Eine zum ISTQB Standard konforme

Kurzanleitung für Tester & Test Manager

Gebhard Greiter, 2009

Schon seit etwa 1975 hat man versucht, methodisches Wissen über die Kunst, Software zu testen, zusammenzutragen. Dies zu tun hat sich lange Zeit als schwierig erwiesen, war aber schließlich doch von Erfolg gekrönt: Seit 2003 existiert der ISTQB Standard – ein aus zwei Büchern bestehendes Werk, in dem alles zusammengetragen ist, was heute für Tester und Testmanager interessant sein könnte.

- Buch 1 (***Foundation Level***) definiert Terminologie und betont Grundwahrheiten, die jedem Tester geläufig sein sollten.
- Buch 2 (***Advanced Level***) wendet sich vor allem an Testmanager, beschreibt aber neben einigen praktischen leider auch sehr viele in den meisten Projekten eher *nicht* anwendbare Methoden.

Ganz grundsätzlich hat der Standard den Nachteil, nicht zu unterscheiden zwischen Methoden, die man kennen muss, und solchen, denen eher nur theoretische Bedeutung zukommt (da sie für den Großteil aller Testprojekte viel zu kompliziert wären). Bücher, die den Standard verdaubar machen wollen, [ISTQB.Spillner] etwa, umfassen viele hundert Seiten – versprechen also keine *schnelle* Hilfe dem, der eben dabei ist, ein Testprojekt zu starten.

Dies erkannt, macht es Sinn, eine Art Kochrezept zu schreiben, das man auch ohne langes Studium der ISTQB Bücher mit Vorteil anzuwenden in der Lage ist: eben dieses Papier hier.

Als Kochbuch besteht es aus

- einigen wichten Rezepten (die oft schon ausreichen)
- ergänzt um Anhänge, die man einzeln hinzunehmen oder auch ignorieren kann.

Je länger das Kochbuch in Gebrauch ist, desto größer könnte die Zahl seiner Anhänge sein: Man muss sie nicht alle schon am ersten Tag haben.

Wir beginnen – wie ISTQB auch – mit der Definition einer Begriffswelt und mit Klarstellungen, die genau zu verstehen von zentraler Bedeutung ist:

1 Wichtige Begriffe & Klarstellungen (nach ISTQB)

Eine ***Fehlhandlung*** einer Person führt zu einem ***Fehlerzustand*** (zu fehlerhaftem Code, zu falscher Verkabelung, oder ähnlichem).

Der hierdurch gegebenen ***Fehler*** wird dem Anwender des Systems (auch dem Tester) aber nur sichtbar über unerwartete Wirkung (sog. ***Fehlerwirkung***).

Es kann durchaus sein, dass eine konkret beobachtete Fehlerwirkung auf mehr als nur einen Fehler zurückzuführen ist. Auch kann sich umgekehrt ein und derselbe Fehler so auswirken, dass man unerwartete Wirkungen beobachtet, denen nicht mehr anzusehen ist, dass sie alle ein und dieselbe Ursache haben.

Fehlerfreiheit ist durch Test ***nicht*** beweisbar.

Test kann stets nur Fehlerwirkung entdecken (sog. ***Bugs***).

Unter ***Debugging*** versteht man die Analyse eines Bugs mit dem Ziel, die Fehlerursache, den Fehlerzustand also, zu lokalisieren und zu beseitigen.

Bitte beachten: Issues im Bug Tracking System beschreiben zunächst nur Fehlerwirkung – erst später kann (und wird man, wenn wichtig) im Ticket auch Analyseergebnisse festhalten.

Wir sehen: Was unser Kunde als „Fehler“ bezeichnet, ist nach ISTQB eine „Fehlerwirkung“. Das ist kein Wunder, denn der Kunde nimmt das System nur wahr über seine Wirkung, kennt aber i.A. keinerlei Implementierungsdetails.

Testen lohnt sich, denn:

Nach einer als repräsentativ geltendenen Untersuchung der Carnegie Mellon University aus 2003 enthält Software kurz nachdem sie produktiv geht i.d.R. immer noch

1 bis 10 Fehlerzustände pro 1000 Lines of Code

Der jeweils beobachte konkrete Wert korellierte mit dem Reifegrad der vorgefundenen Entwicklungs- und Testprozesse (***ungeregelt***, ***geregelt*** bzw. ***geregelt und gemessen***):

http://www.greiterweb.de/spw/prozessreife.htm

http://www.sei.cmu.edu/library/abstracts/news-at-sei/wattsnew20041.cfm

2 Testaktivitäten

Nach ISTQB umfasst der Testprozess folgende Aktivitäten:

- Testplanung und -steuerung
- Testentwurf, Realisierung oder Update geeigneter Testtreiber
- Testdurchführung
- Auswertung der Ergebnisse (und je ein Bericht pro Testdurchführung)
- Abschlussarbeiten (Erfahrungen & Testware konservieren)

Wie praktische Erfahrungen zeigen, können diese Tätigkeiten auf Personen verteilt sein, die sich auf je eine der folgenden Rollen spezialisiert haben:

- Testmanager
- Testexperte (fachlich)
- Testexperte (technisch)

Testentwurf und Analyse applikationslogischer Fehlerwirkungen sollte Aufgabe der Testexperten (***fachlich***) sein.

Testautomatisierung, Testdurchführung und das Erstellen von Werkzeugen für automatische Auswertung der Testlauf-Ergebnisse sollte Aufgabe der Testexperten (***technisch***) sein.

Diese Empfehlung begründet sich über folgende zwei Beobachtungen:

- **Wo nur fachliche Testexperten am Werk sind**, wird viel zu wenig Testautomatisierung erfolgen (mit viel Aufwand wird dann wenig erreicht, denn: Manuelle Testdurchführung ist typischerweise um einen Faktor 10 bis 100 aufwendiger als voll automatisierte).

- **Wo nur technisch orientierte Testexperten am Werk sind** werden deutlich zu wenig Fehler fachlicher Art entdeckt (da solche Personen eigentlich nur prüfen können, ob die Anwendung nirgendwo abstürzt. Dies gilt ganz besonders dann, wenn die Anwendung fachlich kaum dokumentiert ist – ein gar nicht so seltener Fall).

2.1 Planen des Testprozesses

Test zu planen bedeutet i.W.

- Definition der Test- und Qualitätsziele (relativ zu entstehenden Kosten)
- Definition und Bereitstellung der Testumgebung

2.1.1 Definition angestrebter Test- und Qualitätsziele

Hierzu empfiehlt sich Rückgriff auf ein wohldefiniertes Qualitätsmodell, z.B. auf das der ISO-Norm 9126. Sie kennt 21 Qualitätsmerkmale bzw. 6 Gruppen solcher Merkmale.
Sie zugrundegelegt, lässt sich zur Definition der Testziele folgendes Formular verwenden (am besten in Form eines EXCEL Sheets, welches dann weitere Spalten haben kann, wie etwa *Kommentar*, *Bearbeiter*, *Prio*, *Aufwand in BT*, usw.):

Testaufwand in Prozent	Testaufwand in Prozent	**für Qualitätsmerkmal nach ISO 9126**
3		**Effizienz**
	1	▪ Zeitverhalten
	1	▪ Verbrauchsverhalten
5		**Benutzbarkeit**
	1	▪ Verständlichkeit
	1	▪ Erlernbarkeit
	2	▪ Bedienbarkeit
61		**Funktionalität**
	4	▪ Angemessenheit
	55	▪ Richtigkeit
	0	▪ Interoperabilität
	1	▪ Ordnungsmäßigkeit
	4	▪ Sicherheit
16		**Zuverlässigkeit**
	1	▪ Reife
	5	▪ Fehlertoleranz
	5	▪ Stabilität
	5	▪ Wiederherstellbarkeit
9		**Änderbarkeit**
	3	▪ Analysierbarkeit
	3	▪ Modifizierbarkeit
	3	▪ Prüfbarkeit
4		**Übertragbarkeit**
	2	▪ Anpassbarkeit
	1	▪ Installierbarkeit
	1	▪ Konformität
	0	▪ Austauschbarkeit
100	100	Summe

Wichtig: Qualitätsmerkmale, auf die man keinen Wert legt, haben Prozentanteil 0, stehen aber dennoch in der Liste: Es soll ja klar werden, dass nichts übersehen wurde.

2.1.2 Definition eines Rasters, den Testfortschritt zu monitoren

Mit zur Definition der Testziele gehört im Prinzip auch eine Aussage über die angestrebte Mindest-Testabdeckung.

Da es i.A. zu schwierig ist, dieses Ziel hinreichend praxisnah – ausreichend realistisch also – festzusetzen, geht man besser vor wie folgt:

Man definiert ein ***Raster zur Dokumentation von Testabdeckung***. Es muss geeignet sein sicherzustellen, dass die Anwendung überall in gleicher Tiefe getestet wird. Dieses Raster ergibt sich durch ***Definition sog. Testobjekte*** – man versteht darunter sämtliche im Zuge des Testens zu prüfenden Funktionen und Eigenschaften der Applikation. Genauer: all jene, für die es mindestens einen Testcase geben sollte.

Mindestens als je ein Testobjekt definiert sein muss jede Funktion der Anwendung, die

- Endanwender,
- Administratoren oder
- Nachbarsysteme

einzeln aufzurufen in der Lage sind.

Die Liste der Testobjekte sollte man – geordnet nach Priorität und beabsichtigter Intensität des Testens – als Tabelle folgender Form halten:

Testobjekt	TO_Testsuite ID	Zahl im letzten Testlauf erkannter Ok/notOk

Unter einer ***TO_Testsuite*** im Sinne dieser Tabelle versteht man ein automatisch erstellbares Skript, welches sämtliche Testcases zum Ablauf bringt (oder wenigstens auflistet), die auf das genannte Testobjekt zugreifen.

Es sollte passendes Hilfswerkzeug geben, den Inhalt der zweiten und dritten Spalte dieser Tabelle nach jeder Testdurchführung automatisch aktualisieren zu können. Nur so wird es dem Testmanager leicht fallen,

- den Testfortschritt stets genau beurteilen zu können,
- zu wissen, wo sich derzeit Fehler häufen, und
- die notwendigen Schwerpunkte zum Ausbau der Testsuite neu zu setzen.

Theoretiker fordern auch die Definition eines Kriteriums „***Genug getestet ist, wenn ...***“ (ein Testabbruchkriterium also). Praktiker halten davon gar nichts, da man sich hier nur unnötig festlegt. Bereits erreichte Testabdeckung jederzeit zu kennen – d.h. zu wissen, wie viele Prüfungen es zu jedem Testobjekt gibt und wo sich Fehler häufen – hilft da weit mehr.

2.1.3 Definition und Bereitstellung der Testumgebung

Klar sein muss: Wo keine isolierte Testumgebung existiert (eine, in der auch ein hinreichend nicht-trivialer Datenbestand aufgebaut werden kann), kann dem Tester nicht gestattet sein, Funktionen aufrufen, die Daten fortschreiben.

Konsequenz daraus: Das Testteam benötigt

- Eine nur ihm selbst zugängliche Installation der zu testenden Anwendung.
- Sie muss in hinreichend kurzer Zeit komplett neu aufbaubar sein,
- und dies gilt auch – und insbesondere – für einen typischen Datenbestand, für den ein jederzeit neu herstellbarer, wohldefinierter Ausgangszustand spezifiziert ist (nur so ist problemlos Regressionstest möglich).

Es muss explizit festgelegt werden, wie viele Versionen der Anwendung zeitlich parallel testbar sein sollen und welche Anforderungen sich hieraus für die notwendige Verfügbarkeit der Testumgebung ergeben.

2.2 Steuern des Testprozesses

Steuern des Testgeschehens besteht darin,

- Ressourcen einzuplanen
- Termine zu setzen

und dann

- ausgehend von verfügbaren (bzw. eingeplanten) Ressourcen
- hinreichend oft
- erreichte Testabdeckung zu betrachten
- um so – wo sie zu ungleichmäßig oder zu gering ist, oder wenn sich terminliche Engpässe abzeichnen – korrigierend einzugreifen

durch Abänderung bislang in der Testplanung stehender Termine, Prioritäten oder Ziele.

2.3 Testentwurf und Regeln für Testtreiber-Realisierung

Testentwurf dient dem Aufbau (und der Pflege) einer hinreichend umfangreichen Testsuite. Sie besteht aus einzeln aufrufbaren ***Testtreibern*** sowie aus Prozeduren zur automatischen Prüfung durch die Anwendung gezeigter oder neu produzierter Daten (nennen wir sie ***Checktreiber***).

WICHTIG zu wissen:

Die Hersteller gängiger Testwerkzeuge gehen sämtlich davon aus, dass die Testtreiber nicht nur dem Triggern der Anwendung dienen, sondern gleichzeitig auch dem Prüfen der Anwendungsreaktion auf Korrektheit. Dies hat gravierende Nachteile:

- **Nachteil 1:** Es sind stets nur relativ triviale Prüfungen möglich.

- **Nachteil 2:** Was tatsächlich geprüft wird, ist schwer zu durchschauen, da der die Prüfungen implementierende Code vermischt ist mit dem Code, der notwendig ist die Anwendung zu triggern. Verständlichkeit und einfache Wartbarkeit der Testtreiber leiden stark darunter.
- **Nachteil 3:** Entsteht die Notwendigkeit, Prüfungen abzuändern oder auszubauen, muss stets der gesamte Testlauf wiederholt werden (es reicht also nicht, nur die Prüfungen neu aufzurufen). Dies kann zu kritischen Verzögerungen im Testgeschehen führen, bewirkt aber auf jeden Fall einen ganz unnötigen Verlust an Flexibilität.

Konsequenz also:

Jeder Testtreiber sollte so gestaltet werden, dass

- Zunächst alles Triggern der Anwendung passiert,
- die Reaktionen der Anwendung parallel dazu protokolliert werden,
- und die Prüfung der Anwendungsreaktionen auf Korrektheit allein durch Prüfen dieses Protokolls (also auch nachträglich noch) stattfinden kann.

Vorteile dieses Vorgehens sind:

- Man vermeidet alle oben genannten Nachteile.
- Der Umfang notwendigen Prüfcodes sinkt teilweise dramatisch.
- Testtreiber brauchen Prüfcode nur noch an wenigen Stellen zu enthalten (dort nämlich, wo sie abbrechen sollen, wenn weiter zu rechnen keinen Sinn machen würde).
- Testtreiber und Checktreiber werden einfacher pflegbar (und teilweise sogar unabhängig voneinander pflegbar).
- Das Testteam kann auf neue Prüfanforderungen deutlich schneller reagieren.
- Erreichte Testabdeckung transparent zu machen wird einfacher.
- Testabdeckung gezielt auszubauen wird ebenfalls einfacher.
- Beim Aufruf einer Testsuite unbeabsichtigt den einen oder anderen Testtreiber gar nicht aufzurufen (ebenso wie Absturz einiger Testtreiber), wird automatisch mit als Fehler erkannt.
- Datawarehouse-Anwendungen gestatten nur so Prüfungen, die nicht allzu trivial sind.

Hinweis: Wo Testautomatisierung unterbleibt, ist der Testtreiber nur ein Skript, aus dem hervorgeht, wie und in welcher Reihenfolge der Tester die Anwendung zu triggern hat und welche Reaktion er auf welche Eingaben hin nach Art und Inhalt erwartet. Protokollierung der Anwendungsreaktionen und Prüfen erst im Nachhinein ist hier also ***nicht*** möglich.

Nicht automatisierte Testtreiber sollten nur ausnahmsweise existieren (oder nur über einen begrenzten Zeitraum hinweg).

Mit Abstand bestes Werkzeug für das Automatisieren von Test, der mit der Applikation nur über eine GUI zu sprechen hat, ist ***HP QuickTest Professional***. Es optimal einzusetzen erlernt man aber nicht unbedingt per Schulung durch den Hersteller.

Nicht automatisierte Testtreiber sollten – damit wenigstens Testabdeckung automatisiert gemessen werden kann – Tabellen sein in z.B. folgendem Format (sie können beliebig umfangreich sein):

Testcase:	Testcase Name	Ok / NotOk	Kommentar
TO	Name angesprochener Testobjekte		
E	Eingaben des Testers		
R	Erwartete Systemreaktion(en)	Ok	
R	Erwartete Systemreaktion(en)	NotOk	
E	Eingaben des Testers		
R	Erwartete Systemreaktion(en)	NotOk	
	...		
			stop

2.4 Testdurchführung

Hierunter versteht man

- den Aufruf einer nach Umfang genau definierten Testsuite
- angewandt auf eine exakt definierte Version der Anwendung.

Die Testsuite aufzurufen bedeutet:

- Manuelles Durchspielen aller darin enthaltenen nicht automatisierten Testtreiber.
- Und parallel dazu den Aufruf eines Skripts, welches sämtliche in der Suite enthaltenen automatisierten Testschritte in bestimmter Reihenfolge durchführt (erst die Testtreiber, danach einen auf eben diese Suite zugeschnittenen Checktreiber).

Resultat solcher Testdurchführung ist ein Verzeichnis, das alle hierbei durch die Treiber geschriebenen Dateien beinhaltet: das sog. ***Testergebnis***.

Was nicht automatisierte Testtreiber betrifft, so muss im Testergebnis eine Kopie davon abgelegt sein mit zutreffend ausgefüllter Spalte Ok/NotOk.

Auch eine Kopie der hinsichtlich Spalte 3 aktualisierten Testobjektliste sollte sich später dort finden. Sie wird erstellt im Rahmen der sog. Testauswertung:

2.5 Testauswertung

2.5.1 Auswertung des Ergebnisses einer einzelnen Testdurchführung

Das Testergebnis auszuwerten in dem Sinne, dass das Ergebnis einfach verstehbare Form annimmt (als Hauptkapitel des zu erstellenden Testberichts) sollte es einen automatisierten ***Auswertungstreiber*** geben.

Eine seiner Hauptaufgaben ist eine klare Beschreibung erreichter Testabdeckung auf Basis des im Zuge der Testplanung festgelegten Rasters sog. Testobjekte (siehe 2.1.2).

Das Testergebnis – ergänzt um den Testbericht – sollte per Configuration Management als Teil der getesteten Systemversion archiviert werden.

2.5.2 Auswertung von Testergebnissen zwecks Trend-Analyse

In jedem Entwicklungsprojekt wird es nicht einen sondern nacheinander sehr viele Test-durchführungen geben. Es macht daher Sinn, sich zu überlegen, wie man durch Vergleich der durch sie produzierten Testergebnisse einen Trend hin zum Besseren oder Schlechteren feststellen kann.

Solche Trend-Analyse sollte sich beziehen auf

- beobachtete Qualität der Anwendung einerseits und
- Testaufwand andererseits.

2.6 Abschlussarbeiten

Wie jedes Projekt, so sollte man auch jedes Testprojekt damit abschließen, die jeweils gemachten positiven und negativen Erfahrungen schriftlich zu fixieren, zu analysieren und für kommende Projekte wiederverwendbar zu publizieren.

Im Projekt entstande oder verbesserte Templates, Checklisten und Hilfswerkzeuge (sog. Projectware) sollten konserviert werden, so dass kommende Projekte nicht Ähnliches neu entwickeln müssen.

3 Bewertung und Verbesserung der Projektprozesse

Die Arbeiten für Entwicklung, Test und hierfür notwendigem Management sind je nach Unternehmen unterschiedlich genau geregelt. Unter der ***Prozessreife*** eines Unternehmens versteht man die Güte solcher Regelung.

Das am weitesten verbreitete Modell zur Charakterisierung von Prozessreife ist das an der Carnegie Mellon University entwickelte CMMI. Es unterscheidet 5 Reifegrade.

Einfacher – aber ebenso nützlich – erscheint das nun folgende nur 3-stufige Modell. Nennen wir es ***das Grundmodell für Prozessreife***. Es sagt:

- Reifegrad 1 (ungeregelt) liegt vor, wenn man *nicht* nach einem schriftlich fixierten Prozess arbeitet.
- Reifegrad 2 (geregelt) liegt vor, wenn alle im Team nach einem projektweit schriftlich fixierten Prozess arbeiten.
- Reifegrad 3 (geregelt und gemessen) liegt vor, wenn zusätzlich noch Erfolg und Misserfolg anhand ständig erhobener Kennzahlen quantifiziert werden. Wie diese Zahlen – sog. ***Key Performance Indicators*** – errechnet werden, muss schriftlich fixiert sein.

Die in http://www.sei.cmu.edu/library/abstracts/news-at-sei/wattsnew20041.cfm diskutierte Untersuchung deutet darauf hin, dass die bei Auslieferung von Software an den Kunden vorhandene Fehlerdichte um etwa den Faktor 10 geringer ist, wenn man im Projekt nicht ungeregelt sondern stattdessen geregelt und gemessen gearbeitet hat.

Jedes Unternehmen sollte daher bestrebt sein, projektübergreifend zu lernen, um so schrittweise von ungeregelter zu geregelter und gemessener Projektarbeit zu kommen.

Der ISTQB Standard berücksichtigt das über seine Definition der (Test-) Abschlussarbeiten.

Inhaltsverzeichnis

Quellenverzeichnis

[ISTQB.Spillner]: Praxiswissen Softwaretest

Ein Werk in zwei Bänden (zusammen etwa 700 Seiten):

- Aus- und Weiterbildung zum Certified Tester - Foundation Level, nach ISTQB Standard, 296 Seiten, 2008, ISBN 978-3-89864-358-0
- Aus- und Weiterbildung zum Certified Tester - Advanced Level, nach ISTQB Standard, 419 Seiten, 2008, ISBN 978-3-89864-557-7