Automatisiertes Software Testing

Katharina Ziegler
Matrikel Nummer 791521
Korbinianstraße 18
82515 Wolfratshausen
Tel 017623494763
lichttechnik.ziegler@web.de

Beuth Hochschule für Technik Berlin
Wissenschaftliche Projektarbeit
Studiengang
Medieninformatik Bachelor online
WS 2016/17

Abstract

Fehlerfreie Software gibt es nicht. Jedoch kann das korrekte Verhalten der Software in verschiedenen Anwendungsfällen getestet werden. Hierfür muss das korrekte Verhalten im Requirment-Management definiert werden. Setzt sich dann jemand aus dem Entwicklerteam hin und probiert die Anwendungsfälle aus? Natürlich nicht. Über die ISTQB-Qualifizierung ist das Berufsbild des Software Testers entstanden. Verschiedene Spezialisierungen wie „Test Analyst“ oder „Test Manager“ decken die verschiedenen Aufgabenbereiche ab.
Nicht für alle Testarten ist eine Automatisierung sinnvoll und effizient. Jedoch kann sie, an der richtigen Stelle, unter Verwendung eines geeigneten Werkzeugs, eine Menge Geld und Zeit sparen.

General Terms

Automatation, Testing, Software Lifecycle

Keywords

Software Testing, Testcase, Testmanagement, Tools, Design, Architecture, Automation

Introduction

Warum Software Testen?
Software Entwickler kennen alle die Horror Stories von gescheiterten Projekten, die durch banale Software Fehler verursacht wurden. Zum Beispiel [F01] die Explosion der unbemannten Ariane Rakete der European Space Agency, welche am 4. Juni 1996 nur vierzig Sekunden nach ihrem Start in Kourou, Französich-Guayana explodierte. Die Rakete war auf ihrem ersten Flug und kostete inklusive ihrer Fracht 500 Millionen Dollar. Verursacht wurde der Schaden durch eine ungültige Datentypkonvertierung. Eine 64 bit floating point number sollte in einen 16 bit signed integer konvertiert werden.

Oder vom Großprojekt Hartz IV [F02] , bei dem die Arbeitslosegeld-II-Empfänger den Jahresbeginn 2005 erst einmal ohne Geld verbringen mussten. Das Arbeitslosengeld hatte seine Empfänger nicht erreicht, weil die Kontonummern, die kleiner als 10-stellig waren rechtsbündig und nicht linksbündig aufgefüllt wurden. Alleine rund 200 000 Kontoinhaber bei der Postbank hatten ihr Geld nicht bekommen.

Getrieben durch die stetig wachsende Komplexität [SWT10] , den Integrationsgrad und insbesondere die hohe Abhängigkeit (Geschäftswert oder sogar Menschenleben) von Softwaresystemen, wird der Software Test strategisch immer bedeutender.
Dennoch ist das Testen der entwickelten Software mehr eine lästige Pflicht als eine interessante Aufgabe und vielerlei wird das Testen in der Planung eines Projekts mit wenigen Ressourcen bedacht oder auch zu spät hinzugenommen.

Gerade aus der Vielzahl der Beispiele für folgenschwere Softwarefehler [S-T02] , müsste die Folgerung nahe liegen, dass dem Testen eine wichtige Stellung eingeräumt werden müsste.
Dies ist nicht in dem Ausmaß der Fall, wie wir das nach 50 Jahren, in denen Software entwickelt wird, erwarten könnten. Die Gründe sind hauptsächlich im rasantem Wachstum des Programmumfanges und im Vordringen der Software in immer neue Gebiete der Technik und des menschlichen Lebens zu suchen.
Die Programme erreichen eine unbeherrschbare Größe, welche ein Vorhandensein von derart vielen Fehlern mit sich bringt, dass selbst ein Auffinden einer großen Fehlerzahl noch nicht ausreicht um dem Anwender ein ansatzweise fehlerfreies Produkt zu übergeben.

„Testautomatisierung ist die Durchführung von ansonsten manuellen Testtätigkeiten durch Automaten.“ [BTa15-1]

Der Schritt vom manuellen Testen hin zur Test Automatisierung hat neben geringeren Kosten auch noch andere Vorteile. Die Automatisierung ist ein Werkzeug, welches dem professionellem Software Tester erlaubt seine kreativen Ideen umzusetzen und auch große Testmengen mit vernünftigen Aufwand und in gebotener Zeit zu bewältigen.

Automatisierung [BTa15] ist nicht nur effektiv bei oft wiederholten Abläufen während der Testdurchführung. Sie ist ebenso bei der Testfallerstellung, der Testdatengenerierung, der Testauswertung oder auch der Testumgebungsherstellung und -wiederherstellung einsetzbar.
Ein manueller Testfall kann von menschlichen Tester interpretiert werden und daher wesentlich abstrakter entworfen werden als ein voll automatisierter Testfall. So ist immer projektspezifisch zu bewerten, ob eine Automatisierung vom Aufwand im Verhältnis zum Nutzen steht.

	Testvorbereitung			Testdurchführung (pro Testlauf)		
	Durchschn.	Min.	Max.	Durchschn.	Min.	Max.
Automatisiert	19.2 h	10.6 h	56.0 h	0.21 h	0.1 h	1.0 h
Manuell	11.6 h	10.0 h	20.0 h	3.93 h	0.5 h	24.0 h

Abb. 1: Studie von Dustin zum Aufwand von manuellen und automatisierten GUI-Softwaretests [SWT10-Abb1]

Die Studie von Elfride Dustin von 1999 zeigt, dass der Testvorbereitungsaufwand im automatisierten Fall etwa doppelt so hoch ist wie im manuellen Fall. Der Aufwand für die Testdurchführung ist sehr viel geringer im automatisierten Fall. Da man im Regressiontest in der Regel viele Testdurchläufe plant und eine vergleichsorientierte Verifikationsmethode dafür gut geeignet ist, wird dieser häufig automatisiert.Verschiedene Testarten können im Wesentlichen nur automatisiert durchgeführt werden, zum Beispiel ein Lasttest mit einer Vielzahl von virtuellen Benutzern.

Die Kalkulation [PwSwt08] des zu erwartenden Testaufwands ist eine klassische Projektmanagementaufgabe. Vom Ergebnis hängen insbesondere die Teststrategie und die Aktivitätenplanung der Teststufen ab.
Zu berücksichtigen ist [SWT10], dass der Wartungsaufwand von automatisierten Testfällen höher ist als für manuelle Testfälle. Schließlich sind auch automatisierte Testfälle selbst nur Software und damit den selben Regeln bezüglich guten Designs unterworfen. Besonders kritisch und schwer zu detektieren sind Fehler, die trotz einer Abweichung im Verhalten des zu testenden Systems, einen Testfall nicht fehlschlagen lassen.

Sprich, es ist durch aus auch nötig, den Test kritisch zu betrachten und auch den Test zu testen. Hierfür können Tests eingesetzt werden, bei denen die Fehlerlogik invertiert ist. Oder man baut bewusst einen Fehler in einen Testfall ein und prüft dessen Entdeckung.
Durch Testfall-Debugging werden Fehler im Automatisierungs-Framework sofort behoben und der Testfall erneut durchgeführt.

Entwicklungszyklus

Ebenso [BTa15] spielt der Entwicklungszyklus eine entscheidende Rolle.
Die Testautomatisierung sollte als Teil des Software Tests in den Entwicklungszyklus aufgenommen werden, das heißt, sie sollte auch in der Planung mit ausreichend Zeit und Ressourcen bedacht werden. Ist die Entwicklung der Software schneller als die Entwicklung des Tests, müssen die Tests auf die neue Änderung angepasst werden. Dies aufeinander abzustimmen ist Aufgabe des Testmanagers.

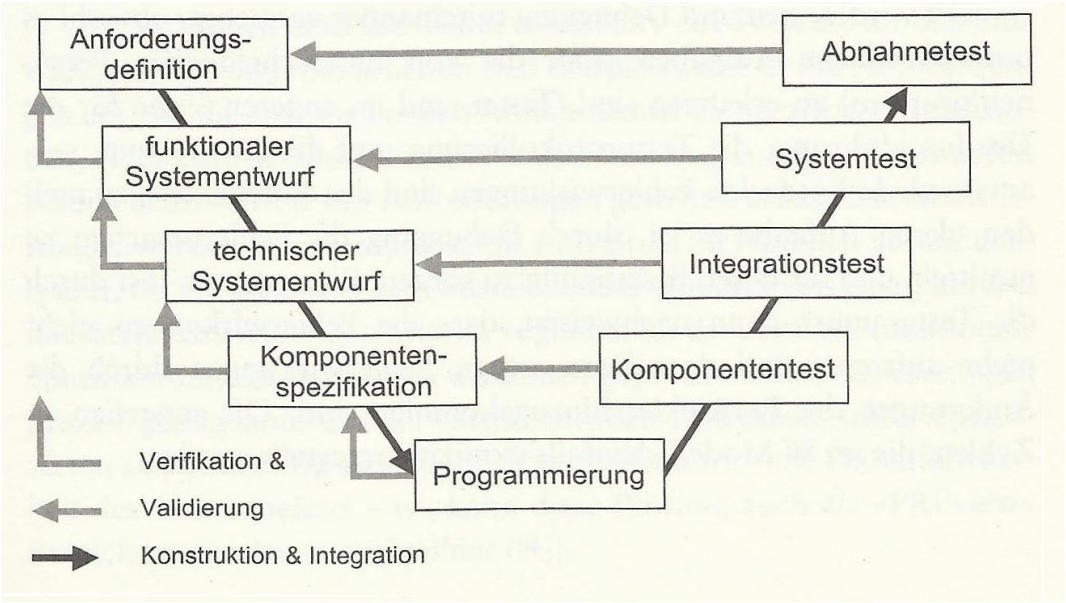

Abb. 2: Allgemeines V-Modell [PwSwt08-Abb1]

Verschiedene Testarten

Funtionale und Nichtfunktionale Tests
Ein funktionaler Test [BaswTe05] umfasst die Prüfung der Testmethoden, die das von außen sichtbare Ein- und Ausgabeverhalten eines Testobjekts beinhalten. Blackbox-Testverfahren werden eingesetzt, das bedeutet, dass der Tester keine Kenntnis von den internen Strukturen der zu testenden Software hat.
Das Verhalten, welches das System oder Systemteile zeigen sollen, wird durch die funktionalen Anforderungen spezifiziert. Sie definieren, was das System leisten soll und deren Umsetzung ist die Grundvoraussetzung für den Einsatz des Systems. Nach ISO 9126 sind Merkmale der Funktionalität: Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit und Sicherheit. Die im V-Modell enthaltene „Anforderungsdefinition“ dient als Projektphase im Entwicklungszyklus der Sammlung von Anforderungen. Diese sollten mit einem Requirement-Werkzeug und /oder einem Lasten- und Pflichtenheft erfasst werden.

Die Attribute des funktionalen Tests werden durch nichtfunktionale Tests beschrieben. Es wird festgelegt in welcher Form und mit welcher Qualität das System eine Funktion beinhalten soll. Die Benutzbarkeit und die Kundenzufriedenheit werden dadurch stark beeinflusst. Merkmale für nichtfunktionale Tests nach ISO 9126 sind: Zuverlässigkeit, Benutzbarkeit und Effizienz.

Folgende nichtfunktionale Tests sollten berücksichtigt sein:

Lasttest:
Messung des Systemverhaltens in Abhängigkeit steigender Systemlast. Simulation von mehreren Benutzern gleichzeitig.

Performanztest:
Messung der Verarbeitungsgeschwindigkeit oder Antwortzeit für bestimmte Anwendungsfälle.

Volumen- Massentest:
Beobachtung des Systemverhaltens in Abhängigkeit zur Datenmenge.

Stresstest:
Beobachtung des Systemverhaltens bei Überlastung.

Datensicherheit:
Ist das System gegen unbefugten Zugriff geschützt?

Test auf Robustheit:
gegenüber Fehlbedienung, Fehlprogrammierung, Hardwareausfall, sowie Prüfung der Fehlerbehandlung und des Wiederanlaufverhaltens (recovery).

Test auf Kompatibilität/Datenkonversion:
Prüfung der Verträglichkeit mit vorhandenen Systemen. Import/Export von Datenbeständen.

Test auf Benutzerfreundlichkeit:
Prüfung von Erlernung und Angemessenheit der Bedienbarkeit.

Prüfung auf Änderbarkeit/Wartbarkeit:
Verständlichkeit und Aktualität der Entwicklungsdokumente, modulare Systemstruktur.

Regressiontests
Je öfter Testfälle wiederholt werden müssen, desto größer kann die Ersparnis an Zeit und Kosten durch die Testautomatisierung werden.
Gerade sich ändernde Systemkonfigurationen können zeitaufwendige Änderungen in der Automatisierung nach sich ziehen. Auch sollte ein geeignetes Intervall gewählt werden. Ein Test der Nightly Builds kann zu oft sein. Hier ist mehr täglicher Aufwand gefordert den Test anzupassen und auszuwerten, als dass er wirklich nützt.

Automatisierte Komponententests
[SWT10] Moderne Unit-Test-Frameworks bieten die Möglichkeit, einen Komponententest direkt in der Entwicklungsumgebung zu definieren und ständig parallel zur Entwicklung durchzuführen beziehungsweise auszuwerten. Unit-Tests werden in der Regel in der selben Programmiersprache und -ungebung definiert wie das Zielsystem selbst und bieten daher auch die Möglichkeit die Komponenten direkt programmgesteuert aufzurufen.
Ein Beispiel ist das Test-Framework JUnit, welches durch Verifikationsmethoden (assertions) den exakten Bereich des Testfalls definieren.

```
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class VerlagTest {
    Verlag m;

    @Before
    public void setUp() throws Exception {
        m = new Verlag();
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    // testen der Berechnung von 0.1 Euro pro Zeichen;
    //1Euro pro 100 Zeichen
    public void testTextToMoney() {
        assertEquals(1, m.textToMoney("abcdefghdr"));

        // testen ob String leer ist

        assertEquals(0, m.textToMoney(""));

        // testen ob gar nichts abgegeben wird
        assertEquals(0, m.textToMoney(null));

        // testen ob String nur aus einem sich
        //wiederholenden Zeichen besteht
    }
}
```

Abb. 3: Code Beispiel für JUnit mit Assertions [JunitAss-Abb1]

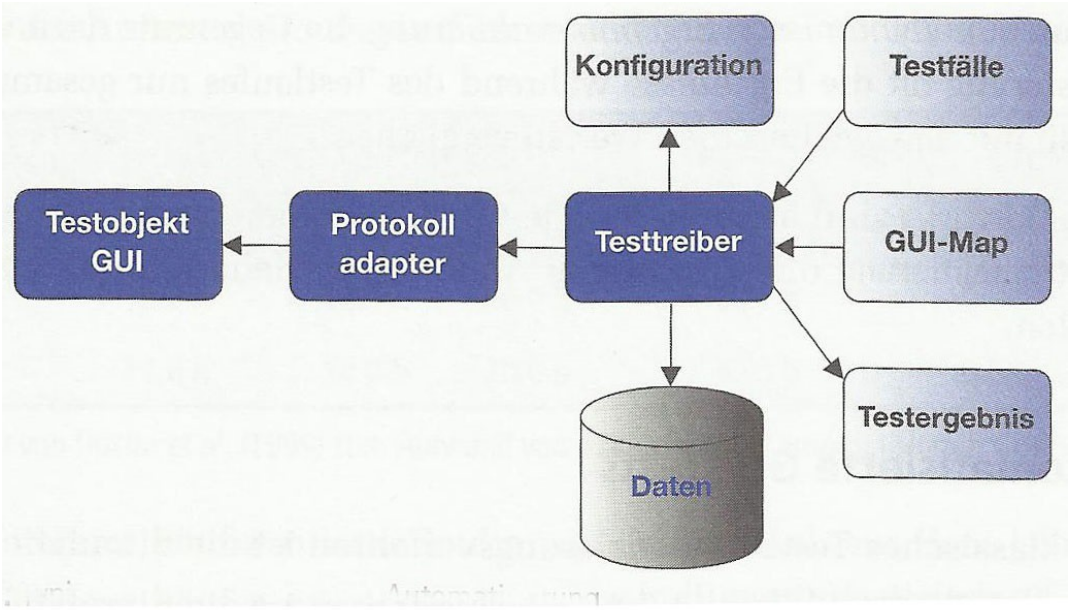

Abb. 4: Typischer Aufbau einer GUI-Automatisierung [SWT10-Abb2]

Automatisierte GUI-Tests [SWT10]
Die Simulation des Benutzers über die Benutzerschnittstelle des Systems ist ein klassisches Testautomatisierungsverfahren. Der Point of Conrol (PoC) sowie der Point of Observation (PoO) ist die Benutzerschnittstelle des Softwaresystems.
Ein übliches Verfahren zur Bestimmung der Aufbauelemente eines GUI-Tests ist das Capture/Replay-Verfahren. Dabei wird die Benutzerinteraktion aufgezeichnet und daraus ein Testskript erzeugt. Nachträglich werden dann in die Benutzerinteraktionen die Assertierungen eingepflegt. Ein Nachteil des Capture/Replay-Verfahrens ist, dass er erst eingesetzt werden kann, wenn das Testobjekt bereits zur Verfügung steht.
Als *Scripting* bezeichnet man das rudimentäre Entwickeln von automatisierten GUI-Tests.
Beides wird in der Praxis oft kombiniert. Hierbei werden zum Beispiel einige initiale Testfälle aufgezeichnet, dann refaktorisiert und die Testdaten ausgelagert. Des Weitern wird die Ansteuerungslogik in einer GUI-Map verwaltet. Die einzelnen GUI-Protokolle (zum Beispiel web, Win32, SAP) werden in Protokolladaptern implementiert und sind damit unabhängig vom Testtreiber und den Testfällen selbst. GUI- Tests können sowohl für funktionale als auch nichtfunktionale Tests(zum Beispiel Performance-Tests) eingesetzt werden.
Je nach Einsatzzweck ändert sich in erster Linie die Art der Verifikation.

Ein zentrales Problem im funktionalen automatisierten GUI-Test ist die Auswahl von geeigneten Verifikationspunkten. Die Möglichkeiten der Automatisierung beschränken sich hier meist auf die Navigation durch das Testobjekt und das Prüfen auf Vorhandensein eines GUI-Elements und dessen qualitativer Überprüfung. Viele potentielle Fehler (Layoutfehler, Synchronisationsfehler, Timing-Verhalten, Reaktion auf unerwartetes Verhalten, Verifikation von Ausdrücken) lassen sich nur schwer erkennen und verifizieren, wobei der Testtreiber immer nur so reagiert, wie es das Skript beziehungsweise dessen Entwickler vorgesehen hat. Automatisierte GUI-Tests können manuelle Softwaretests nicht ersetzen, nur ergänzen.

Testreiber und Testdaten

[SWT10-1] "Als Testtreiber bezeichnet man eine Software, die eine zu testende Software ausführt. Als Testrahmen lädt sie Testdaten, übergibt Werte und protokolliert Ausgaben und Reaktionen der zu testenden Anwendung.
Testdaten werden dazu benötigt alle Entitäten eines Testaufbaus (Zielsystem, Simulatoren und Testtreiber) zu konfigurieren. Ein weiterer wichtiger Verwendungsfall ist das Steuern der Testtreiber während der Testdurchführung mittels geeigneter Testdaten. Dieses wird als datengetriebener Test bezeichnet. Die Trennung von Testfällen und Testdaten ermöglicht es, erst während der Testdurchführung die Testfälle mit konkreten Testdaten zu verbinden. Anstatt des konkreten Wertes wird ein Platzhalter definiert. Dieser Ansatz von datengetriebenen Tests hat wesentliche Vorteile:

- Wiederverwendbarkeit der Testfälle und Testdaten getrennt von einander
- die Wartbarkeit und Lesbarkeit der Testfälle und Testdaten wird erhöht
- Testfälle und Testdatenmengen können in Verwendung stehen
- Testfälle können definiert werden, auch wenn die konkreten Testdaten noch nicht bekannt sind
- Eine Automatisierung der Testfälle ist leichter herzustellen beziehungsweise es ergeben sich besser zu wartende automatisierte Tests."

Konzepte und Tools

Webservices [SOAP-1] wie **SOAP** (Simple Object Access Protocol) oder **REST** (REpresentational State Transfer Architektur) können die Grundlage für die Verwendung von automatisierten Tests sein.
SOAP ist ein Applikations-Kommunikations-Protokoll, welches die Kommunikation zwischen Applikationen auf verschiedenen Plattformen unterstützt. Es ist unabhängig von verwendeten Technologien oder Programmiersprachen.
REST ist eine Architektur für die Kommunikation zwischen Applikationen.
Beide beschreiben eine formale Spezifikation für die Verwendung und Verarbeitung von Antworten.

Fixtures [BTa15] sind Klassen, die vom Automatisierungswerkzeug einen Befehl erhalten um eine bestimmte Aktion durchzuführen. Sie verbinden die Applikation, welche getestet werden soll mit dem Test.
Beispiele für Werkzeuge, die dieses Konzept umsetzen sind Fit, Slim oder FitNesse.

Mocks sind Attrappen für angrenzende Komponenten wie zum Beispiel Datenbanken oder Klassen. Sie werden verwendet um einzelne Komponenten isoliert zu testen und eventuelle Wechselwirkungen auszuschließen. Hierfür wird ein Objekt mit einer Schnittstelle erstellt, die identisch ist mit der Komponente, die simuliert wird.

```
import static org.junit.Assert.*;

import org.junit.Test;
import static org.mockito.Mockito.*;

public class KrkTest {

    @Test
    public void testCalcBeitrag() {
        CalcSteuer csMock = mock(CalcSteuer.class);
        when(csMock.calcNetto(5000)).thenReturn(4000);
        Krk k = new Krk(csMock);
        assertEquals(400, k.calcBeitrag(5000));
    }

}
```

Abb. 5: Code Beispiel für JUnit mit Mock-Objekt [JunitMock-Abb1]

HP Unifed Functional Testing [BTa15]
Ist ein verbreitetes Werkzeug für GUI-Automatisierung und unterstützt hierfür eine große Anzahl von GUI-Technologien von Webapplikationen über .NET und SAP bis zu Windows-Applikationen mit SOAP-Schnittstellen und Terminal-Emulation. Informationen über Schnittstellenobkejte werden mit einem logischen Namen versehen und in „Object-Repositories“ gesammelt. Testschritte können über ein Recording-Modus aufgezeichnet werden. In der Testfall-Ansicht sind Screenshots verfügbar und im Nachhinein können Validierungen und andere Aktionen hinzugefügt werden.

MicroFocus SilkTest [BTa15]
Ist ein Tool für funktionale und Regressiontests von GUIs. Unterstützt werden verschiedene GUI-Technologien angefangen von Adobe AIR über Webapplikationen, bis zu Java, .NET, SAP und Win32-GUIs. Über eine eigene Sprache kann man Testskripte schreiben, welche in GUI Aktionen umgesetzt werden.

IBM Rational Functional Tester [BTa15]
Bietet eine Eclipse-basierte Umgebung zur Erstellung und Durchführung von automatisierten Tests auf Basis von .NET oder Java. Es erstellt Screenshots von der Anwendung und macht es möglich in einfacher englischer Sprache Aufzuzeichnen und zu editieren. Über Object-Maps werden GUI-Elemente verwaltet und aufgezeichnet.

Visual Studio und Coded UI [BTa15]
Der Coded UI Test Builder von Microsoft ist ein in die Entwicklungsumgebung Visual Studio integriertes Capture and Replay Tool. Benutzeraktionen können aufgenommen werden und als UIMaps in einem eigenen Coded-UI-Testprojekt angelegt werden. Bei der Aufzeichnung wird Code generiert, welcher angepasst weiterverwendet werden kann. Last- und Performanz-Tests, sowie Komponententests können angelegt und verwaltet werden.

Selenium [BTa15]
Ist ein sehr verbreitetes Open-Source-Automatisierungswerkzeug für Webapplikationen.
Es besteht aus unterschiedlichen Komponenten:

- Selenium IDE, eine Firefox-Extension für die Aufzeichnung von Testfällen
- Selenium Core für das Abspielen von Testfällen
- Selenium WebDriver, für die Verwendung als Skript- und Programmiersprachen, sowie für die Durchführung von Testskripten auf unterschiedlichen Clientrechnern

Standards und Normen

Standards und Normen definieren allgemein anerkannte Regeln der Technik und damit den fachlich, rechtlichen Bezugsrahmen.
Für jeden Bereich gibt es Normen, die eine Software erfüllen muss, wenn sie dort eingesetzt werden soll.
Wird eine Software beispielsweise in einem Flugzeug eingesetzt, muss sie DO 178 B/ED-12B beziehungsweise DIN EN 14160 erfüllen. Wird eine Software für die Steuerung eines Atomkraftwerkes eingesetzt, muss sie DOE G414.1-4, IAEA TR-384. IAEA NS-G1.1 oder IEC 60880 erfüllen.
Diese Normen betreffen hauptsächlich sicherheitstechnische Aspekte.

Im Thema Software Testing gibt es folgende Normen:

Normart Testphase	Terminologie und Verträge	Prozesse	Produkte	Dokumentation	Methoden und Techniken
Testplanung Teststeuerung	**BS 7925-1** **DIN 55530** ISO 2382 **ISO 9000** ISO 12207-0 **IEEE 610.12** DIN 61508-4 ISO 14598 ISO 15408 IEEE 1062	ISO 12207-0 **ISO 9000-3** **IEEE 730** **IEEE 1008** **IEEE 1012** IEEE 1061	**ISO 9126-x** ISO 12119 ISO 12207-1 ISO 15026 IEEE 982.1 **IEEE 1228**	DIN 55350 **IEEE 730** **IEEE 829** IEEE 1012 IEEE 1228	ISO 12207-2 ISO 16085 IEEE 730.1 IEEE 982.1 **IEEE 1059**
Testanalyse Testdesign		DIN 66273 IEEE 1008 IEEE 1012 ISO 15939	**ISO 9241** ISO 12119 IEEE 1044 IEEE 982.1	**IEEE 829** IEEE1063	**BS 7925-2** IEEE 1028 ISO 60812 DIN 61508-7 EN 50128 DIN 61025 DIN 66241 DIN 66273
Testrealisie-rung und Testdurch-führung		IEEE 1008	ISO 9241 IEEE 828 IEEE 1209 IEEE 1348	**IEEE 829** IEEE 1209 IEEE 1348	IEEE 1028 ETSI 201-873 IEEE 1042 IEEE 1209 IEEE 1348
Testaus-wertung und Bericht		IEEE 1008	IEEE 982.1 **IEEE 1044** **DIN 66271**	**IEEE 829**	IEEE 1044.1 ISO 12119

Abb. 6: Normen und Standards in den Phasen des Testprozesses [PwSwt08-Abb2]

Berufsbild Software Tester und Qualifikation

Entwickler sollten nicht ihre eigene Software testen. Spezialisten, die nur testen, können sich als „ISTQB Certified Tester“ qualifizieren.
Das International Software Testing Qualification Board (ISTQB) [ISTQB] bildet in einem dreistufigen Programm zum Software Tester aus. Die erste Stufe ist das Foundation Level, welches die Grundlagen
im Bereich Softwaretest beinhaltet. Die zweite ist das Advanced Level, welches weiterführende Kenntnisse im Prüfen und Testen von Software beinhaltet, sowie Vertiefungsmöglichkeiten für Black- und Whitebox-Testverfahren im Modul „Technical Test-Analyst“ oder „Test Analyst“ oder zum Thema Testmanagement als „Test Manager“.
Die letzte Stufe ist das Expert Level. Dieses beinhaltet die Module Test Process Management und Test Management. Geplant sind noch Module zu Test Automation und Security Test.

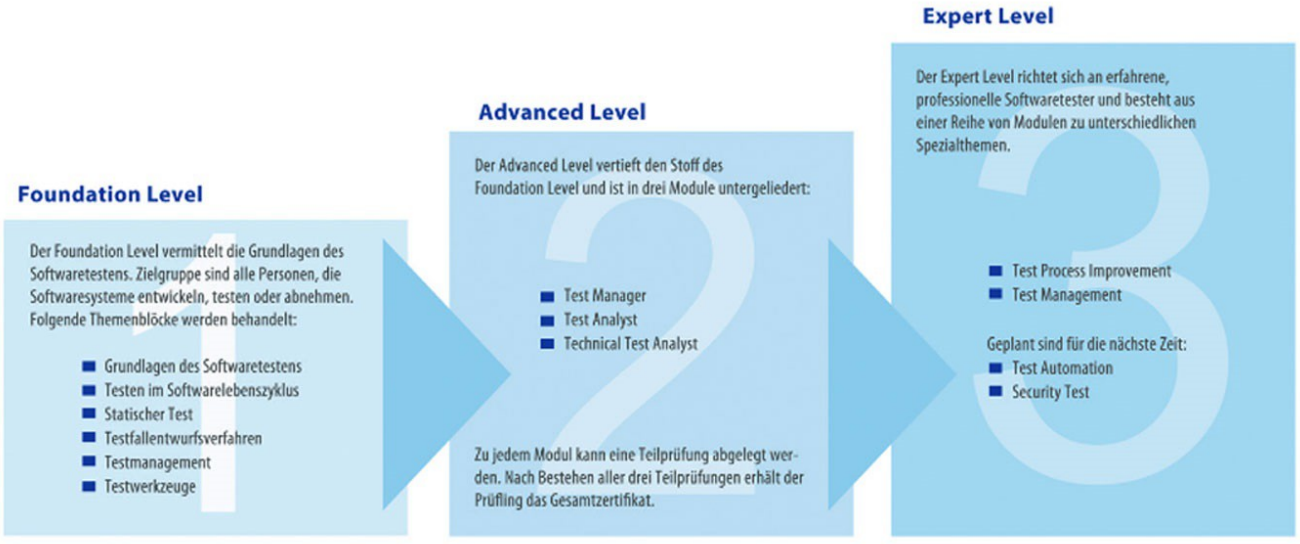

Abb. 7: Level der ISTQB-Qualifizierung [ISTQB-Abb1]

Zusammenfassung und Fazit

Mit immer größer werdenden Software Projekten und weiterer Verbreitung der Verwendung von Applikationen durch die voranschreitende Digitalisierung und Ausbreitung der mobilen Anwendungen wird der Einsatz von fehlerarmer Software immer wichtiger. Auch dadurch, dass eine steigende Zahl der Nutzer Digital Natives sind, steigt auch der Anspruch an die Qualität der Software. Die Markt- und Technologienenwicklung sind rasant. Keine Firma kann sich einen Imageverlust durch Fehlverhalten ihrer Software leisten. Deshalb wird es immer wichtiger Software umfangreich zu testen. Wie bei jedem Produkt spielen die Kosten eine wichtige Rolle. Die Automatisierung kann Zeit und Arbeit sparen. Die Angst um Arbeitsplätze durch die Einführung von automatisierten Software-Tests ist weitgehend unbegründet, denn die Tests bedürfen ebenfalls einer Entwicklung, Wartung und Auswertung. Es ist mit der ISTQB-Qualifizierung sogar ein neues Berufsfbild entstanden. Outsourcing an Orte mit geringen Personalkosten ist nach wie vor in allen wirtschaftlichen Bereichen ein Thema. Jedoch ergibt sich hier auch wieder eine Produktionskette, die kommuniziert und überwacht werden muss. Hocheffiziente Teams vermeiden diese Entwicklung durch effektives Projektmanagement.

Quellenverzeichnis:

[SWT10] Thomas Grechening, Mario Bernhart, Roland Breteneder, Karin Kappel: Softwaretechnik, Pearson Studium 2010, München, ISBN 978-386894-007-7

[SWT10-1] Thomas Grechening, Mario Bernhart, Roland Breteneder, Karin Kappel: Softwaretechnik, Pearson Studium 2010, München, ISBN 978-386894-007-7, Seite 336

[S-T02] Georg Erwin Thaller: Software-Test, Verifikation und Validation, Verlag Heinz Heise GmbH &Co KG, Hannover, ISBN 3-88229-198-2

[BTa15] Thomas Bucsics, Manfred Baumgartner, Richard Seidl, Stefan Gwihs: Basiswissen Testautomatisierung, Konzepte, Methoden und Techniken, dpunkt.verlag 2015 Heidelberg, ISBN 978-3-86490-194-2

[BTa15-1] Thomas Bucsics, Manfred Baumgartner, Richard Seidl, Stefan Gwihs: Basiswissen Testautomatisierung, Konzepte, Methoden und Techniken, dpunkt.verlag 2015 Heidelberg, ISBN 978-3-86490-194-2, Seite 7

[PwSwt08] Andreas Spillner, Thomas Roßner, Mario Winter, Tilo Linz: Praxiswissen Softwaretest, Testmanagement, Aus- und Weiterbildung zum Certified Tester, Advanced Level, nach ISTQB-Standard, dpunkt.verlag, 2008 Heidelberg, ISBN 978-3-89864-557-7

[BaSwTe05] Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, Aus- und Weiterbildung zum Certified Tester, Foundation Level, nach ISTQB-Standard, dpunkt.verlag, 2005, Heidelberg, ISBN 3-89864-358-1

Onlinequellen:

[F01] Website School of Mathematics, University of Minnesota: http://www.math.umn.edu/~arnold/disasters/ariane.html (letzter Stand 10.11.2016)

[F02] Website Heise online: http://www.heise.de/newsticker/meldung/Hartz-IV-GAU-bei-der-Arbeitslosengeld-II-Zahlung-Update-124331.html (letzter Stand 10.11.2016)

[SOAP-1] http://www.w3schools.com/xml/xml_soap.asp (letzter Stand 17.12.2016)

[ISTQB] German Testing Board: http://www.german-testing-board.info/ (letzter Stand 16.12.2016)

Abbildungen:

[SWT10-Abb1] Abbildung 1: Thomas Grechening, Mario Bernhart, Roland Breiteneder, Karin Kappel: Softwaretechnik, Pearson Studium 2010, München, ISBN 978-386894-007-7, Seite 332

[PwSwt08-Abb1] Abbildung 2: Andreas Spillner, Thomas Roßner, Mario Winter, Tilo Linz: Praxiswissen Softwaretest, Testmanagement, Aus- und Weiterbildung zum Certified Tester, Advanced Level, nach ISTQB-Standard, dpunkt.verlag, 2008 Heidelberg, ISBN 978-3-89864-557-7, Seite 33

[JunitAss-Abb1] Abbildung 3: Beispiel-Code aus dem Fach Softwaretechnik von meiner Übung WS 15/16

[SWT10-Abb2] Abbildung 4: Thomas Grechening, Mario Bernhart, Roland Breiteneder, Karin Kappel: Softwaretechnik, Pearson Studium 2010, München, ISBN 978-386894-007-7, Seite 334

[JunitMock-Abb1] Abbildung 5: Beispiel-Code aus dem Fach Softwaretechnik von meiner Übung WS15/16

[PwSwt08-Abb2] Abbildung 6: Andreas Spillner, Thomas Roßner, Mario Winter, Tilo Linz: Praxiswissen Softwaretest, Testmanagement, Aus- und Weiterbildung zum Certified Tester, Advanced Level, nach ISTQB-Standard, dpunkt.verlag, 2008 Heidelberg, ISBN 978-3-89864-557-7, Seite 373

[ISTQB-Abb1] Abbildung 7: http://www.german-testing-board-.info/fileadmin/gtb_repository/downloads/pdf/news/de/2012-12_metzger_muenzel_OS_01_13_z5yc.pdf (letzter Stand 16.12.2016)