BSZ Reichenbach
Z.H. Herr Dr. Bensing
Rathenau Straße 2

08468 Reichenbach

Facharbeit Informationstechnik

Thema: Leitfaden für Softwaretests

Abgabedatum: 01.03.2006

Autor: Frank Richter

Frank Richter
Gamigstraße 5

01239 Dresden

Kontakt

Tel.: 0179 32 42 973
Mail: Frank.Richter@v-ice.org
Web: http://v-ice.org

Autor: Frank Richter
Betreuer: Herr Dr. Bensing

Softwaretest

Maßgaben zum Qualitativen Software-/ Hardwaretesten

Anmerkung:

Dieses Dokument bleibt geistiges Eigentum des Autors bzw. Urhebers und darf ohne Genehmigung weder weitergegeben noch kopiert werden. In der Regel reicht eine Email an Frank.Richter@v-ice.org Vielen Dank!

Einführung
Software wie auch Hardware unterliegen einer Reihe von Qualitätsmerkmalen, die es zu erfüllen gilt. Um nun ein Maß an zu Qualität finden, verschiedene Produkte oder Produktionszyklen zu vergleichen und letztendlich auch eine Aussage über die Erfüllung einer Anforderung zu geben werden verschiedene Testszenarien benötigt. Im Folgenden wird in dieser Arbeit aufgezeigt, auf welche Art und Weise speziell Software auf solche Anforderungen geprüft werden kann und eine Art Leitfaden gegeben, der es Schülern wie Lehrern ermöglichen sollte selbst Produkte mit Anforderungen vergleichen zu können und Aussagen darüber zu treffen wie gut die Qualität verschiedener Projekte ist um damit einen Teil des QM[1] Praktisch anwenden zu können.

Softwaretest als Bestandteil des QM

→ Qualität ist ein Maß für das Erreichen von Vorgaben!

Grundgedanke ist, dass jede Anforderung bzw. Vorgabe von z.B. Kunden in Form eines Pflichtenheftes als Qualitätsmerkmal definiert werden kann.

Das können z.B.: Vorgaben sein, wie garantierte Laufzeiten eines Computersystems oder DAU[2] Absicherung von Eingabemasken.
Diese Vorgaben werden dann als Anforderung in das zu programmierende Projekt übernommen. Genau diese Anforderungen sind dann auch gleich das zu testende Kriterium.
Werden diese 100% erfüllt, ist auch die Qualität dieser einen(!) Anforderung zu 100% erfüllt.

Also gilt hierbei:
100% erfüllte Anforderungen = 100% Qualität
80% erfüllte Anforderungen = 80% Qualität

Bsp.
Ausgangsituation/Anforderung:
Die Hintergrundfarbe der Webseite bsz.reichenbach.de sollte das gleiche Grau enthalten wie der Hintergrund der Seite fos.reichenbach.de

Vergleich durch Softwaretest:
Der Farbton ist nicht ganz der Gleiche. Bei bsz.reichenbach.de wurde eine RGB Farbpalette verwendet, bei fos.reichenbach.de CMYK
Ergebnis:
Also ist die Anforderung das gleiche Layout wieder zu verwenden nicht ganz erfüllt und damit nicht 100% Qualität.

Black Box Test

Black Box Tests sind Tests, die von externen Personen oder Automaten durchgeführt werden, die keinen Sachverstand der Technik haben müssen. Bei einem Black Box Test werden nur äußere Einflüsse und alle von außen sichtbare Reaktion auf die Umwelt untersucht. Das hat den Vorteil, dass die Tester nicht nach evtl. schon im White Box Test[3] gestesteten Vorgehensweisen verfahren. Hierbei wird von einer breiten Masse getestet und ohne rein technologisches Know How weiterzugeben.

Beispiel:
Eine bekannte Firma gibt eine spezielle Version ihres Betriebssystems vor dem final release[4] als sog. Betaversion[5] an eine breite Gruppe von Personen heraus. Diese müssen dann während der Laufzeit ein Formular ausfüllen und diverse aufgetretene Fehler oder Probleme beschreiben und erläutern. Dieses Verfahren ist dann eine Variante des so genannten Blackbox Tests – „keiner weiß was drin ist“

Testablauf Black Box Test

Vorgehensweise beim Black Box Test, vergleiche auch Anhang A – Schema eines Black Box Tests.
Zu testende Software wird in ein Testbett übertragen und daraus der Ablauf für den Tester entwickelt.

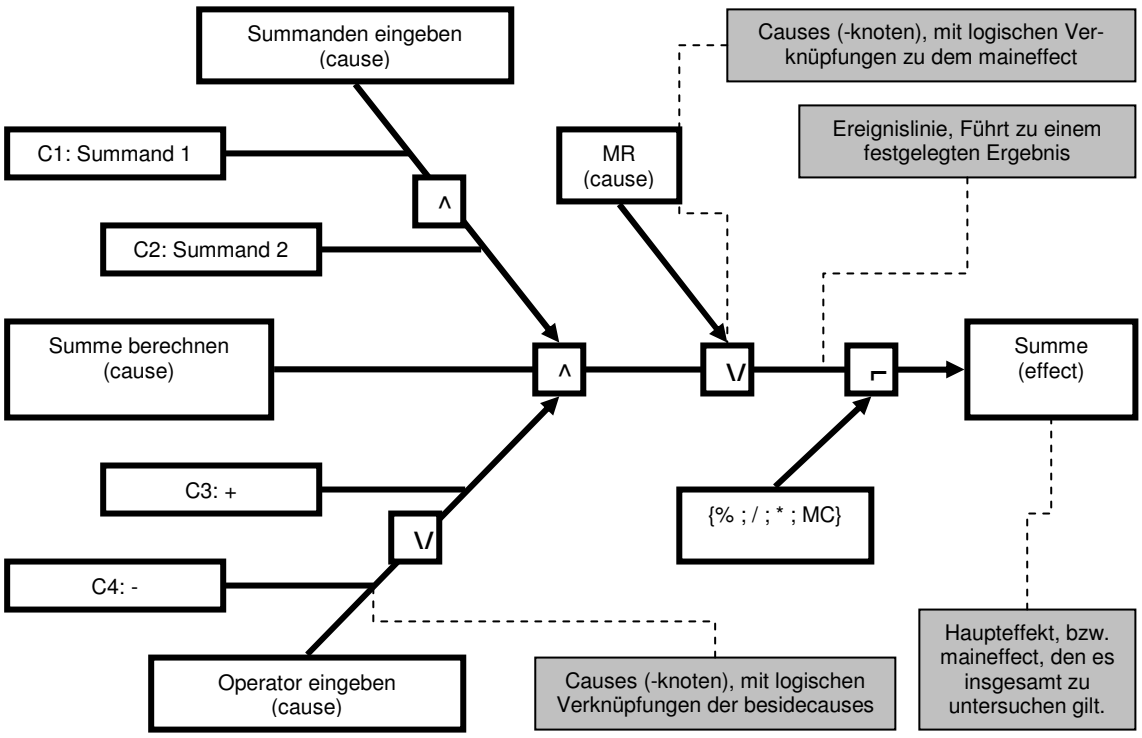
Cause Effect Diagram

→ Was ist wie verknüpft?

In der Praxis auch als Fishbone-, Gräten- oder Ishikawadiagram bekannt.
Damit kann der Grundstein für eine Testlogik im Black Box Test des QM erstellt werden. Als Grundgedanke wird ein (Main-)effect (E) hergenommen, der mit festgelegten causes (C) logisch verknüpft ist.
Jeder cause liefert immer genau ein logisches Ergebnis TRUE (Wahr) oder FALSE (Falsch). Jeder einzelne cause kann wiederum in sich ein eigenständiges cause effect diagram sein.
Die Verknüpfung untereinander liefert wiederum ein logisches Ergebnis.
Wird eine oder mehrere Logische Verknüpfung die zum Erreichen des Haupteffektes nötig sind nicht erfüllt, wird der effect (E) auch nicht erfüllt und liefert somit ein FALSE als logisches Ergebnis.

Die Verknüpfung wird mit Boolescher Logik dargestellt.
Zeichen Bedeutung:
^ UND
V ODER
¬ NICHT
Weitere Symboliken und zusätzlich Boolesche Ausdrücke sind möglich, jedoch im Interesse eines Übersichtlichen Testentwurfs nicht empfehlenswert.

Im Folgenden habe ich ein Beispiel cause effect diagramm erstellt, welches einen Teil der Abläufe des Windows Taschenrechners abbildet. Wie man sieht hat alleine die Standart Oberfläche des WinTR[6] eine Vielzahl von Verzweigungen, die man noch weiter aufspalten könnte. Natürlich ist die Detailtiefe immer an das jeweilige Projekt und den maximal vorhandenen Ressourcen anzupassen.

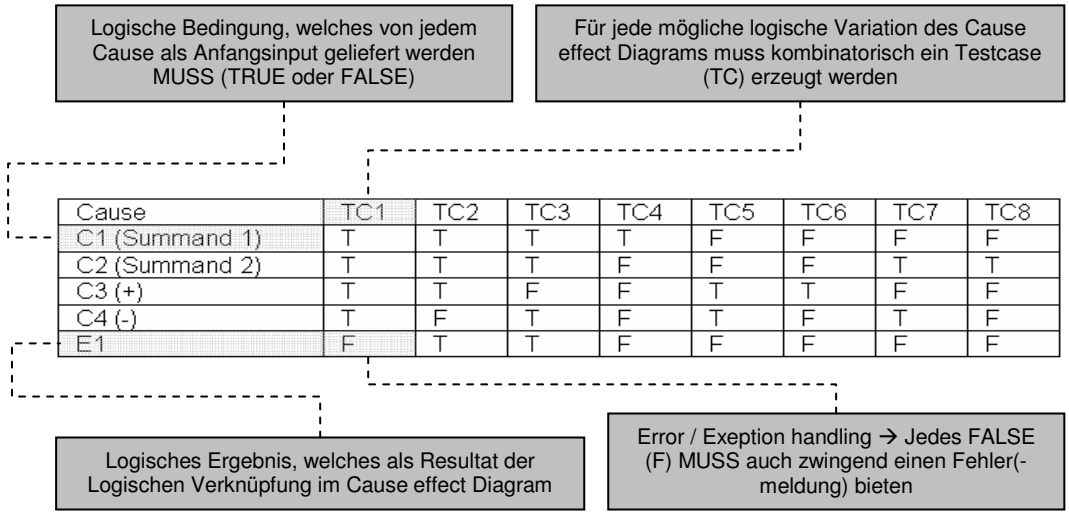


Um nun vernünftig ein Cause Effekt Diagram zu erstellen, bietet es sich an, z.B. bei graphischen Programmoberflächen oder Webseiten, jedes Menu und jeden Button als einen Testcase aufzunehmen und diese dann miteinander zu verknüpfen. Durch diese Methodik entgehen einem auch als Standart angenommene Funktionen wie Öffnen oder Schließen einer Datei oder bestimmte Verzweigungen einer Homepage nicht.
Es muss jedoch immer darauf geachtet werden die Logischen Verknüpfungen sinnvoll aufeinander abzustimmen. Es wird wenig Sinn geben bei einer Homepage den Button „home“ und „kontakt“ mit UND zu verknüpfen. Ein ODER wäre hier allenfalls sinnvoll.

Testlogic

→ Wann ist welches Ergebnis zu erwarten?

Die Testlogik gilt für den Teil der Realität, der mit dem cause effect diagramm modelliert wurde.
D.h. es wird selten bis nie Umgebungseinflüsse wie bestimmte Hardware oder Zusatzsoftware (Typisches Beispiel - der Virenscanner) beachtet.



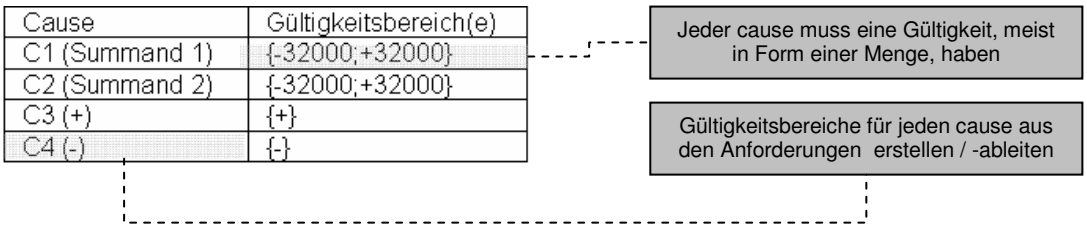
Cause	TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8
C1 (Summand 1)	T	T	T	T	F	F	F	F
C2 (Summand 2)	T	T	T	F	F	F	T	T
C3 (+)	T	T	F	F	T	T	F	F
C4 (-)	T	F	T	F	T	F	T	F
E1	F	T	T	F	F	F	F	F

Ich habe hierbei eine Testlogik auf dem vorherigen Beispiel des WinTR aufgebaut, jedoch nur unter Betrachtung der Grundrechenfunktionen. Aus den einfachen Verknüpfungen des cause effect diagrams kann man nun eine Tabelle erstellen, in der die möglichen Ergebnisse (nicht die zu erwartenden!) eingetragen werden. Dadurch kommen kombinatorisch verschiedene Zustände zusammen, die so genannten Testcases. Und je nachdem wie die logische Verknüpfung im cause effect diagram war, wird bei verschiedenen Eingangssituationen (Ergebnisse oder effects der causes) auch ein definierter Endzustand der (Main-)effect) erreicht. (E1)

Validities

→ Was ist ausschließlich gültig?

Aus den Anforderungen für das jeweilige Programm wird für die Causes des cause effect diagram eine entsprechende Gültigkeit festgelegt. Damit daraus genau ablesbar wird, was zum Beispiel Extremwerte oder falsche Werte sein können.
Es ist im Übrigen völlig unabhängig von den technisch machbaren Werten für z.B. INT Werte, sondern ausschließlich davon abhängig was in den Anforderungen steht, bzw. von einem Kunden gewünscht wurde.

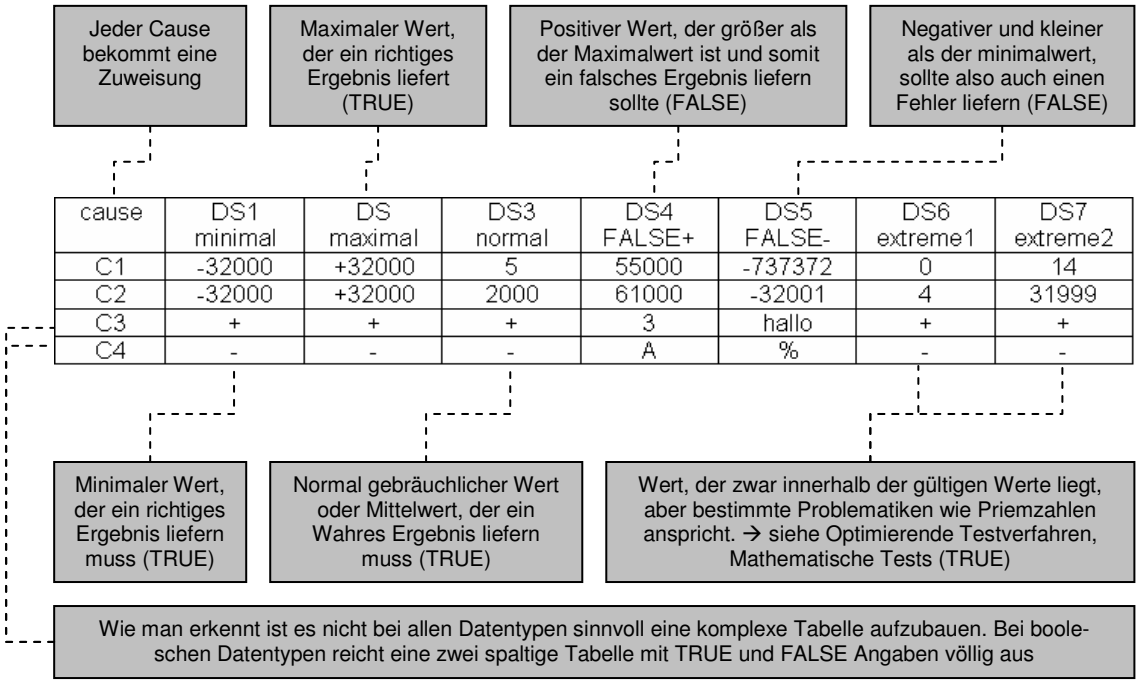


Im Beispiel wieder aufbauend auf dem Vorherigen der Gültigkeitsbereich für den WinTR. Die Werte sind hierbei von mir frei gewählt worden da ja keine Anforderungen oder gar ein Pflichtenheft für den WinTR vorliegen.

Physical Test Case

→ Was kann getestet werden?

Aus den vorangegangenen Tabellen und Diagrammen kann nun eine weitere Tabelle erzeugt werden, in denen Extremwerte, Richtige, Falsche und spezielle z.B. mathematische Werte abgeleitet werden. Hierbei ist zu beachten, dass alle richtigen Werte nur innerhalb der Gültigkeitstabelle auch wirklich Richtig sind.



cause	DS1 minimal	DS maximal	DS3 normal	DS4 FALSE+	DS5 FALSE-	DS6 extreme1	DS7 extreme2
C1	-32000	+32000	5	55000	-737372	0	14
C2	-32000	+32000	2000	61000	-32001	4	31999
C3	+	+	+	3	hallo	+	+
C4	-	-	-	A	%	-	-

Im Beispiel wäre der cause C3 wohl rein technologisch ein 8bit CHAR Datentyp und dieser kann einen beliebigen Wert z.B. zwischen 0 und 255 annehmen, jedoch ist alles ungleich 43 (ASCII Code für +) ein „unrichtiger“ Wert laut Gültigkeitstabelle.
Aus den ganzen Extremwerten entstehen nun gleichzeitig noch Datensätze die für einen Tester jeweils zu einem erwarteten Ergebnis führen müssen.

Testscenario

→ Wer testet was, wann, wie und womit?

Die hergeleiteten Datensätze können nun nacheinander von verschieden Menschen, mit oder ohne Automatismen getestet werden.
Es muss hierbei immer genau ein Mensch als Verantwortlicher dabei sein um den Testablauf zu überwachen. Damit kann gewährleistet werden, das ein Testablauf und damit ein eventuell gefundener Fehler reproduzierbar ist und dann wirklich auf die zu Testende Software zurückzuführen oder eben auszuschließen ist.
Dabei ist es unabhängig, ob es ein einzelner Mensch als Tester oder ein Verantwortlicher für eine ganze Gruppe von Testern ist.
Ein ganz wichtiger Punkt wird hier deutlich, die Reproduzierbarkeit von Fehlern. Denn, wenn ein Fehler nicht reproduzierbar ist kann es durchaus an der Testumgebung selbst oder an massiver Fehlbedienung liegen, das dieser aufgetreten ist.

Nachdem dann der Tester die Datensätze erhalten hat und diese nun durchspielen soll, werden sein Name, sowie Zeit, der getestete Datensatz und die Art wie der Datensatz getestet wurde zusammen mit dem Ergebnis festgehalten.
Es gibt durchaus Sinn die Datensätze noch durch zufällig generierte Werte zu erweitern um damit noch einmal Fehlerquellen abzufangen die bis zur Erstellung der physical test cases gemacht wurden.

Tester	Zeit	DS	Art	SOLL Ergebnis	IST
Hans	01.01.2006 19:00 bis 19:30	1	Manuelle Eingabe des DS gefolgt von Bestätigung mit Enter und Prüfung der Berechnung	Ergebnis Korrekt	Ergebnis Korrekt
Hans	02.01.2006 19:00 bis 19:30	2		Ergebnis Korrekt	Ergebnis Korrekt
Hans	03.01.2006 20:00 bis 20:30	3		Ergebnis Korrekt	Ergebnis Korrekt
Hans	04.01.2006 19:00 bis 19:30	4		Ergebnis Korrekt	Ergebnis Korrekt
Kai	01.01.2006 19:00 bis 19:30	5		Fehlermeldung	Fehlermeldung
Steffen	01.01.2006 19:00 bis 19:30	6		Ergebnis Korrekt	Ergebnis Korrekt
Lutz	01.01.2006 19:00 bis 19:00	2		Ergebnis Korrekt	Fehlermeldung
Max	01.01.2006 15:00 bis 19:30	1		Ergebnis Korrekt	Ergebnis Korrekt
Max	02.01.2006 10:00 bis 11:30	4		Ergebnis Korrekt	Fehlermeldung
Max	03.01.2006 19:00 bis 19:30	6		Ergebnis Korrekt	Ergebnis Korrekt
Moritz	01.01.2006 08:00 bis 11:00	2		Fehlermeldung	Fehlermeldung

Im Beispiel habe ich ein Testszenario für den WinTR aufgebaut, wie es ausschauen könnte. Die Werte sind natürlich erfunden, da der WinTR doch funktionieren sollte.
Es wurden also verschiedene Tester losgeschickt, die nun die testcases fleißig durchprobieren und notieren was dabei geschehen ist. Das Resultat aller Tester kann dann obige Tabelle sein.
Die Spalte mit dem SOLL und IST Ergebnis eines testcase ist dann der eigentlich wichtigste Vergleich, denn anhand dieser sieht man nun ob ein Fehler vorliegt oder die Software ordnungsgemäß funktioniert.
Aber Vorsicht, nicht verwirren lassen, denn: ein testcase, der laut Physical test case einen Fehler verursachen soll, muss auch in der Praxis einen Fehler verursachen. Nur sobald in den Spalten SOLL und IST unterschiedliche Ergebnisse auftauchen ist sehr wahrscheinlich ein Fehler in der Programmierung entdeckt.
Im Beispiel bei den Testern Lutz und Max aufgetreten. (Fiktive Fehler, der WinTR funktioniert auch weiterhin problemlos bei Ihnen!)

Ein ganz entscheidendes Kriterium für Qualität möchte ich an dieser Stelle erwähnen. Eine Software wie der WinTR wird von den Programmierern mit Sicherheit die größten Datentypen verwenden, die rein technisch möglich sind. Das würde aber bedeuten, dass das was die Programmierer gut gemeint haben indem sie alles Programmieren was sie können, im Testszenario KEINEN Fehler verursacht, obwohl er es sollte. Werte über 32000 sollten laut der Gültigkeitstabelle einen Fehler verursachen, tun sie das nicht und funktioniert das Programm trotzdem, handelt es sich um ein unkontrolliertes Feature.
Dadurch entstehen teils gravierende Sicherheitslücken die zu falschen Ergebnissen oder gar zu Systemabstürzen führen können. Denn, ein Feature was nicht durch eine Anforderung bekannt ist und damit auch nicht getestet werden kann, wird demzufolge auch nicht durch Restfehlermanagement gewartet.

Ein schönes Beispiel ist übrigens das „Feature“ des eingebauten Flugsimulators[7] in Excel97… Welcher Tester wird dieses „Feature“ jemals auf seine Anforderungen getestet haben?

Optimierende Testverfahren

Optimierende Testverfahren sind Verfahren, die die bisherigen Schritte zum Softwaretest ergänzen oder verbessern können oder Software etwas unabhängiger von den Anforderungen testen können.
Vor allem können damit teilweise Fehler oder Probleme aufdecken sollen, die vor allem im Nachgang auftreten, also meist während die Software bereits als Endversion im Einsatz ist.
Sie sollen aber auch Fehler aufdecken die durch die normalen Testverfahren aus diversen Gründen nicht erkannt haben. Gründe hierfür können unter anderem zu hohe Kosten oder zu hoher Aufwand sein. Auch besteht die Möglichkeit das Fehler auftauchen die beim Testen aufgrund der Anforderungen an die Software gar nicht bemerkt worden sind.

Funktional coverage

→ Wie viel Funktionalität des Programms wird vom Test abgedeckt?

Da bei der schnellen Entwicklung und den unterschiedlichen Systemzusammensetzungen nicht alle Eventualitäten für einen Softwaretest getestet, geschweige denn überhaupt bedacht werden können, muss man sich im Klaren sein, das nur ein Teil der Software getestet werden kann.
Um nun herauszufinden, wie genau ein Testszenario die möglichen (sinnvollen) Konfigurationen abdeckt, muss man die zu testenden Funktionen beziehungsweise testcases zuerst gewichten und dann die Gewichtung (siehe Grafik) mit dem verfügbaren Budget (in den meisten Fällen Zeitbudget) abwägen.
Sehr unwahrscheinliche Testfälle können oder müssen sogar unter Umständen ganz wegfallen.

Beispiel:
Bei dem WinTR sind der Cause „Summand 1 eingeben“ UND der Cause „Operationszeichen + eingeben“ Funktionen, die extrem oft verwendet werden. Daher sind diese sehr wichtig und bekommen auch eine sehr hohe Gewichtung. Eine Funktion des wissenschaftlichen Bereichs jedoch wird weniger oft verwendet und daher weniger hoch gewichtet.
Wenn es jetzt auch noch einen testcase gäbe, in dem zum Beispiel alle Funktionen gleichzeitig mit der Maus gedrückt werden sollten, so sollte klar sein, dass dieser Fall ganz auszulassen ist. Denn eine so massive Fehlbedienung könnte man kaum noch mit natürlichen (bezahlbaren) Mitteln abfangen. Schließlich sollte erstmal gewährleistet sein, dass eine Software bei normaler Bedienung richtig funktioniert.

Allerdings kann man sich im Vorfeld der Testerstellung nie ganz sicher sein, ob solche Fehlbedienungen durch Usereingabe wirklich von Anfang an auszuschließen sind. Daher müssen auch solche cases prinzipiell erst einmal mit bedacht werden.

Um nun eine Gewichtung durchzuführen, wird das cause effect diagram und dessen testcases hergenommen und dann nach der Häufigkeit gewichtet, in der sie WAHRSCHEINLICH (Erfahrungswerte und Anforderungen als Grundlage hernehmen) auftreten werden. Wenn die „Muss“ - V und „Kann“ – (V) testcases nun im

Black Box Test getestet werden und die anderen im Prinzip fallengelassen werden, bekommt man mit der Gleichung
„Menge von V + (V) *100 / Maximale testcases = %“
die Abdeckung der Testfälle in Prozent heraus.
Durch diese Gewichtung der Testfälle kommen von 100% theoretischer Abdeckung die prozentuale Anzahl der praktisch abgedeckten Testfälle heraus.

Jedoch ist zu beachten, das bei größeren Programmen, oder einem entsprechend hohem Budget kein testcase wirklich wegfällt, sondern die hoch gewichteten im Vergleich zu den weniger gewichteten nur mit mehr, meist zufällig generierten, testcases für den jeweiligen Fall getestet werden.

Testpfade

→ Wo sind Redundanzen?

Die Testpfade dienen zur weiteren Optimierung des Testablaufes durch Vermeiden von Redundanzen im Testverfahren. Es werden alle erstellten Black Box Tests miteinander verglichen und bestimmte Kombinationen und Redundanzen herausgestrichen, die so nicht getestet werden müssen. Dabei geht man bevor man den eigentlichen test beginnt jeden einzelnen Pfad der testcases durch und notiert die Reihenfolge der testcases. Dabei kann man dann die einzelnen Black Box Tests Sortieren und neu verteilen um eine maximale Ausnutzung der Zeit zu ermöglichen und eben Redundanzen zu vermeiden.

Beispiel:
Bei dem WinTR werden die Grundrechenoperationen jeweils in einem cause effect diagram für die Standart Ansicht und für die Wissenschaftliche Ansicht erstellt. Dabei werden nun jedoch vor beziehungsweise während dem Erstellen der Testlogik die Testpfade erstellt, bei denen alle Redundanzen entfernt werden.

Load Test

→ Wie verhält sich das System unter verschiedenen Lasten/Auslastungen?

Verschiedene Belastungen verursachen verschiedene Auswirkungen im System.
Es werden hierbei auch extreme MÖGLICHE Auslastungen getestet. Das heißt ein Wert der praktisch gar nicht zu testen ist, ist KEIN realistisch möglicher Test.
Es müssen aber trotzdem auch realistische Tests beachtet werden, die von den Anforderungen nicht gefordert werden.

Beispiel:
Der WinTR soll Daten zwischen -32000 und +32000 verarbeiten können. Allerdings kann man davon ausgehen, dass rein technisch auch Werte bis 32 Bit funktionieren sollten oder könnten. Aber auch 64Bit oder gar 128Bit sind heutzutage keine besondere Hürde mehr und könnten funktionieren. Also wäre der maximal mögliche Wert jeweils 32Bit, 64Bit und 128Bit. Also gleich 3 realistisch mögliche Lastgrenzen bis zu denen getestet werden kann.

Laufzeit Test / Stabilität

→ Wie lange „rechnet“ ein Teilprogramm?

Hardware, Betriebssystem, Datenaufkommen und parallel laufende Software wird mit dem Programm getestet und dabei wird gestoppt wann das Programm einen Fehler oder gar einen Absturz verursacht.
Dabei sind als maximale Laufzeit 2 Wochen oder mehr keine Seltenheit.

Mit solchen Tests lassen sich im Übrigen nicht nur Aussagen über die zu testende Software machen, sondern auch über die Hardware auf der es getestet wurde. Die Bezeichnung hierfür lautet mean time between failure (MTBF[8]) – Durchschnittliche Zeit zwischen Fehlern.

Real-world-behavior

→ Imitieren von realistischem Benutzerverhalten

Im Prinzip die eigentliche Bezeichnung für Beta Tests, bei denen die Software in den Umgebungen und von den Personen getestet werden soll, die diese dann später auch in der Endversion nutzen sollen. Diese Tests sollen vorwiegend alle die Fehler finden die in keinem Testpfad abgesichert wurden oder bei Firmen, die sich nicht die Mühe machen wollen wirklich über Testabläufe nachzudenken.

Mathematische Tests

→Wie rechnet ein z.B. mathematisch orientiertes Programm mit speziellen Zahlen und Extremen?

Auswahl einiger Methoden:
- ***Zahlen um 0*** - für Division durch Null Problematik und zu kleine Register.
- ***Primzahlen*** - da manche Compiler intern zur Optimierung von Code diverse Umrechnungen zwischen Zahlenformaten wie dual, octal, dezimal und hexadezimal durchführen. Dabei gehen aber manche Primzahlen verloren oder werden dann „weggerundet“. Treten bei solchen Werten Fehler auf, kann man bei solchen Fehlern ungünstige oder gar falsche Compileroptionen vermuten.
- ***Extremzahlen*** z.B.: Wurzel aus -2, also Werte, die direkt Fehler verursachen MÜSSEN, müssen also auch abgefangen werden.
- ***Savagetest*** z.B. Wurzel aus 4 = 2^2 = 4 Wenn ein solcher Schritt mehrmals gerechnet wird, (auch mit anderen Testwerten wie 1.23354212123 oder ähnliche) können durch Fortpflanzung von Rundungsfehlern irgendwann andere Werte als der Ursprungswert rauskommen.
- ***Interne Stellen*** sichtbar machen: z.B.: (3.1415 * - 3.1415) * 10000 =???
Es werden durch solche Rechnungen die letzen Stellen angezeigt und dadurch Fehler bei Rundungen genauer eingekreist.

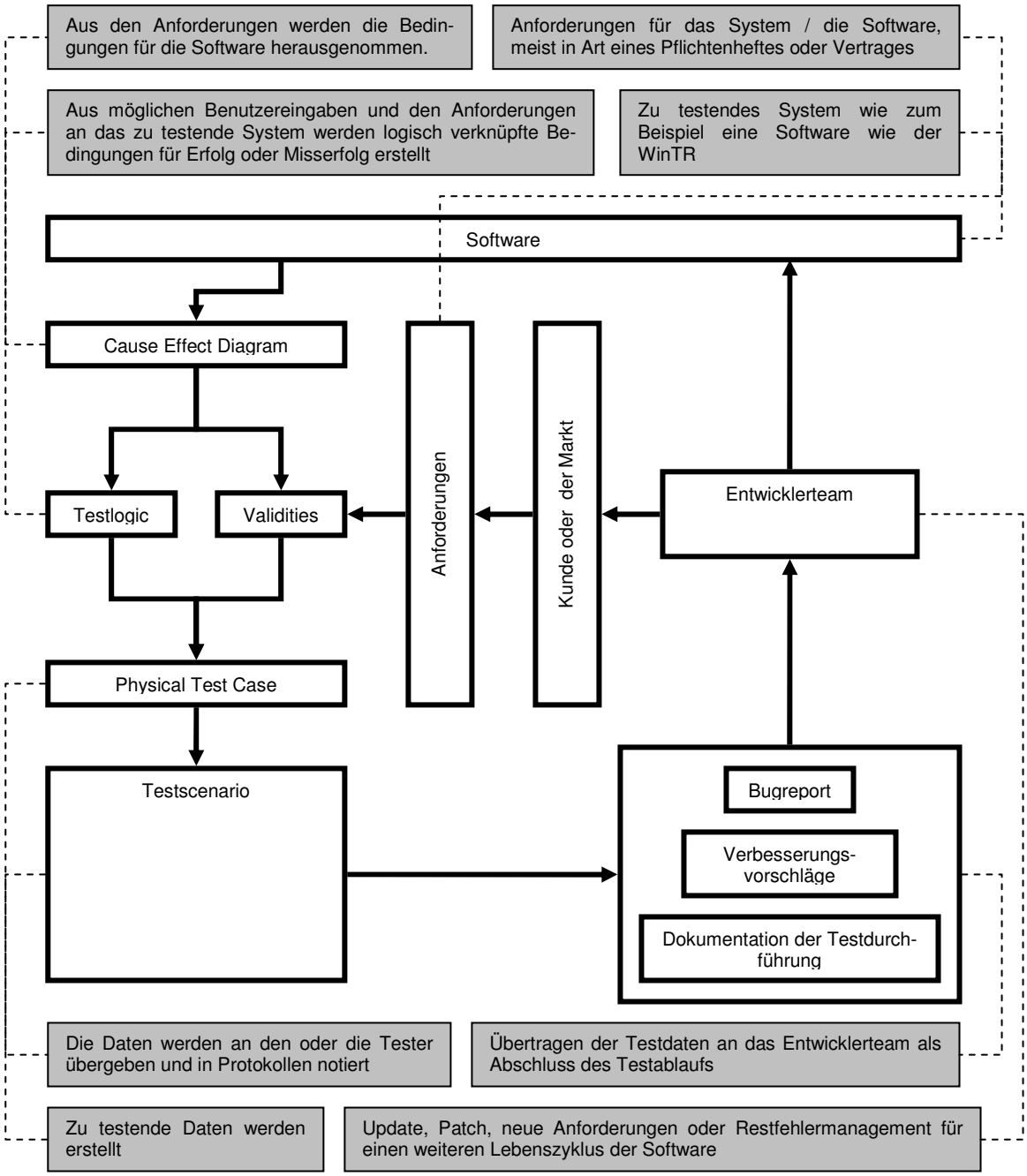
Fazit

Um eine Software (und evtl. auch Hardware) zu testen gibt es eine Menge zu beachten. Man sollte dabei vor allem kleine Details nicht außer Acht lassen. Jedoch können mit schon relativ kleinem Aufwand bereits sehr schnell und sehr viele Aussagen über die qualitative Umsetzung und mögliche Fehler getroffen werden. Leider zeigen vor allem Große Firmen, das sie Außerstande sind über solch relativ kleine Verfahren nachzudenken und beweisen dies immer wieder durch stark fehlerbehaftete Software.

Zu meiner Arbeit selbst möchte ich noch hinzufügen, das ich sicherlich auch hätte mehr oder weniger schreiben können um auf eine genaue Seitenzahl von genau 10 Seiten zu kommen, jedoch habe ich versucht den Schwerpunkt auf einen anwendbaren Nutzen der Arbeit zu halten. Daher auch die vielen Selbsterstellten Grafiken, anhand derer ich mehr und gezielter ausdrücken konnte was wichtig ist als wenn ich es „nur“ umschrieben hätte.

Anhang A – weitere Materialien

Ablauf Black Box Test



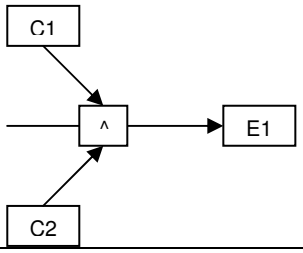
Funktional Coverage

Gewichtung der Testcases:
V
Auf jeden Fall! – Muss getestet werden!

(V)
Evtl. sollte noch getestet werden, wenn nach dem Budgetplan aller MUSS Notwendigkeiten noch Mittel vorhanden sind

(X)
Sollte nicht mehr getestet werden, da in dem Fall eine sehr wahrscheinliche Fehlbedienung vorliegt, die in der Regel nicht mehr durch normale Bedienung auftreten kann

X
Fällt weg, da doppelter Testfall oder technisch nicht mehr möglich



	TC1 V	TC2 (V)	TC3 (V)	TC4 (X)
C1	T	T	F	F
C2	T	F	T	F
E1	T	F	F	F

Anhang B - Quellen

[1] Chris Rupp – Sophist Group, Requirementengeneering und Management, Hanser Verlag München Wien, 2004
[2] http://www.Wikipedia.de , IV/2005 – I/2006
[3] Bob leGrand, Q-Course, unbekannt 2003
[4] H.J. Günther, SEN I-V, 2005
[5] Tom deMarco, Spielräume – Jenseits von Burn-out, Stress und Effizienzwahn, Hanser Verlag München Wien, 2001
[6] http://www.mogelpower.de/easter/eggs/index.php, I/2006

Anhang C - Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als angegeben verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort: Reichenbach
Datum: 28.02.2006

Unterschrift

Anhang D - Glossar

[1] QM – Qualitätsmanagement

[2] DAU - Dümmster anzunehmender User. Kann zum Beispiel jemand sein, der noch nie an einem Computer Gesessen hat und durch Zufall oder Absicht Bedienungen durchführt, die so niemals von jemand gemacht werden würden, der sich mit der Bedienung eines PCs auskennt.

[3] White Box Test – Typischer Testverlauf während eines Entwicklungsprozesses, bei dem das gesamte zu testende Know How vorliegen sollte. Beispiele für WBT sind Debugging, Reverse Engeneering oder Code Review.

[4] Final Release – Letzte Version einer Software im Entwicklungsprozess, von der normalerweise gesagt wird, dass sie Fehlerfrei ist und alle Features enthält.

[5] Betaversion – Erste Versi0on einer Software im Entwicklungsprozess, die an eine breite Masse gegeben wird und alle Features mindestens in Ansätzen enthält.

[6] WinTR - Musterbeispiel in Form des Microsoft Windows System Taschenrechners, auch als calc.exe bekannt.

[7] Flugsimulator in Excel 97
Öffnen Sie ein neues Rechenblatt. Drücken Sie [F5] für "Gehe zu", und springen Sie zu X97:L97. Drücken Sie [Tab], um zu M97 zu wechseln. Halten Sie [Strg] + [Umschalt], und klicken Sie auf den Diagramm-Assistenten (das blau-gelb-rote Diagramm-Symbol). Sie finden sich im Cockpit eines Flugsimulators wieder; mit der Maus steuern Sie über die Fraktallandschaft. Mit ein bisschen Geschick finden Sie eine Laufschrift mit den Namen der Entwickler des Programms.

[8] MTBF - Durchschnittliche Laufzeit eines Systems, bis es einen Fehler verursacht