REZENSION

UWE VIGENSCHOW
OBJEKTORIENTIERTES TESTEN UND TESTAUTOMATISIERUNG IN DER PRAXIS
KONZEPTE TECHNIKEN UND VERFAHREN
DPUNKT.VERLAG 2005

KURZBEWERTUNG

„Objektorientiertes Testen und Testautomatisierung in der Praxis“ behandelt nicht nur den Bereich Testen, sondern auch eine Fülle von Themen der Qualitätssicherung. Wie der Autor in seinen einleitenden Worten darlegt, geht es ihm um „einfache, wirkungsvolle Verbesserungen im Rahmen unserer täglichen Softwareentwicklung“, die einer „Gut-genug-Strategie“ folgen. Seine Zielgruppe sind dabei nicht Qualitätsmanager und deren Anspruch an ein „System“, sondern die Entwickler, deren Sicht er als Ausgangspunkt seiner Überlegungen wählt und darauf aufbauend konkrete Umsetzungsvorschläge entwickelt. Er wird diesen Ansprüchen und seiner Zielgruppe durchwegs gerecht.

DAS WESENTLICHE IM DETAIL

Teil I begründet, warum wir Tests benötigen. Der Autor führt hier vor allem die für ein technisches Buch eher überraschenden Themen wie Kommunikation und unser Gedächtnis, die oft zu beobachtende Unfähigkeit von Fachbereich und Softwareentwicklern, einander zu „verstehen“ und dann erst die Komplexität der Software, als Quellen von Fehlern an. Schon in der Einleitung arbeitet er die positive Einstellung zu Fehlern als den Schlüssel erfolgreicher Projektarbeit heraus. Er nennt diese Einstellung positive Fehlerkultur, der er zu Recht ein ganzes Kapitel widmet.

Besonders hervorzuheben ist die sehr kompakte, aktuelle Sammlung der wesentlichen Definitionen und Begriffe rund um die Themen Qualität, Qualitätsmerkmale, Anforderungen, Fehler und Testen, zitiert aus den gängigen ISO-Normen und insbesondere der führenden Ausbildung zum Tester, dem Certified-Tester-Programm. Besser kann man es sich auf elf Seiten kaum wünschen.

Teil II behandelt sechs Themenkomplexe, gruppiert um die Bereiche Technisches, Analytik, Methodik und Organisation.

Kapitel 5 „**Lösungen für technische Probleme**“ sticht mit einer Fülle an fundierten Vorschlägen für einen Coding Style Guide heraus. Es kann jeder Entwicklergruppe nur empfohlen werden, sich hiermit zu beschäftigen.
Genauso spannend sind die Ausführungen zum Thema Debuggen, wo auf die notwendigen Fähigkeiten eingegangen wird, die Entwickler mitbringen sollten - zumindest Kenntnisse des Projektes, der verwendete(n) Sprache(n), der benutzten Techniken und optional je ein Teammitglied das Betriebssystem und/oder den Prozessor. Der vorgestellte Debugging-Prozess, die Systematik der Fehlersuche, überzeugt mit seinen Analyseschritten vom Einfachen zum Komplexen. Damit kann dem häufig zu beobachtenden chaotischen Treiben vorgebeugt werden, auch unter Zeitdruck, sofern die dazu notwendige Standhaftigkeit aufgebracht wird.

Das Kapitel 6 „**Lösungen für analytische Probleme**“ beschreibt das FURPS-Modell (= Functionality, Usability, Reliability, Performance, Supportability) als Gedankenstütze zur Bestimmung, was getestet werden muss. Anschließend wird die Ableitung von Testfälle grob angerissen und für Testdatenermittlung sowie die jeweiligen Testarten – Unit-Testfälle, Klassenmethoden, Kettentests und Systemtestfälle – sehr knapp beschrieben und auf die folgenden Kapitel verwiesen.

Kapitel 7 „**Lösungen für methodische Probleme**“ legt die Problematik beim Test durch den Entwickler dar, die eigene Arbeit destruktiv zu betrachten. Daraus werden die Vorteile des testgetriebenen Ansatzes zuerst Testfälle zu generieren und dann zu programmieren mit zwei Argumenten untermauert abgeleitet, einerseits einer tiefer gehenden Analyse, das heißt eben vorab die richtigen Fragen zu stellen und andererseits durch

A-1160 Wien, Lorenz Mandl-Gasse 47/29, Tel. & Fax: +43 1 493 28 15, Mobil: +43 (0) 676 3550 686
email: dietmar.kropfitsch@kropfitsch-consulting.com, http://www.kropfitsch-consulting.com

das Schreiben der Testfälle auch ein besser testbares Design zu erarbeiten. Da die Entwickler „Nebenbei"-Tester sind, wirkt die Vielzahl an Testmethoden abschreckend. Dieses Faktum berücksichtigt der Autor bei der Vorstellung der wesentlichen Testmethoden Grenz-/Extremwerte, Äquivalenzklassen, Überdeckungen (Anweisung, Zweig, Pfad, Schleifen, Bedingungen) anhand kurzer und prägnanter Beispiele.
Für Codereviews, eine weitere effiziente „Testtechnik", vom Autor als asynchrones Pair Programming bezeichnet, findet man pragmatische und mit geringem Aufwand umsetzbare Durchführungshinweise.

Kapitel 8 behandelt „**Lösungen für fortgeschrittene Probleme**", wobei hier die Erläuterungen für das zustandsraumbasierte Testen in der Darstellung überzeugen, während die Themen Rekursion und Nebenläufigkeit in ihrer Vielfalt beschrieben werden und eine Fülle an Hinweisen zum Test enthalten, im Verhältnis zu den anderen Kapiteln aber das eine oder andere Beispiel vermissen lassen.

Kapitel 9 „**Lösungen zum Test objektorientierter Software**" legt dar, in welcher Reihenfolge objektorientierte Software getestet werden soll: Test von oben nach unten zuerst entlang den Assoziationen einer Ebene, dann in der Vererbungshierarchie eine Ebene tiefer gehen und dort entlang den Assoziationen testen, dann wiederum in einer Ebene tiefer die Assoziationen testen, bis man auf der untersten Ebene angelangt ist.
Sehr intensiv wird auf die Vererbung eingegangen, insbesondere auf Überlegungen zum Testumfang mit dem Ansatz des „Flattening". Die Betrachtung von sieben Fehlern, die gerne gemacht werden und eine detaillierten Behandlung von drei der insgesamt 37 Testmustern nach Binder – Modale Klassen, modale Hierarchie, nicht-modaler polymorpher Server - rundet dieses Thema ab.
Abgeschlossen wird dieses Kapitel mit einer schlüssig aufbereiteten Anleitung zur Priorisierung von Tests für spezielle Klassenarten auf Basis der Designleitlinie Entity-Control-Boundary (ECB). Dabei hat man sich die zentrale Frage zu stellen: „Wo sind die komplexen, fehleranfälligen Abläufe und Zusammenhänge im Code repräsentiert?" Dieses Thema gewinnt insbesondere dann an Bedeutung, wenn die Zeit sehr knapp ist.

Kapitel 10 „**Lösungen für organisatorische Probleme**" erläutert die Prinzipien der Vorgehensmodelle Wasserfall und V-Modell, des eXtreme Programming und stellt das inkrementell-iterative Vorgehen vor. Für letzteres wird klar herausgearbeitet, dass die Phase „Entwurf und Architektur" mit einem geprüften, stabilen und abgenommenen Architekturkonzept inklusive seiner ersten Implementierung abgeschlossen werden muss. Dies ist insofern von zentraler Bedeutung, weil die Produktreife von Anfang an zu erreichen ist. Die eigentliche inkrementelle Entwicklung erfolgt dann in der Konstruktion, also der Implementierung aller weiteren Funktionen.

Der Planungsansatz, je Inkrement auch weniger wichtige Arbeitspakete quasi als Buffer aufzunehmen und dann bei Terminproblemen zugunsten der Fertigstellung der wichtigen Arbeitspaket gestrichen werden können, ist interessant, dürfte aber nur von Organisationen, die auf dem Gebiet der Planung einen hohen Reifegrad erreicht haben, umsetzbar sein.

Sehr gut gefällt der Vorschlag der Prozessgestaltung „Innen inkrementell-iterativ, außen V-Modell", wobei kundenseitig mit dem V-Modell und die Implementierung selbst inkrementell-iterativ geplant werden.

Der Autor stellt das testgetriebene Design im Detail vor. Der Wert dieses Ansatzes liegt darin, dass beim Erstellen der Testfälle analytische Fragen gestellt, Fehler früher gefunden und lange Debugging-Sessions vermieden werden. Redesign/Refactoring wird so überhaupt erst vertretbar, allerdings sind automatisierte Tests die Voraussetzung dafür. Die Konsequenz daraus ist, dass Entwicklungszeit für die notwendigen Testfälle, insbesondere in der Wartungsphase, vorzusehen ist. Unbedingt zu beachten ist die Reihenfolge der Arbeitsschritte: Refactoring vor Weiterentwicklung. Damit gewinnt man die Sicherheit, dass Refactoring „fehlerfrei" gelungen, ist bevor man neue Funktionen implementiert.

Für die Aufwandsbetrachtungen stellt der Autor eine umfangreiche Liste der Tätigkeiten im gesamten Entwicklungsprozess zur Verfügung. Er schlägt vor, je Inkrement zu schätzen. Die häufige Durchführung der Abschätzungen sollte zu stetiger Verbesserung der Schätzgenauigkeit mit Abweichungen unter 10% führen. Als Problem sieht er eher, alle notwendigen Arbeitspakete zu finden. Die Vorstellung eines Fehlermodells zur Aufwandsschätzung rundet das Thema ab.

Aufschlussreich sind die Argumente dafür, das testgetriebene Vorgehen trotz Zeitdrucks beizubehalten. Zwei „Feldversuche“ werden geschildert:

- Ein Team arbeitete trotz hohen Zeitdrucks bis zum Ende testgetrieben und erreichte eine halb so hohe Fehlerdichte unter denselben Rahmenbedingungen wie ein zweites Team, das das testgetriebene Vorgehen im Laufe des Projektes aufgab.
- In einem Selbstversuch hat ein Kollege des Autors die eine Hälfte der Klassen herunterprogrammiert, die andere Hälfte testgetrieben entwickelt. Im Durchschnitt benötigte er je Klasse eine Stunde, hat aber bei testgetriebenem Ansatz neben dem Produktcode auch noch automatisierte Testfälle sowie etwas Dokumentation erstellt. Sein subjektives Empfinden war jedoch, testgetrieben wesentlich langsamer zu sein.

Teil III „Umsetzung in die Praxis“ legt den Schwerpunkt auf eine Einführung zur Testautomatisation mit der xUnit-Familie, zieht für die praktische Umsetzung ein Resümee hinsichtlich Anforderungen an die Organisation von Softwareentwicklungsprojekten, liefert eine Umsetzungsstrategie und legt einen „emotionsloseren Umgang mit Fehlern nahe.

Eingangs Kapitel 11 **„Automatisierung von Entwicklertests“** wird das Konzept der xUnit-Familie

- Trennung von Code und Test
- Tests in derselben Sprache wie der Code
- Automatisierter Ablauf, von jedem startbar
- Verifikation und Protokollierung des Testergebnisses durch den Test selbst
- Einfache Kombination von Tests verschiedener Autoren

vorgestellt und anschließend ein organisatorisches Problem bei großen Projekten, die fehlende Trennung von Testablauf und Testdaten behandelt. Dazu sind Zusatztools notwendig, z.B. JTestCase, das die Daten als XML-Dateien hält und den Einsatz einer XML-DB ermöglicht, ohne viel Handarbeit und einen Testtoolverantwortlichen aber nur ineffizient einsetzbar ist.

Drei Tools, JUnit, CppUnit (C++-Version) und NUnit (JUnit unter .NET) werden mit ihren Eigenheiten vorgestellt. Drei JUnit-Testbeispiele beschreiben die konkrete Umsetzung, die Listings sind im Anhang verfügbar.

Auf wesentliche Konstrukte zur erfolgreichen Implementierung automatisierter Testfälle wird intensiver eingegangen, die Stellvertreterobjekte – Stub, Dummy, Mock. Sie werden im Klassentest, spätestens aber im Klassen-Integrationstest notwendig, um frühzeitig testen zu können. Der Vorteil liegt darin, Hintertürchen für zusätzliche get-Methoden zu erhalten, Grenzfälle einfacher simulieren zu können und keinen testspezifischen Code in die auszuliefernde Software aufnehmen zu müssen.

Zur Testautomatisation über die GUI gibt der Autor einen kurzen Überblick zu diesem Thema mit den unterschiedlichen Ansätzen, lineare bis zu schlüsselwortgetriebenen Skripte mit ihren Vor- und Nachteilen.

In Kapitel 12 „**Was haben wir aus der Betrachtung der Verfahren gelernt**“ leitet der Autor Kriterien für erfolgreiche Projekte, Anforderungen an das Entwicklerteam und den Projektleiter ab. Insbesondere das Bild, das Entwicklerteam als Chor und nicht Fußballmannschaft zu begreifen, ist exzellent: „So kann ich als Torwart oder Feldspieler trotz einer Niederlage der Mannschaft eine gute Einzelkritik bekommen. Beim Chor reicht hingegen ein schlechter Sänger aus, um das Gesamtergebnis zu ruinieren.“

Dass der beste Entwickler selten die wichtigsten Fähigkeiten eines Projektleiters besitzt - Ausgleich aller kommunikativen, führungsthematischen und organisatorischen Defizite - sollte inzwischen Allgemeinwissen sein, die „Sachzwänge“ im täglichen Leben scheinen diese Erkenntnis häufig hinwegzufegen. Interessant ist das Bild, Projektmanagement als angewandte Mängelverwaltung zu bezeichnen. Sehr treffend!

Kapitel 13 „**Teststrategie: Der Weg ist wichtiger als das Ziel**“ ist ein Plädoyer für „mittelmäßiges, eher pragmatisches QS-Konzept erfolgreich umgesetzt ist von deutlich größerem praktischen Nutzen als ein optimiertes Konzept, dessen Umsetzung nicht gelingt.“ Der Empfehlung, kleine Schritte zu gehen, Überzeugungsarbeit zu leisten, treue Weggefährten für den langfristigen Weg zu gewinnen und im Tagesgeschäft immer wieder Zeit für Verbesserungen freihalten, kann nur uneingeschränkt zugestimmt werden.

Insgesamt gelungen ist die Entwicklung einer pragmatischen Entwicklertest-Strategie mit Verweis auf die im Buch behandelten Themen sowie die Empfehlung der Reihenfolge, technische, analytische, testmethodische und erst dann organisatorische Maßnahmen zu setzen und die notwendige Weiterbildung sowohl intern als auch extern abzudecken.

Der Autor nimmt sich in Kapitel 14 des zentralen Themas im Umfeld Qualitätssicherung und Test, der „**Fehlerkultur**“, an. Nur eine sachliche, ergebnisorientierte Atmosphäre, insbesondere dann, wenn große Probleme zur Lösung anstehen, führt zum Erfolg, daher ist es entscheidend, eine konstruktive Fehlerkultur „aus Fehlern lernen“ zu fördern. Damit einher geht die Initiative als Voraussetzung zur kreativen Entwicklung von Lösungen: „Der Grad der möglichen Freiheitsgrade im Denken bestimmt unsere Innovationsfähigkeit.“. Ein innovationsförderndes Umfeld wird durch vier Elemente geschaffen: Freiräume, Handlungsentlastung, Interessendeckung, konstruktive Fehlerkultur. Der Aussage „In konstruktiven Fehlerkulturen werden mehr Fehler gemacht, jedenfalls offiziell.“ muss nichts hinzugefügt werden.

Der **Teil IV** liefert vertiefende Anregungen zum „**Test von Realtime und Embedded Systems**“, eine Gegenüberstellung von „**UML 1.5 vs. UML2**“ und als Ausklang den Wunsch, dass das „**Zusammenwachsen von Entwicklung und Qualitätssicherung**“ gelingen möge.

Ein umfangreicher Anhang mit allen Beispielen, 37 objektorientierten Testmustern, Glossar, Abbildungs-, Tabellen- und einem exzellenten Literaturverzeichnis zum Thema Testen rundet dieses Werk ab.

RESÜMEE

Insgesamt ist ein kompaktes, einführendes Buch entstanden, das in sehr flüssigem Stil geschrieben leicht zu lesen ist und sich vor allem durch eine einprägsame, bildhafte Sprache, gute Vergleiche, gerade so viel Theorie wie nötig, viele Details aus der Praxis und einfache Beispiele auszeichnet. In erster Linie für Entwickler geschrieben, können auch langjährige Qualitätssicherer und Tester eine Fülle an Informationen, Gedanken, Anregungen und neue Blickwinkel für ihre Praxis entdecken.

A-1160 Wien, Lorenz Mandl-Gasse 47/29, Tel. & Fax: +43 1 493 28 15, Mobil: +43 (0) 676 3550 686
email: dietmar.kropfitsch@kropfitsch-consulting.com, http://www.kropfitsch-consulting.com
