CEM KANER
JAMES BACH
BRET PETTICHORD

Lessons Learned in SOFTWARE TESTING

A Context-Driven Approach

"Pick up this book, open it anywhere, read it for two minutes, and take one lesson as a suggestion. Your testing, test planning, test management, or thinking about testing will improve dramatically."

Johanna Rothman, Rothman Consulting Group, Inc.

"If you test software, or depend on people who do, then read this book. Each page bubbles with hard-won advice for handling the practical problems you encounter every day."

Sam Guckenheimer
Senior Director of Automated Testing Technology
Rational Software Corporation

"Definitely a book worth reading and keeping around. Smart, practical, insightful and thought-provoking."

Ross Collard, Collard & Company

"These three distinguished test professionals have written a precisely-stated and thought-provoking book that offers a distinctive and important perspective on testing and test project management."

Rex Black, Author of Managing the Testing Process
and Critical Testing Processes

"This is the book the testing community has been looking for and didn't realize it. A must read for any test engineer or manager."

George Hamblen Jr., Director of Software Quality Assurance
for a large financial services company

"This isn't textbook stuff. It's better. It's real life under discussion and observation. I'm excited to see so many aspects of testing being brought together into one book. I expect great discussions to be had because of this book."

Steve Tolman, Manager of Software Quality, PowerQuest

"These lessons contain wonderful insights about software testing in the real world, from the leading practical experts on software testing. Whether you test software, or work with people who do, this book is great stuff."

Alan Myrvold

"Clear and succinct. It has brought clarity to many of my own learning experiences and provoked a lot of new thoughts."

Fran McKain
Software Test Manager
Hewlett-Packard Company

"Reviewing this book was one of my greatest learning experiences. I warmly recommend this as a must-have for any testing professional."

Hans Buwalda, Author of Integrated Test Design and Automation

"The book is packed with nuggets of gold derived from years of practical experience. The chapter on test automation alone is more useful than any of the books I've seen on test automation. The chapter on techniques has powerful ideas, simply stated!"

Doug Hoffman, Consultant, Software Quality Methods, LLC

"*Lessons Learned in Software Testing* is a must read for the beginner who needs tried and true tips and for the mature test manager who is looking for more refinements for his or her organization."

Chris DeNardis, Supervisor of Software Engineering, Rockwell Automation

" . . . offers an invaluable collection of real world practices based on years of experience shared by the authors collectively and many of their colleagues . . . an absolute must for anyone who has a serious interest in software testing."

Hung Q. Nguyen, President and CEO, LogiGear Corporation
Author of Testing Applications on the Web

"The lessons format is simple and succinct, just the thing for us to use in late night test planning sessions. Where other books have been long on theory and are great for study, this is long on reality, practicality, and immediate usefulness."

Mary Romero Sweeney, Author of Visual Basic for Testers

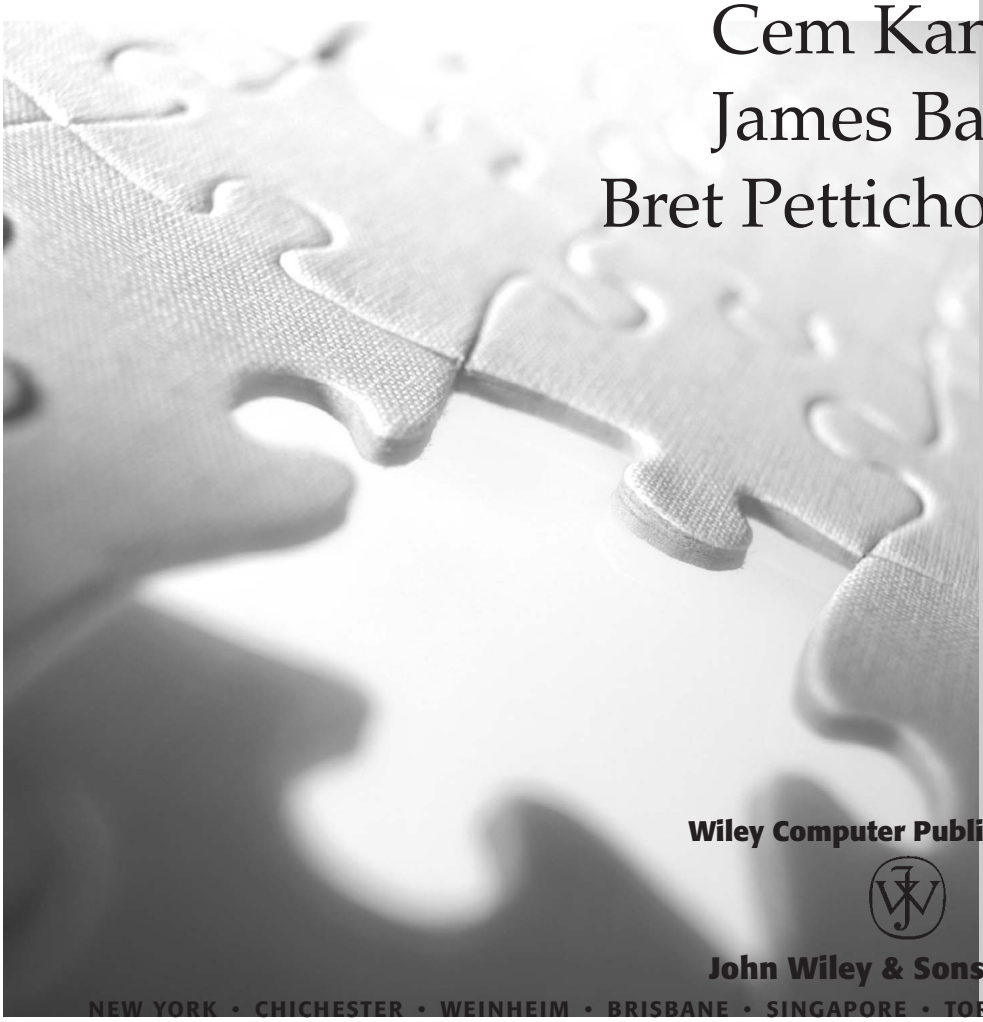
"This is an excellent book. I have had similar experiences as documented in this book, without being able to learn the lessons whilst in the middle of the problem."

Ståle Amland, Amland Consulting, Norway

Lessons Learned in Software Testing

A Context-Driven Approach

Cem Kaner
James Bach
Bret Pettichord



Wiley Computer Publishing

John Wiley & Sons, Inc.

NEW YORK • CHICHESTER • WEINHEIM • BRISBANE • SINGAPORE • TORONTO

Publisher: Robert Ipsen
Editor: Margaret Eldridge
Assistant Editor: Adaobi Obi
Managing Editor: Micheline Frederick
Text Design & Composition: D&G Limited, LLC

This book is printed on acid-free paper. 

Published by John Wiley & Sons, Inc., New York.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Kaner, Cem.
Lessons learned in software testing : a context-driven approach / Cem Kaner, James Bach, Bret Pettichord.
p. cm.
"Wiley Computer Publishing."
Includes bibliographical references and index.
ISBN 0-471-08112-4 (pbk. : alk. paper)
1. Computer software—Testing. I. Bach, James. II. Pettichord, Bret. III. Title.

QA76.76.T48 K34 2001
005.1'4—dc21 2001046886

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

DEDICATION

To Brian Marick and Sam Guckenheimer, who set the spark for this book.

To Dave Gelperin, who believed in us and built a community.

To Jerry Weinberg, whose life and work embodies the highest ideals of an expert tester.

In memoriam, Anna Allison, colleague and friend, September 30, 1952–September 11, 2001.

CONTENTS

Chapter 1 The Role of the Tester

Chapter 2 Thinking Like a Tester

Chapter 7 Interacting with Programmers

Chapter 8 Managing the Testing Project

FOREWORD

Imagine that you are holding a bottle of 50-year-old port. There is a way to drink port. It is not the only way, but most folks who have enjoyed port for many years have found some guidelines that help them maximize their port-drinking experience. Here are just a few:

Lesson 1: Don't drink straight from the bottle. If you have no glass, nor any other possible containers available, pour a tiny amount of port into the palm of your hand and sip from there. The port aroma should be smelled as you sip. Let the port swirl over your tongue. Don't gulp down the port.

Lesson 2: Don't drink the entire bottle. If you are drinking because you are thirsty, put down the port and drink a big glass of water. A small amount of port each time maximizes the enjoyment of the entire bottle.

Lesson 3: Don't pollute the port. If someone tells you that you ought to try a new cocktail made with orange juice, seawater, and port, politely refuse. With a big smile, say, "But I would enjoy a glass of port."

Lesson 4: Don't covet the port. Hoarding your port means that you will never have the pleasure of gentle conversation while sipping. Port is best shared with friends who also enjoy a glass. Remember, they have a bottle somewhere, too.

You are not holding a bottle of port in your hands. You are holding *Lessons Learned in Software Testing*, a very valuable book about software testing. This book has been ripening to perfection throughout the 50 work-years of the authors' experiences. While port is for your taste buds, this book is for your brain. I think that you will find any other differences to be marginal. I have

savored this book and offer you this advice to maximize your reading experience.

Lesson 1. Don't drink straight from the bottle. Bring your own vessel to the reading of this book. That is, bring all of your experiences in software development and testing. If you have never participated in a serious software effort, this book will be too heady for you. It will leave you confused and unable to operate heavy machinery for some time. If you are experienced, savor the book's contents in the context of your own projects.

Lesson 2. Don't drink the entire bottle. Don't read this book in one sitting. Read a lesson or two, close the book, and decide how you respond to the words of Messrs. Kaner, Bach, and Pettichord. As you will discover, they call their approach to testing "context-driven." Only you know the context of your own work. You must determine where there is a fit between a given lesson and your particular work.

Lesson 3. Don't pollute the port. Somebody out there is going to make a list of the titles of the 293 lessons contained in this book. Please don't let it be you. The heart of the book is the explanation attached to each of the lessons. Be warned as well that someone else will immediately try to ISO-ify or CMM-ummify the contents. I can see the article title now, "Getting to CMM Level 293 using the 293 Lessons of *Lessons Learned in Software Testing*." Argh! As the authors explain, " . . . we don't believe in 'best practices.' We believe some practices are more useful than others under some circumstances." Written by pros, those statements represent the quintessence of attaining the state of Master-crafter of Software Testing.

Lesson 4. Don't covet the port. If there ever is a book to be read with your colleagues, this is it. Buy a carton-load and hand a copy out to everyone who tests and to everyone who thinks that she or he manages testers. Read a few selected lessons at a time and get together to talk about them over coffee, lunch, even port! Read, ruminate, enjoy. Cheers!

Tim Lister
August 17, 2001
lister@acm.org
The Atlantic Systems Guild, Inc.
New York City

PREFACE

The *Software Engineering Body of Knowledge* (SWEBOK) is being proposed as a suitable foundation for government licensing, for the regulation of software engineers, and for the development of university curricula in software engineering. The SWEBOK document claims to be consensus-based. One would expect such a document to carry the accumulated knowledge and wisdom (the accumulated *lessons*) of the field.

Here is everything that SWEBOK has to say about exploratory testing:

> Perhaps the most widely practiced technique remains ad hoc testing: tests are derived relying on the tester skill and intuition ("exploratory" testing), and on his/her experience with similar programs. While a more systematic approach is advised, ad hoc testing might be useful (but only if the tester is really expert!) to identify special tests, not easily "captured" by formalized techniques. Moreover it must be reminded that this technique may yield largely varying degrees of effectiveness. (SWEBOK 0.95, 2001, 5-9)

How does SWEBOK treat what it acknowledges is the field's most widely practiced technique? Nothing about how to practice the technique well. Only a statement that exploration should be done only by *real experts*, that other approaches are advised, and the suggestion that other, formalized techniques will yield less varying degrees of effectiveness.

Hah!

We *don't* pretend to offer a consensus document that describes our field's alleged *body of knowledge*, but *we do have a lot more* to say about our field's most common practices. Rather than dismissing exploratory testing, this book shows you what testing looks like through the eyes of people who use exploratory methods (and many other methods) in a drive to achieve excellent testing under real-life conditions.

Welcome to *Lessons Learned in Software Testing*

This book is about software development as we've experienced it. We've had a combined total of 50 to 60 years of development experience (depending on how you count). We've been at this for a while. We've seen a lot of great work and a lot of not-so-great work.

This book is not about how software engineering might be in a tidier and more controlled world. We're writing about the world as we've worked in it.

In our world, software development teams often work under ambitious deadlines, discovering what needs to be done at the same time as they're discovering how to do it. Sometimes their approaches are more formal, sometimes less. It depends on a wide range of circumstances.

We follow the *context-driven approach* in software testing. We expect that a method that works wonderfully under some circumstances will not work under others. Rather than talk about best practices, we talk about practices that are well-suited to the context at hand. We discuss the context-driven approach at the end of the book, but in essence, context-driven testing looks at the "what" of testing (the techniques, tools, strategies, and so on) in terms of "who," "when," "where," "why," and "what if."

Our goal is to match our selection of practices to the circumstances at hand in a way that achieves excellent testing. We don't expect to achieve great testing by taking over the project, nor by stamping our feet and telling the project manager (or executives) how Real Professionals would run the project. We don't expect to achieve great testing by intimidating programmers, nor by being obsequious to them. We don't expect to do great testing by filling out thousands of little pieces of paper (or comparable electronic records), nor by wasting everyone else's time on unnecessary processes.

We don't have a political or a bureaucratic or a formal-methods recipe for great testing.

That recipe doesn't exist!

We think great testing involves skillful technical work (searching for defects) and accurate, persuasive communication.

The skilled search is always exploratory. There's an infinite amount of testing to do, and a tiny amount of time in which to do just a little bit of that infinity. Every test we do, every document we write, every meeting we attend takes time away from running other tests that could expose a key defect. Facing that constraint, we optimize our testing processes so that they take advantage of our ever-growing knowledge of the product, its market, its applications, and its weaknesses. What we learn today gets reflected in more powerful tests tomorrow.

Even if:

- the product is well specified, and
- the specification accurately reflects a requirements document, and
- the requirements document accurately represents the actual needs of the product's stakeholders

(have you *ever* been on a project where all these "even if's" were true?); we will still learn a lot about how to test the product while we are testing it. In particular, as we discover errors, we learn how this group of programmers can go wrong. The specifications tell us about how the program is supposed to work when it is coded correctly. They don't tell us what mistakes to anticipate, nor how to design tests to find them. At that task, our key task, we get better from the start to the end of the project, on every project.

No matter what it looks like from the outside, whenever we are testing with our brains engaged, our work is exploratory.

Who This Book Is For

This book is for anyone who tests software, anyone who manages testers, and anyone who has to deal with testers in their software development projects. That includes project managers and executives.

The primary reader that we have in mind, the "you" in this book, has been testing for a few years and might recently have been promoted to a supervisory role. We hope that you'll see a lot in this book that matches your experience, that you'll gain new insights from our lessons, that you'll find lessons that are useful for quoting to your manager, that you'll like a few of our statements so much that you'll photocopy them and stick them on the

outside of your cubicle, and maybe that you'll react to at least one statement so strongly that you'll stick a copy on the middle of your dart board. (We want to stimulate thought, not only agreement.)

Newcomers to testing (and those of you who are just applying for a testing job) won't have as many occasions to feel that you've already experienced what we're writing about. For you, this book might provide some early insights and warnings, giving you a good taste of what issues testers face.

Hint: If you are absolutely new to testing and you are looking for a book to study from in order to prepare for a job interview, this is the wrong book. If this is the only book available to you, pay your most careful attention to the chapters on "Testing Techniques" and "Your Career in Software Testing." If you have a choice of books, we suggest the first five chapters of *Testing Computer Software* (Kaner *et al.,* 1993).

Programmers, project managers, and executives who have to work with testers will find this book a useful source of ideas for setting your expectations of the testing group. We hope that it will help you evaluate and discuss your concerns with the testing group if you disagree with their policies or feel that they are spending their time unwisely.

What This Book Is About

Over the years, we've learned many useful practices and helpful ways of evaluating situations. Our conclusions are based on experience. We summarize much of our experience in this book, in a series of short, readable descriptions of a few hundred *lessons*.

We adopted several criteria for including a lesson in this book. They are:

- The lesson should be useful or should provide insight.
- The lesson should pass the 90-minutes-thinking-alone test. A lesson is not worth including in the book if almost anyone who thought about testing for 90 minutes of undistracted time could come up with it.
- The lesson must be based on our actual experience. At least one of us (preferably all three of us) had to have successfully applied the advice we give. At least two of us had to have been burned by attempting to follow a practice we criticize. (*Note:* Sometimes, we come to different conclusions, based on different experiences. Occasionally, you'll see that we choose to provide two viewpoints rather than one. Even if only one view is presented, you cannot assume that all three of us fully agree with it—in the event of disagreement, we are likely, among the three of us, to defer to the one or two who have the most extensive experience with a given situation.)

- Lessons should be tempered by the experiences of our colleagues. We've been collecting detailed experience reports at the Los Altos Workshops on Software Testing, the Software Test Managers Round Tables, the Workshops on Heuristic and Exploratory Techniques, the Austin Workshops on Test Automation, the Patterns of Software Testing Workshops, the Workshop on Model-Based Automated Testing, Systems Effectiveness Management Groups, at dozens of software testing conferences, and at less formal peer cross-trainings (such as the annual Consultants' Camps in Crested Butte).
- The lesson should be brief and to the point but easily understood.
- A lesson can be longer, but only to the extent needed to explain how to do something or to provide a useful tool. Long descriptions and detailed background information are for the textbooks.
- Lessons should be self-contained. You should be able to start reading anywhere in the book.
- The collection of lessons should give you a feel for how we do and think about testing.

What This Book Is Not

This book is not a comprehensive guide to software testing.

This book is not a collection of lessons that are always true. These are *our* lessons, based on *our* experiences. We believe these are broadly applicable, but some lessons that have been useful and important in our careers may not work for you. You have to use your own judgment. As a particular limiter on the generality of this work, we should note that we've worked more on the development of software for the mass market and on contract-specified software than we have on software developed for in-house use. Our experience with life-critical software and embedded software is limited.

This book is not a collection of best practices. In fact, we don't believe in "best practices." We believe some practices are more useful than others under some circumstances. We are concerned that many things sold as best practices are pushed (and applied) uncritically, to situations that aren't appropriate to them.

How to Use This Book

We structured this book to make it easy for you to skim it or flip through it, rather than trying to read it from start to finish. At some point (we hope), you'll find a nugget, an idea that is very appealing to you. We cannot

recommend that you apply this idea uncritically. (Our lessons are not best practices.) Instead, we urge you to evaluate the lesson in terms of its appropriateness to your situation.

Here are some questions that might help you perform that evaluation:

- Under what circumstances would an application of this lesson work in your company?
- Under what circumstances would the application not work?
- Has anyone you know tried anything like this lesson before? What happened? Why? What's different about your current project from that person's project? Should that difference matter?
- Who is most likely to benefit from the attempt to apply this lesson?
- Who is most likely to be disadvantaged as a result of an attempt to apply the lesson?
- What will you learn from attempting to apply this lesson?
- Trying anything new adds risk. Does it make sense to see how well this lesson would apply to your company by running a pilot study, a situation in which you try out the lesson without committing yourself to it, or under a situation of low risk? Is it feasible to conduct a pilot study in your company, on (or in the service of) your current project?
- How will you know whether your application of this lesson worked? If it did work, how will you know whether the success came more from the intrinsic value of the lesson, which will last as you keep applying it, than from your enthusiasm in trying out the idea?
- What are the best and worst things that could happen to you as a result of trying to apply this lesson?
- What are the best and worst things that could happen to another stakeholder, such as a user or another member of the development team?
- What if a key person in your company disagrees with a lesson that you want to follow? How will you overcome their objections and then sell it to them?

We Hope This Book Stimulates Conversations and Debates

This book draws sharp contrasts between our views and some others. We think that clarity will help fuel debates that the field should be having. We don't believe there is an accepted paradigm in software testing or in software engineering as a whole. That's why we're not among the folks pushing

government licensing and regulation of software engineers after standardization on a *body of knowledge*. In our experience, there are remarkably different, credible views as to the best ways to proceed.

We want to be very clear about this. We often critique the work of people we respect highly. In many cases, we refer to work done by good friends of ours. Don't mistake an attack on an idea with an attack on the proponent or clearest author of that idea.

We think the field will benefit by head-on comparisons and contrasts of those views. It's important for our field to have conversations about our methods. We advocate the further development of skilled practices within every one of the main approaches. Eventually, we will all learn the circumstances under which the different approaches are the best. Until then, let the thousand flowers bloom.

A Few Notes on Vocabulary

Here are some of the key words in this book, and how we use them:

We. The authors.

You. The reader.

A *fault* is an error, a mistake in the implementation or design of the program.

As we use the term, an *error* is a fault.

A *failure* is the misbehavior of the program, resulting from the program encountering a fault.

Failures occur under *conditions*. For example, a program will crash when it attempts to divide by zero. The fault in this case is in the code that allows division by zero. The failure is the crash. But you don't see a failure unless the critical variable in the division has the value zero. That variable, having a value of zero, is the critical condition that must be met for the failure to occur.

A *symptom* is like a failure but less serious. For example, if the program has a memory leak, it might start slowing down long before it fails with an out-of-memory error message. The slowdown is a symptom of an underlying problem (memory shortage) that is not directly visible.

The word *bug* is a catch-all term. It could refer to anything wrong with the software. Someone reporting a bug might describe a fault, a failure, or a limitation of the program that makes it less valuable to a stakeholder.

If we define *quality* as value to some person (Weinberg, 1998, 2.i.), a *bug report* is a statement that some person considers the product less valuable because of the thing being described as a bug.

The word *defect* carries a legal charge. It means Something Is Definitely Wrong With The Product. Some companies don't allow the words *defect* or *defective* to appear in their bug tracking systems or bug-related memos.

Some companies prefer to say *anomaly, problem,* or *issue* instead of *bug*.

After you report a bug, the programmers (or the Change Control Board) will fix it or decide not to fix it. They will mark the bug *resolved* (fixed, deferred, not reproducible, works as designed, and so on).

Black box testing. Testing the external behavior of the program, by feeding the program inputs and examining the outputs. In typical black box testing, the tester has no knowledge of the internals of the code and is instead more familiar with the explicit and implicit product requirements, such as the ways the program will be used, the types of data likely to be fed to the program, any regulatory issues associated with the problem the software is attempting to solve or help solve, and the hardware and software environments in which the software will run.

Behavioral testing. Testing the external behavior of the program, similar to black box testing, but using as much knowledge of the internals of the program as is available to the tester and relevant to the test.

Functional testing. Black box or behavioral testing.

White box or glass box testing. Testing with knowledge of the internals of the program.

Structural testing. White box testing that is focused on the internal structure of the program, such as the flow of control from one decision or action to the next.

Smoke testing or *build verification testing.* A standard suite of tests applied to a new build. The tests look for fundamental instability or key things missing or broken. If the build fails these tests, you do not test it any further. Instead, you keep testing the old build or wait for the next one.

Project manager. The person who is accountable for shipping the right product on time and within budget. Some companies split the work that we ascribe to the project manager between a program manager and a development manager.

MaxInt. The largest integer possible on the user's platform or the programmer's development language. A number larger than MaxInt cannot be stored as an integer.

Client. Someone whose interests it is your job to serve. This probably includes everyone on the project to some extent, as well as the ultimate users of the product.

ACKNOWLEDGMENTS

This book would not have been possible without the support and assistance of a great many people. We thank Lenore Bach, Jon Bach, Becky Fiedler, Leslie Smart, and Zach Pettichord for support, understanding, and assistance while three frantic men thought of nothing but their book. We thank Pat McGee for providing research assistance at a crucial time.

We benefited from detailed and thoughtful reviews of early drafts. We've added several of our reviewers' examples and descriptions of alternative perspectives to the book. We thank Ståle Amland, Rex Black, Jeffrey Bleiberg, Hans Buwalda, Ross Collard, Lisa Crispin, Chris DeNardis, Marge Farrell, Dorothy Graham, Erick Griffin, Rocky Grober, Sam Guckenheimer, George Hamblen, Elisabeth Hendrickson, Doug Hoffman, Kathy Iberle, Bob Johnson, Karen Johnson, Ginny Kaner, Barton Layne, Pat McGee, Fran McKain, Pat McQuaid, Brian Marick, Alan Myrvold, Hung Nguyen, Noel Nyman, Erik Petersen, Johanna Rothman, Jane Stepak, Melora Svoboda, Mary Romero Sweeney, Paul Szymkowiak, Andy Tinkham, Steve Tolman, and Tamar Yaron.

This book has benefited enormously from numerous discussions we've had at the Los Altos Workshops on Test Automation, the Workshops on Heuristic & Exploratory Techniques, the Software Test Managers' Roundtable, the Austin Workshops on Test Automation, the Patterns of Software Testing Workshops, and at many other workshops, conferences, classes, and worksites with so many people who have put their hearts into finding better ways to test software. We thank you all.