Diplomarbeit – Software Testing

# Softwarre Testing – Introduction

Softwaretests sind ein wichtiger Bestandteil bei der Entwicklung einer Software. Sie dient dazu um die Qualität einer Software zu sichern und um Fehlerwirkungen vorzubeugen.   
Qualität ist definiert durch das Produkt, inwiefern dieses die Anforderungen erfüllt. Ein Standard für Qualitätsmanagement ist zum Beispiel das ISO 9000 oder das ISO 9126 speziell für Software.

Die ISTQB ( International Software Testing Qualification Board ) ist die führende gemeinnützige Zertifizierungsstelle für professionelle Software Tester in Europa. Ihr Ziel ist es, Softwaretests zu standardisieren.

# Allgemeine Informationen; Standards;

# Wieso Software Testen?

Eine gewisse Fehlerquote ist bei jedem Projekt zu erwarten, da Fehler nicht vollständig vermeidbar sind. Je nach Art des Fehlers, kann dieser jedoch das Produkt unbrauchbar machen, Verlust einbringen oder den Ruf einer Firma oder der Entwickler schaden.

Um dies zu vermeiden werden Produkte getestet, bevor sie dem Kunden übergeben werden. Ziel dieser Tests ist es, Fehler zu finden und die Qualität zu sichern.

#Wieso wird getestet Wieso hab ich mich dafür entschieden?

# Allgemeine Begriffe

Im Bereich der Software Tests gibt es bestimmte Grundbegriffe. Die genauen Definitionen sind sehr wichtig für das Verständnis von Softwaretests und die Literatur die sich damit befasst.

|  |  |
| --- | --- |
| Fehlerwirkung | Eine Fehlerwirkung ist die Abweichung einer Komponente/eines Systems von der erwarteten Leistung oder Lieferung oder eine Abweichung des Ist-Zustand/verhalten vom Soll-Zustand/verhalten.  p 251 Glossar |
| Fehlerzustand | Fehler/Defekt in einer Komponente oder einem System, welcher während der Laufzeit des Systems, Fehlerwirkungen hervorrufen kann. |
| Sollverhalten | Eine vorhergesagter Ausgabewert oder Verhalten eines Systems aufgrund der Spezifikation oder anderen Quelen. |
| Istverhalten | Das tatsächliche Verhalten/Ergebnis eines Systems bei einem Test unter spezifizierten Bedingungen. |
| Debugging | Tätigkeit die getan wird, um einen Fehlerzustand zu lokalisieren. |

# Testziel & Testumfang

Allgemein ist das Ziel von Softwaretests die Qualitätssicherung, unter anderem primär die Aufdeckung von Fehlerwirkungen im System. In der Praxis entwickelt ein System beim Finden von Fehlerwirkungen eine gewisse Testresistenz. Je mehr Fehlerwirkungen gefunden werden, desto schwieriger wird es, weitere zu finden. Der Testaufwand wird daher relativ zu der Dauer des Testens, immer größer. Grundsätzlich gilt, dass das komplette Testen und das Abdecken aller möglichen Eingabewerte für ein Programm, de facto unmöglich ist.

Dies lässt sich an einer einfachen Rechnung darstellen:  
  
Als Beispiel wird die Additionsfunktion eines Taschenrechners genommen, welcher bis zu 2 verschiedene ganzzahlige Werte addieren kann. Diese Zahlen werden im Programm, geschrieben in Java, als Integer dargestellt.

Ein Integer hat 2\*2^(31) mögliche Kombinationen.  
2\*2^(31) = 4294967296   
2^4294967296 sind die Anzahl der Möglichen Eingaben.

Werden automatisierte Tests genutzt und wird davon ausgegangen dass ein Test 0.002 Sekunden braucht, ergibt sich folgende Rechnung:  
2^4294967296 \* 0.002  
  
In Jahren umgerechnet, würde ein vollständiger Test mehr als 1 Milliarde Jahre dauern. Theoretisch aber nicht praktisch möglich.  
  
Um trotzdem viele Fehlerwirkungen zu finden, gibt es verschiedene Testmethoden und Teststrategien die im späteren genauer erläutert werden. Den tatsächlichen Testaufwand abzuschätzen ist jedoch sehr schwer. Auf die Frage ob sich ein Aufwand überhaupt auszahlt, kommt die Gegenfrage: „Im Vergleich zu was?“ (DeMarco 93)

Zumeist hängt der Testaufwand von dem Risiko ab. Das Risiko definiert sich durch die Höhe des Schadensfalles bei Eintritt und die Eintrittswahrscheinlichkeit. Ist das Risiko hoch, so ist ein höherer Testaufwand gerechtfertigt. Aus einem Artikel der Frankfurter Zeitung vom 17.01.2012 mit dem Titel „IT-Systemausfälle kosten viele Millionen“, geht hervor, dass der Ausfall von einer Stunde in der Börse 7,8 Mio. Dollar kostet. Bei anderen Systemen wie einem KIS ( Krankenhausinformationssystem ), kann ein Fehlverhalten sogar Menschenleben kosten.  
  
#Was ist das Ziel von Testing?   
Testobjekt?  
Testumfang?

# Softwarequalität

In einem Projekt werden Tests ausgeführt um Fehlverhalten zu finden und somit die Softwarequalität zu verbessern. Die Softwarequalität wird jedoch durch mehrere Faktoren definiert. ISO 9261 wird dabei als Standard angesehen, darin werden folgende Merkmale für die Softwarequalität definiert:

* Funktionalität
* Zuverlässigkeit
* Benutzbarkeit
* Effizienz
* Wartbarkeit
* Übertragbarkeit

## Funktionalität

Die Funktionalität definiert inwieweit die Software die geforderte Funktion erfüllt. Diese Funktionen werden oft durch ein spezifisches Ein-/Ausgabeverhalten und/oder Auswirkung auf das System beschrieben. Das Merkmal der Funktionalität kann in folgende weitere Teilmerkmale unterschieden werden:   
Richtigkeit, Interoperabilität, Sicherheit und Konformität

## Zuverlässigkeit

Offizielle Definition nach ISO/IEC 25010:2011:

„degree to which a system, product or component performs specified functions under specified conditions for a specified period of time“

ISO/IEC 25010:2011, Online in Internet: URL: <https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en>, 17.03.2016.

Zuverlässigkeit beschreibt inwiefern das System ein bestimmtes Leistungsniveau über einen bestimmten Zeitraum aufrechterhalten kann. Sie lässt sich in folgende Teilmerkmale aufteilen:  
Fehlertoleranz und Wiederherstellbarkeit.

## Benutzbarkeit

Bei Systemen die interaktiv sind und von vielen Kunden benutzt wird, spielt die Benutzbarkeit eine große Rolle. Sie beschreibt den Aufwand für die Nutzung der Software.

## Effizienz

Das Verhältnis der Leitung zu den verbrauchten Ressourcen beschreibt den Grad der Effizienz. Bei guter Leistung mit wenig Ressourcenverbrauch ist eine gute Effizienz gegeben und umgekehrt.

## Wartbarkeit

Wartbarkeit wird in der Fachliteratur auch als Änderbarkeit bezeichnet. Sie beschreibt wie gut das System modifizierbar und änderbar ist.

## Übertragbarkeit

Die Übertragbarkeit beschreibt wie gut das System sich in einer anderen Umgebung intrigieren lässt.

# Psychologie des Testens

S35

Bei der Entwicklung einer Software wird ein neues Produkt erschaffen. Softwareentwicklung wird generell als etwas Konstruktives angesehen, während Softwaretests als etwas eher Destruktives angesehen werden. Tester werden oft ungern gesehen, da diese auf unangenehme Fehler hinweisen.

Um Probleme vorzubeugen ist es wichtig, dass Probleme und Fehlerwirkungen sachlich und neutral dem Entwickler übermittelt werden. Die Fehlerwirkungen sollten als Verbesserungsvorschläge in Bezug auf die Software und nicht als Kritik an den Entwickler dargestellt werden. Entwickler und Tester sollten zwar unabhängig aber keine Feinde sein.

Die Unabhängigkeit zwischen Entwickler und Tester sind insofern wichtig, da Entwickler alle Use Cases und alle möglichen Sonderfälle schon während der Entwicklung berücksichtigen. Ein testen der schon sowieso berücksichtigten Fälle wäre weniger effizient. Ein unabhängiger Tester kann jedoch andere Testszenarien berücksichtigen und anhand der Spezifikation die Sicht des Endkunden nachvollziehen. Natürlich kann ein Entwickler als Tester auch Vorteile bringen, vor allem bei geringen finanziellen und zeitlichen Ressourcen, eine allgemein gültige Aussage was besser ist, kann nicht getroffen werden.

Im Idealfall sollte der Entwickler sachlich mit dem Tester zusammenarbeiten, um so die Qualität des Produktes zu verbessern.

#ProblemeBeiFehlerFIndung

# 7 Prinzipien des SoftwareTestens

Aus dem Buch „Software-TEST IT PROFESSION@LlY“ von Mag. Anja Kribernegg geht hervor, dass sich in den letzten Jahren Grundsätze zum Testen gebildet haben, welche als Leitlinien angesehen werden. Diese sind wie folgt:

**Grundsatz 1: Durch Tests werden Fehlerzustände gefunden**Durch das Testen sollen möglichst viele Fehlerwirkungen gefunden werden und damit die Fehlerzustände nachgewiesen werden. Absolute Fehlerfreiheit kann jedoch nicht durch das Testen nachgewiesen werden, sie verringert nur die Anwesenheit unentdeckter Fehlerzustände.

**Grundsatz 2: Keine Software kann vollständig getestet werden** Das vollständige Testen und ausführen alle möglichen Testfälle, ist nur in der Theorie aber nicht in der Praxis möglich. Tests werden mit Stichproben durchgeführt. Je vielfältiger die Stichproben, desto höher der Testaufwand. Der Testaufwand entspricht relativ dem Risiko und somit der Priorität.

**Grundsatz 3: Je früher desto besser**Das Testen einer Software sollte so früh wie möglich begonnen werden. Am vorteilhaftesten ist das Testen während der Entwicklung. Auf fehlerhaften Komponenten aufbauen, kann zu noch mehr Fehlern führen, außerdem ist Testen an sich eine sehr zeitintensive Tätigkeit.

**Grundsatz 4: Fehlerdichte beachten**Erfahrungsgemäß häufen sich die auftretenden Fehlerwirkungen in bestimmten Bereichen eines Programmes.

**Grundsatz 5: Testresistenz**Mit zunehmender Testdauer und Testaufwand steigt auch die Testresistenz. Je mehr Testfälle ausgeführt wurden, desto weniger neue Fehlerwirkungen werden gefunden.

**Grundsatz 6: Das Umfeld ist entscheidend**Je nach Umfeld, kann das Vorgehen anders sein. Generell wird eine Eismaschine anders getestet als ein Krankenhausinformationssystem (KIS). Sowohl der Aufwand als auch die verwendete Teststrategien weichen sehr stark voneinander ab.

**Grundsatz 7: Fehlerfreiheit bedeutet nicht Kundenzufriedenheit**Fehlerfreiheit kann nie garantiert werden und sie bedeutet auch nicht, dass das System dann auch den Erwartungen und Vorstellungen des Kunden entspricht.

# Testen im softwarezyklus

Die richtige Integration der Testaktivitäten in das laufende Projekt ist sehr wichtig. Das Testen während der Entwicklung hilft dem Team frühzeitig Fehler zu erkennen und Folgefehler zu vermeiden, es muss jedoch eine richtige Planung zuvor erfolgen die je nach Vorgehensweise des Projektes angepasst wird.

## Modelle

Im Bereich des Projektmanagements gibt es verschiedene Modelle die eine gewisse Vorgehensweise zur Entwicklung eines Produktes beschreiben. Ein paar der bekannteren Modelle:

* Wasserfallmodell
* Extreme Programming
* Kanban
* Scrum
* V – Modell

Für Tester ist das V-Modell nach Barry W. Boehm ein sehr interessantes und wichtiges Modell. Es setzt die Wichtigkeit von den einzelnen Teststufen mit denen der Entwicklung gleich. Boehm fokussiert mit dem V Modell 4 wesentliche Ziele an: Completeness, consistency, feasibility und testability.  
Das Modell besteht grundsätzlich aus zwei Zweigen die zusammen ein V bilden. Der linke Zweig beschreibt die einzelnen Schritte zur Erstellung einer Software, vom Design bis hin zur Programmierung. Die Schritte werden von grob immer genauer, während auf dem rechten Zweig parallel die Verifizierung und Validierung durchgeführt wird.

Das Testen endet aber nicht zwingend, nachdem eine Software auf dem Markt kommt. Bestehende Systeme werden oft Jahrzehnte lang genutzt und benötigen immer wieder Updates. Diese Updates können wiederum Fehlerwirkungen hervorrufen, die davon nicht da waren. Funktionalitäten die bei vorherigen Tests keine Fehlerwirkungen hervorgerufen haben, können nach den Updates ihr Verhalten ändern. Aus diesem Grund gibt es Regressionstest. Regressionstest sind Tests die wieder ausgeführt werden, um die Fehlerfreiheit in den schon durchgeführten Testfällen, nach einem Update zu gewährleisten. Im Zusammenhang mit Regressionstest, spielen automatische Tests (siehe Glossar) eine große Rolle.

## Komponententest

Meistens entwickelt jeder Entwickler einen Teil der Software, eine Komponente davon, die dann später mit den Komponenten anderer Entwickler zusammengeführt werden.

Der Komponententest dient dazu, diese Komponenten im einzelnem zu testen. Diese Komponenten, auch Module oder Units genannt, können wenn sie fehlerhaft sind, später bei der Integration, Fehlermaskierungen hervorurrufen.

In zahlreichen Programmiersprache wie zum Beispiel C#, gibt es eigene Frameworks wie NUnit die sich mit automatisierten Unit Tests auseinandersetzen.

## Integrationstest

Integrationstests befassen sich mit der Zusammenarbeit der einzelnen Komponenten untereinander. Ziel des Integrationstestes ist es, Fehlerwirkungen zwischen den Komponenten zu finden.

Auch wenn die einzelnen Komponenten fehlerfrei funktionieren, können bei der Integration der einzelnen Teile in das System, Fehler entstehen. Komponente A könnte Komponente B irrtümlich falsche Datentypen übergeben, oder richtige Datentypen im falschen Einheitsmaß.

Ein gutes Beispiel für einen schweren Fehler der beim Integrationstest übersehen wurde, ist der Mars Climate Orbiter (MCO) bei dem eine Sonde abstürzte. Der Grund war eine Berechnung im falschen Einheitensystem. Im offiziellen Bericht hieß es:

„Findings of the failure review board indicate that a navigation error resulted from some spacecraft commands being sent in English units instead of being converted to metric. This caused the spacecraft to miss its intended 140 - 150 km altitude above Mars during orbit insertion, instead entering the martian atmosphere at about 57 km. “

## Systemtest

Der Systemtest befasst sich mit dem System als Ganzes. Während der Integrationstest die Interaktion zwischen den einzelnen Komponenten aus technischer Sicht testet, wird im Systemtest die Software als Einheit gegen die Produktdefinition geprüft. Wie die Software aufgebaut ist, wird dabei nicht berücksichtig.

## AbnahmeTest

Der Abnahmetest ähnelt dem Systemtest, jedoch dient der Abnahmetest nicht zur Qualitätssicherung, sondern zum Nachweis, dass die geforderten Funktionen gesetzlich erfüllt worden sind. In den meisten Ländern ist die Abnahme eines Produktes gesetzlich geregelt.

So heißt es zum Beispiel im deutschen Bürgerliches Gesetzbuch (BGB) Paragraph 640:

„Der Besteller ist verpflichtet, das vertragsmäßig hergestellte Werk abzunehmen, sofern nicht nach der Beschaffenheit des Werkes die Abnahme ausgeschlossen ist. Wegen unwesentlicher Mängel kann die Abnahme nicht verweigert werden.”

(BGB01, Parag. 640, Abs. 1, Satz 1 und 2)

Die Abnahme wird zusammen mit dem Kunden und einem Abnahmeprotokoll durchgeführt. Erfüllt das Produkt vertraglich alle Anforderungen, wird dies mit einer Unterschrift bestätigt.

# Grundlegende Softwaretestvefahren

Es kann während der verschiedenen Teststufen, zwischen verschiedene Testarten unterschieden werden. Je nach Teststrategie und Test Ziel, werden verschiedene Testarten genutzt.

## Funktionale und nicht funktionale Tests

Die einfachste Unterscheidung ist die Unterscheidung zwischen funktionalen und nicht funktionalen Tests. Softwarequalität befasst sich nicht nur mit dem Bereich der Funktionalität. Diese wird mit funktionalen Tests geprüft, es gibt aber auch andere Merkmale der Qualität wie Benutzbarkeit und Zuverlässigkeit. Diese werden mit nicht funktionalen Tests geprüft. Nichtfunktionale Tests sind schwieriger zu planen als funktionale Tests, da die Sollwerte von den nichtfunktionalen Tests keine allgemein gültige Metrik haben und deshalb auch keine eindeutig definierbare Sollwerte.

Ein gutes Beispiel sind Videospiele, bei denen Benutzerfreundlichkeit eine wichtige Rolle spielt. Was jedoch benutzerfreundlich ist, ist jedoch schwer definierbar.

## Statische und Dynamische Tests

Eine weitere Unterteilung kann mit statischen und dynamischen Tests gemacht werden. Bei statischen Testverfahren wird das System getestet, ohne es auszuführen, während bei der dynamischen Variante das System während der Laufzeit geprüft wird.

### Statische Testverfahren

Statische Testverfahren werden vor der Implementierung und Ausführung der Software ausgeführt. Sie werden unter anderem auch „Menschenbasierte“ Testverfahren genannt und haben den entscheidenden Vorteil, dass diese schon frühzeitig durchgeführt werden können. Die hat den Vorteil dass weitere Fehler vermieden werden können und die Programmierung noch während der Entwicklung optimiert werden. Hinzu kommt noch, dass statische Tests meist nicht die Fehlerwirkungen sondern die Fehzustände aufdecken, was die Behebung des Problems beschleunigt. Die Erfahrung zeigt, dass statische Tests sehr effizient sind.

#### Diverse ReviewArten

Reviews stellen eine Art von statischen Testverfahren da, welche schon frühzeitig genutzt werden können. Dabei werden Sitzungen unter Aufsicht eines Moderators abgehalten, bei denen die Dokumente und Codes der Mitarbeiter besprochen werden.

Der Ersteller des Dokumentes/Codes ist der Autor und dessen Arbeit wird von einem anderen, einem Gutachter, nach den Richtlinien geprüft. Das Ergebnis wird zusammen mit dem Moderator und anderen Gutachten bei der Sitzung präsentiert.

Einige wichtige Richtlinien für Reviews nach Karol Frühlauf:

* Ein Review sollte nicht länger als 2 Stunden dauern. Falls nötig wird, frühsten am nächsten Tag, eine weitere Sitzungen einberufen.
* Der Moderator hat das Recht eine Sitzung abzusagen, wenn Leute fehlen oder schlecht vorbereitet sind.
* Nicht der Autor, sondern das Testobjekt steht zur Diskussion. Auf die richtige Formulierung ist zu achten.
* Jeder Gutachter muss ausreichend zu Wort kommen.
* Lösungen und Stilfragen stehen nicht zur Diskussion.

Je nach Wunsch und Art des Projektes, gibt es verschiedene Arten von Reviews, die sich teilweise unterscheiden. Manche Reviews sind förmlicher, manche weniger.

Ein paar der wichtigsten Review Arten:

* Walkthrough
* Inspektion
* Technischer Review
* Informelles Review

#### Statische analysewerkzeuge

Neben den Reviews gibt es auch noch statische Analysewerkzeuge. Es sind Werkzeuge, wie zum Beispiel ein Compiler, der die Prüfobjekte automatisch auf Fehler prüft.

Fehler die durch statische Analysewerkzeuge geprüft werden, können zum Beispiel Syntax Errors, Verletzung der Programmierkonventionen oder ungenutzte bzw. falsch genutzte Variablen.

Da statische Analysewerkzeuge schnell, effizient und zumeist wenig aufwendig sind, sind sie für gewöhnlich in jedem Projekt anzutreffen.

### Dynamische Tests

Dynamische Tests werden während der Laufzeit des Systems ausgeführt. Idealerweise in der Umgebung in der das System ihren Einsatz findet und auch mit derselben Auslastung.

Diese dynamischen Tests haben natürlich einen viel höheren Testaufwand sind jedoch dementsprechend „realitätsnah“. Es können Situationen geprüft werden und Fehler gefunden werden, die es mit statischen Tests überhaupt nicht gibt, wie zum Beispiel falsche Eingaben und Andere.

Generell kann bei dynamischen Tests zwischen White Box und Black Box Tests unterschieden werden.

#### White box

Beim White Box Test wird davon ausgegangen, dass die innere technische Struktur und Architektur des Systems bekannt ist. Der Programmtext muss vorliegen und wird, falls notwendig manipuliert. Grundlegend wird im Idealfall versucht, alle Stellen des Codes zumindest einmal auszuführen. Entsprechend dem Jargon gibt es den Begriff des Code Coverages, welcher angibt, wie viel (%) vom Code während der Tests ausgeführt wurde.

Es lassen sich aus dem White Box Test folgende weitere Testfallentwurfsmethoden unterscheiden:

* Anweisungstest
* Entscheidungstest
* Bedingungstest
* Pfadtest

#### Black Box

Beim Black Box Test wird die Struktur und Architektur des Programmes nicht heran gezogen bzw. sie ist unbekannt. Daraus folgt, dass die Spezifikationen zum Herleiten der Testfälle genutzt werden. Wenn alle Kombinationen der Eingabewerte durchgeführt und getestet wurden, gilt das System zu 100% getestet. Dies ist jedoch leider nicht möglich (siehe kapitel Testaufwand).

Zum Bilden der Testfälle aus den Spezifikationen, können 2 verschiedene Methoden genutzt werden:

* Äquivalenzklassenbildung
* Zustandstests
* Exploratives Testen
* Geschäftsprozess-Test

# Testmanagement

Ein Management hat die Aufgabe Ressourcen optimal zu nutzen, die einzelnen Tester zu Koordinieren und somit möglichst viele Fehlerwirkungen zu finden. Typischerweise werden im Laufe eines Projektes einige Stufen durchlaufen und Dokumente erstellt, die nun genauer erläutert werden. Je nach Projektmodell, kann es speziell bei agilen Vorgehensweisen sein, dass einzelne Schritte sich wiederholen.

Anmerkung: Je nach Standard können sich Details, Inhalt und Gliederung der einzelnen Dokumente ändern. Dies dient nur zur Übersicht.

## Analyse

In einem allgemeinen Projekt ist der erste Schritt oft die Machbarkeitsstudie. Jegliche Planung ist sinnlos, wenn dann das Projekt nicht realisierbar ist. Dementsprechend wird vom Tester immer am Anfang als erstes geprüft, ob das Produkt überhaupt testbar ist.

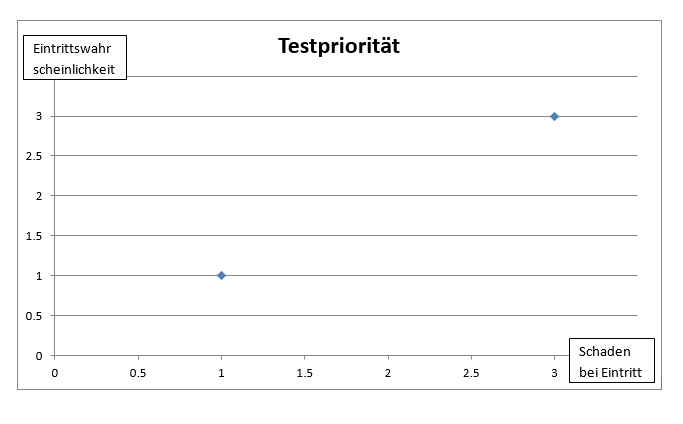
Fehlen Spezifikationen oder andere Dokumente, bzw. sind diese zu ungenau, muss dies angepasst werden. Ungenaue oder unvollständige Dokumente führen zu Verwirrung und großen Zeitverlust.

## Testkonzept

Ist die Analyse abgeschlossen und die Testbarkeit gegeben, wird das Testkonzept geschrieben. Eine sorgfältige Planung ist wichtig, das Testkonzept bildet das Fundament aller anderen Schritte. Im Testkonzept wird unter anderem das Testobjekt definiert.

Nicht immer ist klar, was das Testobjekt ist. Manchmal soll nur ein Teil eines Systems getestet werden, oder der Umfang ist nicht einzuschätzen. Bei einer Mobile App müsste zum Beispiel genau definiert werden, auf welchen Geräten sie getestet werden soll, eine Aussage wie „auf allen“ ist nicht spezifisch. Auch aus rechtlichen Gründen ist es ratsam das Testobjekt mit dem Auftraggeber zu definieren und das dann unterschreiben zu lassen.

Da ein System in der Praxis nicht zu 100% getestet werden kann, ist wichtig festzulegen, welche Teile zuerst getestet werden sollen. Dies geschieht anhand einer Testpriorität. Die Testpriorität setzt sich aus der Eintrittswahrscheinlichkeit einer Fehlerwirkung und dem Schaden beim Eintritt einer Fehlerwirkung zusammen. Folgend eine Grafik:



Je höher Eintrittswahrscheinlichkeit und der Schaden beim Eintritt sind, desto höher die Priorität. Faktoren wie Nutzungshäufigkeit, Wahrnehmung, Priorität und Komplexität spielen auch eine Rolle, lassen sich jedoch in den ersten zwei genannten Faktoren wiederspiegeln.

Weiteres sollte das Konzept einen Zeitplan und eine Verantwortungsmatrix beinhalten, welcher mit Entwicklerteam bzw. Projektteam abgestimmt ist. Ebenso sollte die grundlegende Teststrategie und die zu verwendeten Tools definiert werden. Kriterien für den Teststart und Testende sollten eventuell ebenfalls definiert werden.

Eine Zusammenfassung der wesentlichen Punkte im Testkonzept:

* Testobjekt
* Testpriorität
* Zeitplan
* Verantwortungsmatrix
* Teststrategie
* Tools
* Kriterien für Teststart und Testende

## Kriterien für Testart

Bevor mit dem eigentlichen Testen gestartet werden kann, müssen einige Kriterien erfüllt werden:

* Testumgebung bereit
* Testwerkzeuge sind bereit
* Testobjekt steht zur Verfügung
* Notwendige Testdaten sind vorhanden

Diese Grundvoraussetzungen sollten erfüllt sein, da ein Testen sonst nicht möglich ist. Es ist ratsam diese Punkte vor dem Start rechtzeitig zu prüfen, damit es zu keiner Verzögerung im Zeitplan kommt.

## Test Design und Testfallspezifikation

Die notwendigen Testdaten für den Start der Tests, setzen sich zumeist aus den Dokumenten Test Design und der Testfallspezifikation zusammen.

Das Test Design ist vor allem bei agilen Projekten sehr wichtig. Es definiert noch einmal das Testobjekt, dabei kann im Design ein anderes Testobjekt sein, als im Testkonzept beschrieben. Im Agilen Verfahren wie SCRUM wird in Zyklen entwickelt und getestet, so kann das zu testende Objekt nur ein Teil vom ganzen sein. Anders als beim Testkonzept, wir nun genauer definiert wie getestet werden soll.

Bei der Testfallspezifikation werden schlussendlich die einzelnen Testfälle definiert. Dabei ist darauf zu achten, dass zumindest folgende Informationen für jeden Testfall gegeben sind:

* Testumgebung
* Eingabewert
* Erwartetes Ergebnis
* Ersteller des Testfalles

Vor allem der Eingabewert und das erwartende Ergebnis sind wichtig, um zu erkennen ob der Testfall ein Erfolg war, oder nicht.

## Testprotokoll und Testauswertung

Wenn das Test Design und die Testfallspezifikation fertig sind, landet letzteres beim Tester, welche die einzelnen Testfälle durchführt. Dabei ist zu beachten, dass dieser sich strikt an die Spezifikation hält und alles protokollier. Zusätzlich zu den Informationen die in der Testfallspezifikation zu den einzelnen Testfällen steht, kommen nun im Protokoll folgende Infos dazu:

* Ausführender Tester
* Tatsächliches Ergebnis
* Ausführungszeitpunkt

Ein Protokoll zu einem einzelnen Testfall kann dann zirka so aussehen:

|  |
| --- |
| Test case: #00008 |
| Test case name: Test00008 |
| Designed by: Name |
| Executed by: Name |
| Design date:2016.01.21 |
| Execution date: 2016.01.21 |
| Status: |
| Input: Input |
| Expected output:  Expected output |
| Actual Output:  Actual Output |
| Note: |

Natürlich sind in diesem Beispiel mehr Informationen enthalten und die Testumgebung wurde am Anfang des Dokumentes für mehrere Testfälle definiert, jedoch ist dies ein gutes Beispiel.

Aber nicht nur die Durchführung sondern auch die Auswertung der Tests ist wichtig. Vor allem wenn automatisierte Tests genutzt werden, muss sich der Tester jeden einzelnen Testfall noch einmal anschauen und selbst überprüfen ob es tatsächlich ein Fehler ist. Empfehlenswert ist es, zu überprüfen, ob nicht im Testfall selbst ein Fehler ist. Nichts ist peinlicher als ein Fehler im Testfall.

## Fehlermanagement

Wenn Fehler gefunden werden, sollten diese so schnell wie möglich behoben werden. Obwohl das Debugging die Aufgabe des Entwicklers ist, kann der Tester diesen dabei sehr unterstützen.

Wichtig ist dabei das Einteilen der Fehler in Kategorien. Ein Fehler ist nicht gleich ein Fehler, so sind fehlerhafte Daten in einer Datenbank für ein Krankenaus etwas anderes, als eine falsche Farbe beim Drücken eines Buttons.

Die Kategorien könnten so aussehen:

|  |  |
| --- | --- |
| Klasse | Bedeutung |
| 1 | Menschenleben in Gefahr |
| 2 | Großer Image- oder Geldschaden |
| 3 | Abweichung von der Spezifikation |
| 4 | Unklar |
| 5 | Schönheitsfehler |

Zusammen mit einem gut geführten Testprotokoll kann der Entwickler schnell die Fehler abarbeiten.

## Kriterien zum Test ENde

Da ein System nie zu 100% getestet werden kann (siehe Testaufwand), kann theoretisch unendlich getestet werden, deswegen sind Kriterien zum Ende der Tests wichtig.

Diese könnten zum Beispiel sein:

* Erreichter Testumfang
* Erreichte Produktqualität
* Verbleibendes Risiko
* Wirtschaftliche Rahmenbedingungen

Die ersten drei Punkte werden in der Praxis eher weniger eintreten. Wahrscheinlicher ist es, dass die Ressourcen zu Ende gehen. Meist geht die zeitliche Ressource mit Ende des Projektes zu Ende, oder die finanziellen Mittel gehen aus.

Das Erreichen der gewünschten Produktqualität dürfte zumeist nur bei Systemen vorkommen, welche für Menschenleben verantwortlich sind.

# Sofrwaretesting – Doungen

Das Testobjekt im Projekt war natürlich das Produkt/Spiel „Doungen“. Die Aufgabe und das Ziel war es, die einzelnen Komponenten, welche nach jedem Sprint hinzugefügt wurden, zu testen und Fehlerwirkungen zu finden. Da das Team aus einem Tester und 5 Entwicklern bestand und unter Bedacht, dass die graphische Oberfläche bei einem Spiel einer der wichtigsten Punkte ist, wurde der Fokus darauf gelegt, das Spiel mittels Black Box Verfahren zu testen. White Box Tests wurden nur speziell auf Anfrage durchgeführt.

Als Vorgehensweise wurde

