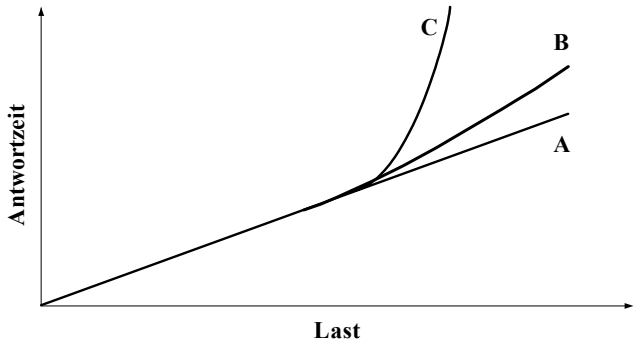

Abbildung 10.2: Szenarien steigender Last [Asböck, 2001]

Zum Test der Skalierbarkeit wird meist eine steigende Last simuliert. Abbildung 10.2 zeigt die Abhängigkeit der Antwortzeit dreier Systeme von der Last. System A zeigt eine lineare Abhängigkeit und ist skalierbar. Eine lineare Abhängigkeit charakterisiert ein perfektes System und ist in der Praxis eher nicht erreichbar. System B ist ein gut skalierbares Echtsystem mit leichter Nichtlinearität ab einem gewissen Punkt. System C hingegen zeigt stark exponentielles Verhalten ab demselben Punkt und sollte somit nur bis zu dieser Grenzlast verwendet werden.

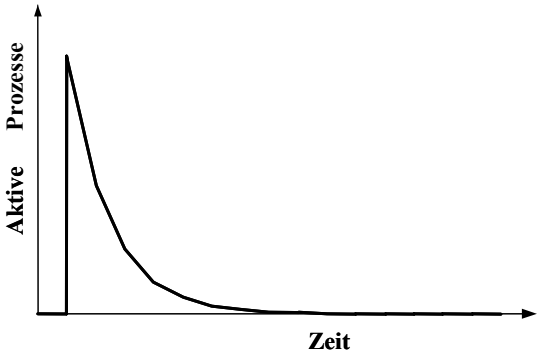

Abbildung 10.3: Sofortiger Anstieg zur Maximallast

Tests mit einer steigenden Last können systematische Fehler im Ressourcen-Zugriff und in der Ressourcen-Freigabe aufzeigen. Ihnen fehlt aber die Komponente der Gleichzeitigkeit. Um das Verhalten des Systems bei wirklich gleichzeitigen Zugriffen zu testen, kann die gesamte Last auf einmal angelegt werden, siehe Abbildung 10.3. Aus der Kurve der noch laufenden Prozesse nach einer gewissen Zeit können Rückschlüsse auf Konflikte im Ressourcen-Zugriff wie beispielsweise Deadlocks geschlossen werden.

Wenn sich Fehler mit einem Lasttest finden, also das Antwortzeitverhalten nicht den Forderungen in der Spezifikation entspricht, muss die Komponente mit der höchsten Auslastung in einem aus mehreren Komponenten bestehenden System herausgesucht werden. Diese Komponente nennt

man Engpass. Um die Leistung eines Systems zu steigern, muss die Leistung seiner Engpasskomponente(n) verbessert werden.

10.2.4 Leistungstest

Bei einem Leistungs- oder Performance-Test misst ein Tester die Antwortzeit von gegebenen zeitkritischen Funktionen. Die Antwortzeit ist die Dauer vom Ende einer Benutzereingabe bis zur entsprechenden Reaktion des Systems. Unter zeitkritische Funktionen fallen solche, deren Ergebnis zu einem bestimmten Zeitpunkt vorliegen müssen, zum Beispiel in einer automatisierten Prozesssteuerung, und solche Funktionen, die ganz einfach so oft durchgeführt werden, dass sie nicht länger als eine bestimmte Zeit dauern dürfen, um einen gewünschten Durchsatz zu erreichen, beispielsweise der Druck einer Kinokarte.

Zur Messung der Antwortzeit gibt es mehrere Möglichkeiten:

- Der Tester stoppt die Zeit manuell mit.
- Das Programm wird durch Codeinstrumentierung dahingehend erweitert, dass es selbst die Ausführungsdauer oder nur die Start- und Endzeiten am Bildschirm oder in eine Protokoll-Datei ausgibt.
- Der Tester verwendet ein funktionales Testwerkzeug, welches die Antwortzeiten im eigenen Log mitprotokolliert.

Performance-Tests werden zur Messung des Zeitverhaltens eines Systems unter optimalen Bedingungen durchgeführt, d.h. kein anderer Benutzer arbeitet zum Zeitpunkt des Tests mit dem System. Dies bedingt, dass Leistungstests entweder auf einem eigens dafür bereitgestellten System durchgeführt werden müssen, oder zu einem Zeitpunkt, zu dem keine anderen Benutzer darauf arbeiten, zum Beispiel nachts oder am Wochenende. In letzterem Fall empfiehlt sich die Verwendung von Zeitsteuerungsmechanismen wie „*at*“ oder „*chron*“ unter Unix und Vergleichbarem unter anderen Betriebssystemen.

10.2.5 Usability-Test

Usability-Tests werden ausgeführt, um zu gewährleisten, dass die voraussichtliche Benutzergruppe ihre Aufgaben effizient, effektiv und zufrieden stellend durchführen kann. Dies ist allerdings ein recht schwammiges Ziel, weshalb gerade hier die Verwendung einer gewissen Methodik angebracht ist.

Prinzipiell stehen verschiedenste Möglichkeiten zur Durchführung eines Usability-Tests zur Verfügung:

- Repräsentanten der zukünftigen Benutzergruppe können die Bedienung des Produkts an einem Prototypen ausprobieren. Durch Messung der Zeit, die sie zur Bewältigung bestimmter Aufgaben benötigen, können Tests miteinander vergleichbar gemacht werden.
- Die Tester versetzen sich in die Lage des Endbenutzers und erstellen so genannte Szenarios, welche die Interaktion des Benutzers mit dem System beschreiben. Die Szenario-Erstellung

geschieht am besten in einem Workshop, bei dem Entwickler, Tester und Benutzer zusammenarbeiten.

- Natürlich kann der Tester auch eine Checkliste verwenden, welche Aspekte beinhaltet, die die Benutzbarkeit eines Systems beeinflussen.

Im Allgemeinen wird ein Usability-Test sehr früh im Entwicklungsprozess durchgeführt. Der Grund dafür ist, dass die Benutzbarkeit sehr stark durch den System-Entwurf beeinflusst wird, und es sollte nach dem Test noch genug Zeit sein, den Entwurf anzupassen. Weiters empfiehlt es sich auch, Richtlinien (Guidelines) und Standards einzurichten, z.B. basierend auf internationalen Standards, und diese während der Entwicklung einzuhalten.

10.2.6 Recovery-Test

Tests auf Wiederherstellbarkeit werden durchgeführt, um sicherzustellen, dass ein System nach einer Hardware-Wiederherstellung die Bearbeitung ohne Verletzung der Transaktionslogik wieder aufnimmt [Beizer, 1984]. Das Ziel von Recovery-Tests ist es, Gewissheit zu erlangen, dass:

- keine Daten verloren gehen,
- Daten nicht unwissentlich dupliziert werden,
- und dass Daten nicht korrumpiert werden.

10.2.7 Sicherheitstest

Die elementarste Form von Verletzung der Sicherheit ist Sabotage – das Einstellen der Systemdienste für berechtigte Benutzer durch den Saboteur [Beizer, 1984]. Darüber hinaus kann ein böswilliger Benutzer Daten, die nicht für ihn gedacht sind, lesen und/oder manipulieren. Das Problem beim Beweis, dass ein System sicher ist, ist ähnlich dem, zu beweisen, dass ein System fehlerfrei ist. Dynamische Sicherheitstests sind üblicherweise sehr teuer, und werden daher auch nur ausgeführt, wenn Sicherheit wirklich relevant ist.

Um die Effektivität von Sicherheitsmechanismen zu bestätigen, werden oft *Penetrationstests* durchgeführt [BSI, 2003]. Penetrationstests sind nach allen anderen Tests durchzuführen, da sich aus diesen Tests Hinweise auf potentielle Schwachstellen ergeben können.

Mit Penetrationstests soll das Produkt auf Konstruktionsschwachstellen untersucht werden, indem dieselben Methoden angewandt werden, die auch ein potentieller Angreifer zur Ausnutzung von Schwachstellen benutzen würde, wie zum Beispiel

- Ändern der vordefinierten Befehlsabfolge,
- Ausführen einer zusätzlichen Funktion,
- Direktes oder indirektes Lesen, Schreiben oder Modifizieren interner Daten,
- Ausführen von Daten, deren Ausführung nicht vorgesehen ist,

- Verwenden einer Funktion in einem unerwarteten Kontext oder für einen unerwarteten Zweck,
- Aktivieren der Fehlerüberbrückung,
- Nutzen der Verzögerung zwischen dem Zeitpunkt der Überprüfung und dem Zeitpunkt der Verwendung,
- Unterbrechen der Abfolge durch Interrupts, oder
- Erzeugen einer unerwarteten Eingabe für eine Funktion.

Zur Bestimmung der Wirksamkeit eines Sicherheitsmechanismus können folgende Regeln angewandt werden:

- Kann der Mechanismus innerhalb von Minuten von einem Laien allein überwunden werden, dann kann er *nicht einmal als niedrig* eingestuft werden.
- Kann ein erfolgreicher Angriff von jedem bis auf einen Laien innerhalb von Minuten durchgeführt werden, dann ist der Mechanismus als *niedrig* einzustufen.
- Wenn für einen erfolgreichen Angriff ein Experte benötigt wird, der mit der vorhandenen Ausstattung Tage braucht, dann ist der Mechanismus als *mittel* einzustufen.
- Kann der Mechanismus nur von einem Experten mit Sonderausstattung überwunden werden, der dafür Monate braucht und eine geheime Absprache mit einem Systemverwalter treffen muss, dann ist er als *hoch* einzustufen.

Eine kostengünstigere und dennoch effektive Methode stellt die Verwendung von Checklisten dar. Beispielhafte Checklisten für Sicherheitstests finden sich z.B. in [Pol et al, 2000] oder [BSI, 2003].

10.2.8 Installationstest

Dieser Test hat zum Ziel, ein System auf seine Installierbarkeit hin zu überprüfen. Die Installierbarkeit hat zwar während des Betriebs des Systems keinen Einfluss auf die Kundenzufriedenheit, stellt aber in den meisten Fällen den Erstkontakt des Kunden mit dem System dar. Auch hier gilt, dass der erste Eindruck zählt. Daher ist eine funktionierende und benutzerfreundliche Installation des Systems wichtig.

Tests der Installierbarkeit werden meist dynamisch anhand von Probe-Installationen durchgeführt. Sie gestalten sich durch den Umstand schwierig, dass für ein aussagekräftiges Ergebnis auf dem Zielsystem das zu installierende Produkt weder bereits installiert noch jemals darauf installiert und wieder deinstalliert worden sein darf. Die am häufigsten auftretenden Fehler betreffen nicht inkludierte Installationsdateien, z.B. DLL-Dateien, weil Installationsroutinen nur auf Entwicklungsrechnern oder anderen Rechnern mit viel installierter Software ausgetestet wurden.

10.3 Zusammenfassung

Beim Testen von funktionalen Qualitätsmerkmalen ist es relativ einfach festzustellen, ob eine bestimmte Funktion vorhanden und korrekt ist. Bei nicht-funktionalen Merkmalen muss erst

einmal ein Maß gefunden werden, weil sie meist nicht oder nur sehr schwer quantifizierbar sind. Deshalb müssen andere Wege gefunden werden, um Erfüllungskriterien zu definieren.

Als Beispiel werden hier einige ausgewählte dynamische Testmethoden für nicht-funktionale Qualitätsanforderungen zusammengefasst:

- *Checklisten*
  Oftmals sind dynamische Testmethoden einfach zu teuer oder zu aufwendig (z.B. Wartbarkeit, Sicherheit, etc.). Deshalb kann man hier auf statische Checklisten zurückgreifen, die das gesamte Wissen eines Unternehmens(teils) widerspiegeln. Sie sollten einfach abzuarbeiten sein und es sollen Faktoren aufgeführt werden, die die Qualitätsmerkmale positiv oder negativ beeinflussen.

- *Massentest*
  Bei einem Massentest wird das Systemverhalten bei Verarbeitung von großen Datenmengen und/oder über lange Zeiträume hinweg überprüft. Dadurch können Instabilitäten im System gefunden werden, die beim Test einzelner Komponenten nicht auftreten. Hierbei wird entweder mit konstanter, steigender oder szenariobasierter Last getestet.

- *Lasttest*
  Beim Last- oder Stresstest wird getestet wie viele parallele Ausführungen das System verträgt (z.B. maximale Zahl der (gleichzeitigen) Benutzer). Tests mit steigender Last können Fehler im Ressourcen-Zugriff oder der –Freigabe aufdecken. Meist können diese relativ schnell auf eine bestimmte Komponente zurückgeführt werden – den so genannten Engpass. Um die Leistung eines Systems zu steigern muss die einzelne Leistung der Engpasskomponenten verbessert werden.

- *Leistungstest*
  Bei Leistungs- oder Performance-Tests wird die Antwortzeit von zeitkritischen Funktionen getestet. Dies geschieht entweder durch Stoppen, direkt eingebaute Zeitmess-Funktionen oder ein externes Testwerkzeug. So können Funktionen identifiziert werden, die der geforderten Antwortzeit hinterherhinken.

- *Usability-Test*
  Usability-Tests werden durchgeführt um zu überprüfen, ob die künftige Benutzergruppe ihre Aufgaben effizient, effektiv und zufrieden stellend durchführen kann. Dazu gibt es verschiedene Möglichkeiten: Man lässt Repräsentanten der Benutzergruppe einen Prototyp ausprobieren oder die Tester erstellen gemeinsam mit Entwicklern und Benutzern ein Szenario. Auch mit Checklisten kann man die Benutzbarkeit eines Systems überprüfen.

- *Recovery-Test*
  Recovery-Tests dienen dem Zweck, sicherzustellen, dass ein System nach einer Hardware-Wiederherstellung noch korrekt funktioniert, d.h. keine Daten verloren gehen, unabsichtlich dupliziert oder manipuliert werden.

- *Sicherheitstest*
  Dynamische Sicherheitstests sind meist recht teuer und werden nur verwendet, wenn Sicherheit wirklich relevant ist, wie z.B. beim Online-Banking oder dem Bezahlen mit Kreditkarte. Um die Effektivität von eingebauten Sicherheitsmechanismen zu überprüfen

werden oft Penetrationstests eingesetzt, bei denen einfach ein oder mehrere Hackversuche unternommen werden, um Hinweise auf potentielle Schwachstellen zu finden.

- *Installationstest*
  Da der Kunde bei der Installation quasi das erste Mal mit dem Produkt in Berührung tritt, sollte die Installationsroutine benutzerfreundlich gestaltet werden. Vor allem ist hier zu testen, wie sich das Programm in allen möglichen verschiedenen Umgebungen installieren lässt. Sonst kann es passieren, dass z.B. DLL-Dateien fehlen, die zwar auf den Entwicklungsrechnern nicht aber beim Kunden vorhanden sind.

10.4 Literaturreferenzen

[APTEST] Applied Testing and Technology, Inc.: "Automated and Manual Software Testing Tools and Services", http://www.aptest.com/.

[Asböck, 2001] Asböck, Stefan: "Load testing for eConfidence", Segue Software Inc, 2001, ISBN 3-00-007359-0.

[Kruchten, 2000] Kruchten, Philippe: „The Rational Unified Process: An Introduction“, 2nd Edition, Addison-Wesley, 1999, ISBN 0201707101.

[Liggesmeyer, 2002] Liggesmeyer, Peter: „Software-Qualität: Testen, Analysieren und Verifizieren von Software“, Spektrum Akademischer Verlag, 2002, ISBN 3-8274-1118-1).

[Nielsen] Nielsen, Jakob: „useit.com: Jakob Nielsen on Usability and Web Design“, http://www.useit.com.

[Pol et al, 2000] Pol, Martin; Koomen, Tim; Spillner, Andreas: “Management und Optimierung des Testprozesses: ein praktischer Leitfaden für Testen von Software, mit TPI und TMap”, dpunkt.verlag, 2000, ISBN 3-932588-65-7.

[RUPIBM] IBM: “Rational Unified Process: Overview”, http://www-306.ibm.com/software/awdtools/rup/.

[Kruchten, 2001] Kruchten, Philippe: “What Is the Rational Unified Process?”, http://www.therationaledge.com/content/jan_01/f_rup_pk.html.

[Thaller,1997] Thaller, Georg Erwin: „Software-Test: Verifikation und Validation“, Heise Verlag, ISBN 3-88229-183-4).

[ZBGK, 2001] Zuser, Wolfgang; Biffl, Stefan; Grechenig, Thomas; Köhle, Monika: „Software Engineering mit UML und dem Unified Process“, Pearson Studium, 2001, ISBN: 3827379272.

[BSI,2003] Homepage des BSI (Bundesamt für Sicherheit in der Informationstechnik): „Testen von Standard-Software“, http://www.bsi.de/gshb/deutsch/m/m2083.htm.

10.5 Übungen und Fragen