Elisabeth Hendrickson, Explore It!, dpunkt.verlag, ISBN 978-3-86490-093-8

Elisabeth Hendrickson (@testobsessed) arbeitet als Testerin, Entwicklerin und »Agile-Enabler«. Im Jahr 1980 schrieb sie ihre erste Codezeile und fand sofort ihre ersten Fehler. 2010 gewann sie den renommierten Gordon Pask Award von der Agile Alliance. Sie ist bekannt für ihren Google Tech Talk über Agile Testing sowie ihre beliebten Testheuristiken-Spickzettel. Sie teilt ihre Zeit auf zwischen lehren, vortragen, schreiben, programmieren und der Arbeit in agilen Teams, die ihr Engagement beim Testen sehr schätzen.

Übersetzerin: Meike Mertsch arbeitet als begeisterte Testerin für Magine AB in Stockholm, Schweden. Sie hat einen Hintergrund als agiler Entwickler und Coach und ein Faible für leichtgewichtige Methoden wie Kanban und Personal Kanban sowie agiles Entwickeln und Testen. In ihrer Freizeit läuft und skatet sie durch Stockholm oder macht die Kletterhallen und Felsen in der Nähe unsicher.

Elisabeth Hendrickson

Explore It!

Wie Softwareentwickler und Tester mit explorativem Testen Risiken reduzieren und Fehler aufdecken

Aus dem Amerikanischen übersetzt von Meike Mertsch

dpunkt.verlag

Elisabeth Hendrickson
elisabeth@testobsessed.com

Übersetzung: Meike Mertsch, Stockholm
Lektorat: Christa Preisendanz
Copy-Editing: Ursula Zimpfer, Herrenberg
Satz: Petra Strauch, just in print, Bonn
Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-86490-093-8

1. Auflage, Translation Copyright für die deutschsprachige Ausgabe © 2014 dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Copyright der amerikanischen Originalausgabe © 2013 The Pragmatic Programmers, LLC.
Title of American original: Explore It! Reduce Risk and Increase Confidence with Exploratory Testing
Pragmatic Bookshelf, The Pragmatic Programmers, LLC, Dallas, Texas, Raleigh, North Carolina
http://pragprog.com
ISBN-13: 978-1-937785-02-4

5 4 3 2 1 0

1 Testen und Erforschen

Egal, welche Berufsbezeichnung Sie haben, Sie werden höchstwahrscheinlich regelmäßig bei Ihrer täglichen Arbeit testen. Testen ist ein integraler Bestandteil jedes Schaffensprozesses. Solange, bis Sie die Software testen – also mit der Software oder dem System interagieren, das tatsächliche Verhalten beobachten und mit Ihren Erwartungen vergleichen –, ist alles, was Sie darüber zu wissen glauben, nicht mehr als bloße Spekulation.

Robert Slater erzählt in *Portraits in Silicon* [Sla89] die Geschichte des Teams, das einen der ersten Computer baute, den ENIAC. Die ersten Computer waren riesig und füllten ganze Räume. Beim Untersuchen des Inneren einer dieser Maschinen sah man Bauteile in Metallgestellen mit großen Kabelbündeln, die sie verbanden. Die Wahl der Kabel wurde also zu einer wichtigen Designentscheidung. Slater erklärt:

> »Es gab die latente Gefahr, dass Mäuse die Kabel anfressen würden. Für einen Test wurden einige Mäuse in einen Käfig gesperrt, mussten einige Zeit hungern und wurden dann mit verschiedenen Kabelarten konfrontiert. Als man herausfand, dass die für ENIAC geplanten Kabel von den Mäusen geliebt wurden, entschied man sich für andere Kabel, die den Mäusetest bestanden.«

Beachten Sie, dass die Teammitglieder ein Risiko erkannten und in eine Fragestellung umwandelten, die sie beantworten konnten. Sie haben nicht über die Nahrungsvorlieben von Nagetieren spekuliert, sondern einigen hungrigen Mäusen verschiedene Kabelsorten vorgesetzt. Sie haben die Ergebnisse ihres Experiments genutzt, um ihre Handlungen zu leiten. Das ist der Kern des Testens: Ein Experiment zu gestalten, mit dem man empirische Beweise sammelt, um eine Frage zu einem Risiko zu beantworten.

Verschiedene Testarten beantworten verschiedene Fragestellungen. Wenn Sie wissen wollen, wie gut ein System unter einer bestimmten Gesamtlast läuft, werden Sie einen Performance-Test durchführen. Wenn Sie wissen wollen, ob ein kleines Stück Code sich so verhält, wie es der Programmierer vorgesehen hatte, werden Sie dieses Stück in einer Reihe Unit Tests isolieren. Wenn Sie wissen wol-

len, ob sich die Anwender ohne Hilfe in der Software zurechtfinden, dann werden Sie einen Usability-Test ansetzen.

Dieses Kapitel erklärt, wie sich exploratives Testen von anderen Testtechniken unterscheidet und wie es in die allgemeine Teststrategie passt.

1.1 Zwei Seiten des Testens

Es ist zwanzig Jahre her, aber ich erinnere mich an das Gespräch, als wenn es gestern gewesen wäre. Marchell, eine meiner Kolleginnen, zeigte auf einen zentimeterdicken Stapel Papier auf ihrem Schreibtisch: Testfälle, die gerade mal einen kleinen Teil des Softwarepakets abdeckten, das wir gerade testeten.

»Es ist so frustrierend«, stöhnte sie. »Egal, wie viele Tests wir schreiben oder wie viele Fälle wir durchführen, wir finden die schlimmsten Fehler immer erst dann, wenn wir vom Skript abweichen.«

Zu der Zeit kannte ich den Begriff *exploratives Testen* noch nicht, obwohl ihn Cem Kaner bereits 1988 in seinem Buch *Testing Computer Software* [Kan88] geprägt hatte. Ich wusste nur, dass Marchell Recht hatte. Egal, wie viele Testfälle wir zu unserer Sammlung hinzufügten, wir fanden immer Überraschungen, wenn wir vom Skript abwichen.

In den zwei Jahrzehnten seit dieser Unterhaltung habe ich dieses Muster immer wieder gesehen: Egal, wie viele geplante Testfälle ein Team durchführt, es gibt scheinbar immer noch Überraschungen zu finden.

Wenn eine Firma die Software auf dem freien Markt veröffentlicht, können die Überraschungen sogar noch schlimmer sein.

Nutzer tun die absurdesten Dinge. Produktivdaten haben die gemeine Tendenz, sich von erdachten Beispielen zu unterscheiden. Echte Konfigurationen sind selten so gepflegt und kontrollierbar wie Testsysteme.

Die echte Welt ist ein chaotischer Ort.

Es ist frustrierend, aber nicht abzustreiten: Sie können nicht alle Tests im Voraus planen, um jeden Fall abzudecken. Es gibt zu viele Variationen der Daten, Konfigurationen, Interaktionen, Abfolgen und Zeitabläufe. Wenn Sie versuchen, eine vollständige Testsammlung für jede Eventualität zu erstellen, werden Sie all Ihre Zeit darauf verwenden, Tests zu schreiben, und haben keine Zeit mehr, sie auszuführen.

Was Sie brauchen, ist nicht die perfekte vollständige Testsammlung. Stattdessen benötigen Sie eine Teststrategie, die zwei grundlegende Fragen beantwortet:

1. Verhält sich die Software wie geplant unter den Bedingungen, die sie abdecken sollte?
2. Gibt es weitere Risiken?

1.1.1 Checking (dt.: Prüfen)

Sie können die erste Frage mit Tests beantworten, die Sie im Voraus erstellen, um zu überprüfen, ob sich die Implementierung mit den unterstützten Konfigurationen und unter bestimmten Bedingungen wie vorgesehen verhält.

Sie können sich diese Prüfungen wie ein Netz aus Stolperdrähten vorstellen, das immer dann ausgelöst wird, wenn sich das Verhalten der Software nicht mit den Erwartungen an sie deckt, wie in Abbildung 1–1 zu sehen. Je besser die Abdeckung durch die Prüfungen ist, desto feiner ist das Netz gewebt.

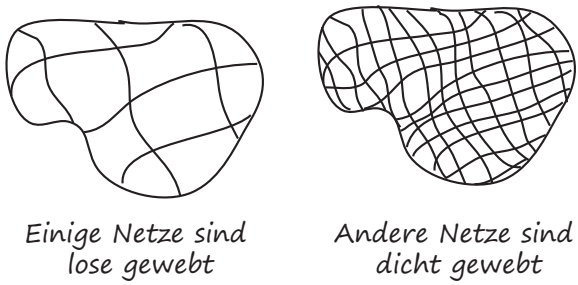

Abb. 1–1 *Tests kann man sich als Netz vorstellen.*

Auch wenn Sie ein feines Netz gewebt haben, werden Sie immer noch die zweite Frage beantworten müssen. Und das ist der Zeitpunkt, an dem das Forschen ins Spiel kommt.

1.1.2 Erforschen

Mit explorativem Testen können Sie auch die Bereiche erkunden, die das Netz nicht abdeckt. Sie interagieren mit der Implementation, erstellen kleine Experimente und führen diese in kurzer Abfolge durch, während Sie die Ergebnisse eines Experiments nutzen, um das nächste zu gestalten.

Während Sie potenzielle Risiken erkennen, gehen Sie tiefer ins Detail. Sie nutzen Ihre Beobachtungs- und Analysefähigkeiten, um Ihre Untersuchungen unterwegs anzupassen. Ihre Experimente liefern Ihnen empirische Hinweise über die Möglichkeiten und Grenzen Ihrer Software. Entlang des Weges entdecken Sie neue Fragen, die einer Antwort harren, und planen neue Testarten ein.

Bei einer Erkundung kann man sich durch die unendlich vielen Möglichkeiten der Varianten bewegen und in Richtung der Risiken steuern. Ihre vorher geplanten Tests können das so nicht. Um weitere Überraschungen zu entdecken, wird Ihnen nicht Wiederholbarkeit helfen, sondern Variation.

Die zwei Fragen stellen also die beiden Facetten des Testens dar: prüfen, dass die Software Erwartungen erfüllt, und Risiken erforschen. Weder Prüfen noch Erforschen reichen ohne das jeweils andere aus.

1.1.3 Getestet = geprüft und erforscht

Sie sind mit dem Testen nicht fertig, bis Sie geprüft haben, dass die Software die Erwartungen erfüllt, und erforscht haben, ob es weitere Risiken gibt. Eine umfassende Teststrategie vereinigt beide Ansätze.

Beim Lesen dieses Buches sollten Sie im Hinterkopf behalten, dass es nur die explorative Seite der Gleichung behandelt. Dieses Buch ist ein Leitfaden für Techniken, um Überraschungen zu entdecken, und keine vollständige Behandlung aller Aspekte des Softwaretestens.

1.2 Hauptbestandteile von explorativem Testen

Eine der am häufigsten zitierten Definitionen des explorativen Testens stammt aus der von James Bach 2003 herausgebrachten Veröffentlichung *Exploratory Testing Explained* [Bac03]. James schrieb: »Exploratives Testen ist Lernen, Testfallerstellung und Testausführung gleichzeitig.«

Dieser Teststil benötigt jederzeit Ihre volle Aufmerksamkeit. Das ist in Cem Kaners Definition des Begriffs offensichtlich:

> »Exploratives Softwaretesten ist ein Stil des Softwaretestens, der die persönliche Freiheit und Verantwortlichkeit des einzelnen Testers hervorhebt, um kontinuierlich den Wert der eigenen Arbeit zu steigern, indem testbezogenes Lernen, Testdesign, Testausführung und Interpretation des Testergebnisses als sich gegenseitig unterstützende Tätigkeiten aufgefasst werden, die während des ganzen Projektes parallel laufen.«[1]

Ich nutze gern eine Abwandlung von James' Definition, um die Praktik zu erklären, indem ich eine Sache ergänze. Die Definition von explorativem Testen, die ich verwende, lautet:

> Gleichzeitiges Erstellen und Durchführen von Tests, um über das System zu lernen, indem Erkenntnisse aus dem letzten Experiment genutzt werden, um das nächste anzuregen.

Jeder Teil dieser Definition ist wichtig: Tests erstellen, Tests durchführen, lernen und steuern. Lassen Sie uns einen Blick auf jeden dieser Aspekte im Detail werfen.

1.2.1 Tests erstellen

Bei der Testerstellung werden Sie Interessantes erkennen, das Sie variieren können, und auch Wege finden, es zu variieren. Es gibt bereits ein reiches Literaturangebot zu diesem Thema, u.a. die Klassiker wie Glenford Myers *The Art of Software Testing* [Mye79] und Boris Beizers *Software Testing Techniques* [Bei90]

1. *http://kaner.com/?p=46*

sowie Lee Copelands neueren und umfassenden Überblick *A Practitioner's Guide to Software Test Design* [Cop04]. Diese Bücher decken Techniken wie Grenzwertanalyse, Entscheidungstabellen und Ursache-Wirkungs-Diagramme genauso ab wie das Ableiten von Tests aus Designmodellen wie z.B. Zustandsdiagrammen, Sequenzdiagrammen und Flussdiagrammen.

All diese Testdesigntechniken sind auch beim Forschen noch relevant. Je mehr Sie sich mit Testdesign auskennen, desto besser werden Sie darin, spontan gute Experimente zu erstellen.

1.2.2 Durchführen

Wenn Sie forschen, führen Sie einen Test schon durch, sobald Sie an ihn denken. Das ist eine der Kerneigenschaften, die exploratives Testen von Testskripts unterscheidet. Die Direktheit der Durchführung unterscheidet das Forschen von anderen Testansätzen. Sie erstellen nicht erst alle Tests, bevor Sie sie durchführen, sondern Sie fangen sofort mit der Durchführung an. Das ist entscheidend: Solange Sie Ihren Test nicht durchführen, wissen Sie nicht, welche Fragestellung es danach zu erforschen gilt. Sofortige Durchführung ermöglicht es, Ihre Nachforschungen in Richtung der interessantesten Informationen zu lenken.

1.2.3 Lernen

Während Sie die Software erkunden, kommen Sie dahinter, wie sie funktioniert. Sie werden ihre Marotten und Eigenarten erfahren. Sie halten sorgfältig Ausschau nach den kleinsten Hinweisen, wo sich ein Nest aus Bugs versteckt halten könnte. Wachsamkeit ist unerlässlich: Je besser Sie beobachten, desto mehr werden Sie lernen. Und es ist schwieriger, als es klingt. Sie müssen ignorieren, was Sie erwarten oder zu sehen hoffen, um zu erkennen, was wirklich passiert (Kapitel 3, *Details beobachten,* stellt Hinweise zur Ausbildung Ihrer Beobachtungsfähigkeiten bereit).

1.2.4 Steuern

Mit jedem Experiment, das Sie ausführen, gewinnen Sie etwas mehr Einblick in das Verhalten der Software. Sie werden merken, welche Bedingungen die Software nicht gut verarbeiten kann, und nutzen dieses Wissen, um noch stärker nachzuforschen. Sie nutzen Ihre Neugier, angetrieben durch das, was Sie bisher gelernt haben, um das nächste interessante Informationsstück zum Freilegen vorzuschlagen. Steuern, während man sich auf die wichtigste zu entdeckende Information konzentriert, ist eine der Hauptfähigkeiten eines guten Forschers.

1.3 In zeitbeschränkten Sessions arbeiten

Software zu erkunden kann ein Unterfangen mit völlig offenem Zeithorizont sein. Ohne einen Mechanismus, um Ihre Bemühungen zu strukturieren und zu organisieren, können Sie Stunden und Tage damit verbringen, ziellos durch die Software zu irren, ohne interessante oder hilfreiche Informationen zu erlangen.

Als Antwort auf dieses Problem erfanden Jon Bach und James Bach das sessionbasierte Testmanagement (engl.: Session-Based Test Management, SBTM)[1]. Dabei strukturieren Sie Ihre Zeit in einer Reihe zeitbeschränkter Sessions. Sie entwickeln schon im Voraus einen Schwerpunkt für Ihre Session (Kapitel 2, *Ihre Forschungen chartern*, zeigt, wie man diesen Schwerpunkt setzt und für sich nutzt). Während der Sessions erforschen Sie fortlaufend die Software, erstellen Tests und führen sie durch: Sie bewegen sich ohne Pause von einem Experiment zum nächsten.

Während jeder Session machen Sie sich Notizen, damit Sie wissen, was Sie untersucht haben und welche Informationen Sie gefunden haben. Sie sollten diese Notizen aber auch weiter nutzen. Sie werden sich auf sie beziehen, wenn Sie mit Stakeholdern eine Nachbesprechung durchführen, aber sie sind nicht wie herkömmliche Testfälle oder Testreports. Ihre unbearbeiteten Notizen werden anderen kaum weiterhelfen. Sie werden Notizen zu Testideen, Fragen, Risiken, Überraschungen, zusätzlichen Bereichen, die es zu erforschen gilt, und zu Fehlern machen.

Am Ende der Session halten Sie die Informationen fest, um sie anderen mitzuteilen. Sie könnten Ihre Beobachtungen über Fähigkeiten und Grenzen, die Sie untersucht haben, schriftlich festhalten oder Sie könnten sich mit den Stakeholdern zusammensetzen, um Ihnen persönlich zu erzählen, was Sie gefunden haben. Wenn Sie Fehler entdeckt haben, die gemeldet werden müssen, dann melden Sie diese Fehler. Wenn Sie Fragen haben, suchen Sie sich jemanden, der sie beantworten kann. Die Sessions stellen Ihnen regelmäßige Ausstiegspunkte bereit, um Ihre Funde zu verarbeiten und den Bereich zu identifizieren, der sich am besten für die nächste Erkundung eignet.

1.4 Für die Praxis

Ein Kernthema dieses Kapitels sind der Unterschied zwischen Prüfen und Forschen und die Idee, dass eine umfassende Teststrategie beides benötigt.

Nehmen Sie sich einen Moment Zeit, um Ihre jetzige Teststrategie zu überdenken. Beginnen Sie damit, eine Liste der Fragen niederzuschreiben, von denen Sie erwarten, dass sie durch Testaktivitäten beantwortet werden. Sie könnten sehr allgemeine Fragen aufschreiben, wie zum Beispiel:

1. *http://www.satisfice.com/articles/sbtm.pdf*

> Kann ein Nutzer die Software wirklich für den geplanten Zweck einsetzen? Funktionieren die grundlegenden Handlungsabläufe?

Sie könnten auch spezifische Fragen zu den Fähigkeiten oder Interaktionen haben wie beispielsweise:

> Wie spielt die Rabattfunktion mit der Bündelfunktion für gleichartige Produkte zusammen?

Sie könnten Fragen zu übergeordneten Belangen oder Eigenschaften haben wie zum Beispiel:

> Vermeidet die Software unter Überlast, dass Daten verloren gehen oder korrumpiert werden?

Sammeln Sie eine Liste an Fragen. Wenn Ihnen die Ideen ausgehen, schauen Sie sich Ihre Liste nochmals an. Überlegen Sie, wie Ihre jetzige Teststrategie jede dieser Fragen beantwortet. Denken Sie für jeden Eintrag darüber nach, ob die Frage eher durch einen vordefinierten Testfall (Prüfung), durch Erforschung oder durch eine Kombination der Techniken beantwortet wird.

Wenn Sie herausfinden, dass Sie für alle Einträge nur eine der beiden Techniken, entweder Prüfen oder Erforschen benötigen, dann denken Sie sich Fragen aus, die nur mit der anderen Technik beantwortet werden können.

Wenn Ihre Fragen zum Beispiel nur die Technik des Erforschens nahelegen, dann sollten Sie über Fragen nachdenken, die sich auf die Grundfunktionen beziehen, die immer funktionieren sollten.

Wenn Ihre Fragen sich jedoch alle durch vordefinierte Prüfungen beantworten lassen, dann sollten Sie diejenigen Risikoarten betrachten, die Sie nur durch aktive Nachforschungen finden würden: ungeplante Folgeerscheinungen und Seiteneffekte, überraschende Interaktionen oder nicht geplante Anwendungsszenarios (wenn Sie immer noch Probleme damit haben, sich Fälle vorzustellen, wo sich Forschen mehr lohnt, sollten Sie die anderen Kapitel dieses Buchteils lesen und danach zu dieser Übung zurückkehren).