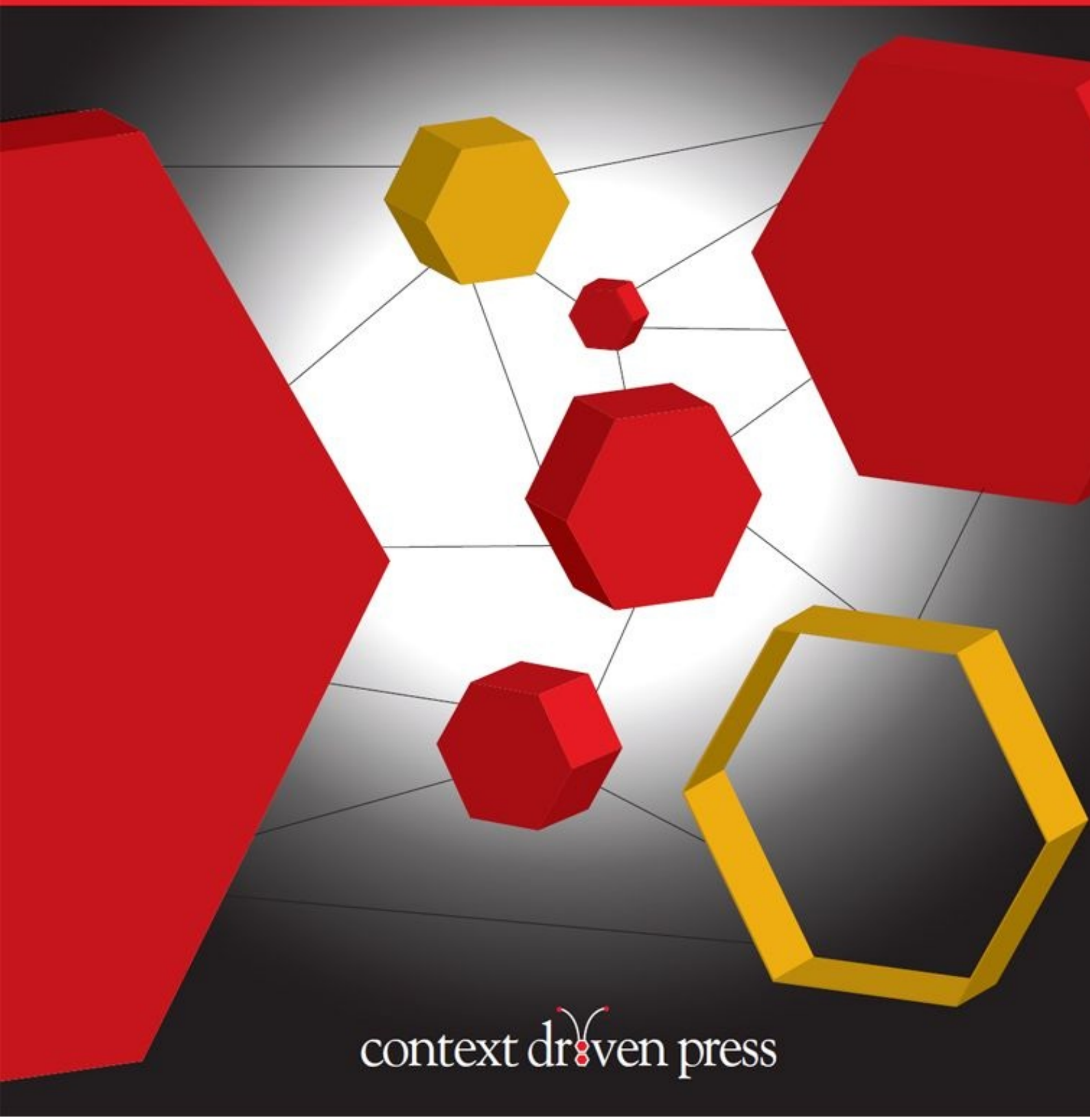
THE DOMAIN TESTING WORKBOOK

CEM KANER • SOWMYA PADMANABHAN • DOUGLAS HOFFMAN



context driven press

THE DOMAIN TESTING WORKBOOK

CEM KANER • SOWMYA PADMANABHAN • DOUGLAS HOFFMAN

context driven press

Library of Congress Cataloging-in-Publication Data:

Kaner, Cem; Padmanabhan, Sowmya; & Hoffman, Douglas

The Domain Testing Workbook

Library of Congress Control Number: 2013947216

ISBN 978-0-9898119-1-0

The Context-Driven Press logo and The Domain Testing Workbook cover design are by Susan Handman, Handman Design, Ne York.

Editor: Rebecca L. Fiedler

Copy Editor: Karen Fioravanti

These materials are partially based on research that was supported by National Science Foundation research grants EIA-01135 ITR/SY+PE: Improving the Education of Software Testers and CCLI-0717613 Adaptation & Implementation of an Activity-Base Online or Hybrid Course in Software Testing. Any opinions, findings and conclusions or recommendations expressed in this mater are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

A SCHEMA FOR DOMAIN TESTING: AN OVERVIEW ON ONE PAGE

Here is a list of several tasks that people often do as part of a domain testing analysis. We organized the book's chapters around this list because it puts the tasks into a logical order.

Please note that for any particular product or variable, you might skip several of these tasks or do them in a different order than we list here.

1. CHARACTERIZE THE VARIABLE

A. Identify the potentially interesting variables.

B. Identify the variable(s) you can analyze now. This is the variable(s) of interest.

C. Determine the primary dimension of the variable of interest.

D. Determine the type and scale of the variable's primary dimension and what values it can take.

E. Determine whether you can order the variable's values (from smallest to largest).

F. Determine whether this is an input variable or a result.

G. Determine how the program uses this variable.

H. Determine whether other variables are related to this one.

2. ANALYZE THE VARIABLE AND CREATE TESTS

I. Partition the variable (its primary dimension).

– If the dimension is ordered, determine its sub-ranges and transition points.

– If the dimension is not ordered, base partitioning on similarity.

J. Lay out the analysis in a classical boundary/equivalence table. Identify best representatives.

K. Create tests for the consequences of the data entered, not just the input filter.

L. Identify secondary dimensions. Analyze them in the classical way.

M. Summarize your analysis with a risk/equivalence table.

3. GENERALIZE TO MULTIDIMENSIONAL VARIABLES

N. Analyze independent variables that should be tested together.

O. Analyze variables that hold results.

P. Analyze non-independent variables. Deal with relationships and constraints.

4. PREPARE FOR ADDITIONAL TESTING

Q. Identify and list unanalyzed variables. Gather information for later analysis.

R. Imagine and document risks that don't necessarily map to an obvious dimension.

DEDICATION

This book is dedicated to my step-mother, Rosemary Kaner, and the spirit of my father, Harry, who raised me in their businesses and taught me to cherish integrity, skepticism, the applicability of mathematics in all situations and the transcendant value of the Person in each of them, of the fundamental importance of data, and the delight in well-designed tables.

— Cem Kan

To my children who are my inspiration and motivation in life: Ria, my wonderful daughter, and Simb my awesome dog.

— Sowmya Padmanabh

I'd like to dedicate my work to the ladies in my life, Connie and Jackie who have stoically endured m sometimes long absences, many of which were for collaboration on this book.

— Doug Hoffm

CONTENTS

REFERENCES

DOMAIN TESTING TABLES

PREFACE

People learn what they do. To develop skills, people need to practice. To practice, people need examples to practice on, time to work on them, and feedback.

This book is about a single software testing technique, *domain testing.* You might know it as *equivalence class analysis* or *boundary testing.* It is our field's most widely-taught technique.

TESTING TECHNIQUES

Testing is a cognitively complex activity. Developing competence with cognitively complex skills requires mastery of routine tasks and formation of schemas (cognitive maps) that can guid you as you do tasks that require more conscious effort (van Merrienboer, 1997).

The fundamental problem underlying testing's complexity is that every tester, of every nontrivia program, must choose from an impossibly large set of potential tests. Test techniques provide a cognitive toolkit for making these choices.

A test technique is both, a design tool and a selection tool:

- As a design tool, it tells you what to include in the test.
- As a selection tool, it provides a method for sampling a relatively small number of interesting tests from the vast set of possibilities.

Domain testing is primarily a sampling strategy:

- Divide the possible values of a variable into subsets of values that are similar in some way (we'll call them *equivalent*).
- Design your tests to use only one or two values from each subset. Pick extreme values (we'l call them *boundaries*) that maximize the likelihood of exposing a bug.

A critical problem with much industrial and academic training is that we teach test techniques a if there were obvious procedures to generate the correct set of tests. There are no such procedures. Instead, each technique involves its own way of thinking; one you get better at over time as you gain experience.

To learn a test technique is to learn a way of thinking about how to test well, not how to follow a procedure.

DOMAIN TESTING AS A TEST TECHNIQUE

According to the domain-testing way of thinking, we focus test designs on the values of variables. We select values for those tests by partitioning variables' values into equivalence classes. We pick values from within those classes that are the most extreme (such as the boundaries) because we're looking for the values most likely to drive the program to failure.

As a sampling strategy, domain testing helps testers:

- Improve their efficiency (testers don't run redundant tests).

- Improve their effectiveness (testers are more likely to find bugs because they design tests to be more powerful).

DOMAIN TESTING IS NOT THE ONLY TECHNIQUE

There are over 100 software testing techniques. Each points to different possible tests. Some are more popular than others, but no technique is "best." The challenge of skilled testing is not just know how to apply techniques but to understand which technique is likely to yield the most useful information at *this* time, with *this* program, on *this* project.

Some test groups rely on a single technique to guide all of their testing. This is a mistake. We focus on only one technique in this book because our goal is to help you develop skill with that technique. But please don't confuse the narrow focus of this book with a suggestion that you can get by with only this technique.

A good technique, well-used, might expose a lot of problems, but relying on that *one* is like restricting your testing to a single corner of a large room. The better goal is to learn many techniques well, along with a deeper understanding of when to use them and how best to combine them in the context of a particular project. This approach requires skill and judgment at many levels. We explore the diversity of test techniques in the BBST® lectures on test design, available at http://www.testingeducation.org/BBST. For books that introduce the field's test techniques, we suggest Ainapure (2007); Black (2007); Craig & Jaskiel (2002); Kaner, Bach & Pettichord (2002); Jorgensen (2008); Myers, Sandler, Badgett & Thomas (2004); and Page, Johnston & Rollison (2009).

WHO THIS BOOK IS FOR

Testers

This book is primarily for software testing practitioners—people who make their living as software testers—and people studying to become testers or trying to understand what testers do in order to work with them or manage them.

- *We expect that you already know a bit about testing.* You have either studied some techniques already or applied them on the job. We're extending your knowledge, not introducing you to the field.

  We don't expect everyone to have the same knowledge, even the same basic knowledge. Therefore, we review many basics of testing as we go. But if you're new to testing, we recommend that you start with a general introduction to testing and return to this book when you have a more general background.

- *We expect you to know some basic facts about programming, but we don't expect you to know how to program.* In our experience, many people who work in this field have little or no experience as programmers. Several testers come from other technical fields (such as customer support) or from the application industry (for example, people who know insurance very well joining a group that tests insurance-related software). We're trying to deepen your skills as system testers.

  To do this, we have to introduce some technical concepts at a level that some readers will find difficult and other readers will have already mastered. Included in this category are

discussions of the nature of data. Domain testing focuses on the values of variables, so you need to know about Integers, Fixed-Point, Floating-Point, Strings, multidimensional Records, and so forth.

- *We expect you to practice on the examples, not to just read them.* A person's preferences for how they're taught vary over time. To succeed with this book, you must approach it with an active mindset. Even though it appears to offer a memorizable structure, what it really offers is a collection of experiences.

You will learn more by trying things for yourself than by reading, listening or watching what someone else has done. You will learn more from explaining your solutions to others than from taking in others' explanations. The more active you are in your learning, the more you will learn and the more deeply you will learn.

INSTRUCTORS AND PROFESSORS

This book provides useful material for courses in black-box testing or test design. In a university course on software testing that is primarily focused on the underlying theory of testing, this book offers a practice-oriented supplement that complements texts like Ammann & Offutt (2008) and Tian (2005).

Please see the Appendix for more on how to use this book in your teaching.

PROGRAMMERS?

Several reviewers have suggested that we create examples that walk through code samples. Some have provided detailed suggestions for analyzing the same problem from an external (black-box) view and from the underlying code.

We like this idea. In a university course on software engineering, we would do this. However, after careful consideration, we decided not to include code samples in this book because we felt it would change the character of the book and chase away potential readers—many of whom are nonprogrammers—studying on their own.

That doesn't mean there's nothing in this book for programmers. Like any book on black-box testing, this book speaks to how people experience the code when they use it. It gives a different perspective on how code can break or be inadequate. And it suggests ways for you to test when you use someone else's code (e.g., library functions) but don't have their source code and aren't allowed to reverse engineer it.

HOW TO USE THIS BOOK

This book comes in three sections that differ in character:

SECTION 1 PROVIDES AN OVERVIEW OF DOMAIN TESTING AND KEY TECHNICAL CONCEPTS

If you're new to domain testing, *skim this. Don't get bogged down in it.* Don't worry about parts you don't understand. Work through the examples. They apply many points made here. Come

back after a few examples and skim it again. At some point, you'll find Section 1 easier to read. You'll probably find it helps you organize what you learn from the examples into a more coherent structure. *There is no need to rush this.*

SECTION 2 WORKS THROUGH THE DOMAIN TESTING SCHEMA

The Schema is a list of 18 tasks that we suggest you use to organize your test designs and testing Each chapter describes one task. Each chapter clarifies its description with worked examples an then asks you to work through some exercises.

- The examples are numbered. Each has its own chapter in Section 3.
- *Don't read the solutions to the exercises before you try them yourself. Even if you know you can't do the exercise, complete as much as you can.* The more you try for yourself, the mor you will learn. If you let us tell you the answer, you will lose the learning experience that comes from figuring out the answer for yourself.

SECTION 3 PROVIDES A COLLECTION OF EXAMPLES

We work through each example, one per chapter, showing how to apply the Schema.

We don't apply every item on the list to every example. We do what we think is useful and skip the rest.

THE EXAMPLES ARE SIMPLE, INCREASING IN COMPLEXITY

Many examples are from other books or courses that presented material well. You might find it instructional to compare our solutions to theirs.

Many examples in this book are artificially simple. We designed examples to bring different Tasks of the Schema into focus. You'll see this as you work through Section 3 (the worked examples). The solution for an example emphasizes some Tasks and deals only briefly with the others. The more advanced tasks call for more complex examples.

Even the simple examples reflect real-world software. When you use domain testing, you focus on variables. Every application has many variables that you can test. It makes sense to test variables individually before testing them in combination. Domain testing suggests ways to do one-variable testing efficiently.

- What we call the classical approach requires almost no contextual information about the variable. If you can identify a variable, you can usually figure out its type and boundaries using a combination of experimentation (testing) and research (such as reading documents and asking people).

Examples of the classical approach, illustrate a straightforward technique you can apply to many types of data. When you test a program, it should be easy to find many variables to test this way.

- The Schema generalizes this to an explicitly risk-based approach. This reflects what we've

seen in the practices of skilled testers. The more you know about the application, the more basis you'll have for imagining risks and designing tests to trigger those risks.

Our design goal for the more complex tasks in the Schema was to create the simplest examples that can help you imagine what questions to ask or where to look for risk-relevant information.

WE DELIBERATELY REPEAT INFORMATION

We hope you'll read the *Introduction to Domain Testing* and skim the *Summaries of Key Technical Concepts*. We expect that many readers will skip through the rest of the book in whatever order feels most useful at the time.

To make it easier for you to read chapters on their own, we deliberately repeat some text. You are particularly likely to notice text from Section 2, used to explain a Task, reappearing in Section 3 to explain an Example.

WE WRITE IN A SPIRAL

A spiral presentation covers the same concepts several times with new information about it or a new application of it, each time. The goal of a spiral presentation is to give you a lot of information about a concept without drowning you in details that you won't need until later.

In this type of writing:

- A concept appears first in a relatively short or simplified form. (In many cases, you'll see th concept first in the *Summaries of Key Technical Concepts*.)
- The concept appears again with details relevant to a particular Task or Example.
- The concept might appear several times with new details each time.
- We often present a concept in detail and refer back to that presentation as needed.
- If a technical topic is presented without enough detail early in the book, check the index. W probably cover it again later.
- If you see a technical topic later in the book and the discussion assumes you know more tha you should, look for a cross-reference to a previous presentation of the topic. If you don't see one, check the index.

DON'T JUST READ THE EXAMPLES—WORK THROUGH THEM YOURSEL

The more actively engaged you're in solving these exercises yourself and teaching solutions to colleagues, the richer your learning will be and the more of that learning you will be able to transfer to real-world tasks.

We recommend that you work through the exercises and examples with a friend.

- *Attempt each exercise on your own.*
- *Explain how you solved it to your friend.*
- *Critique your friend's work.* Be friendly, but be specific. You will learn from this and your friend will learn from this.
- *Only then should you* compare your ideas with the solution.
- If your friend still doesn't get it, *try another example.*

Explaining things involves different cognitive processes than reading about them or doing them. To explain something to someone else, you have to organize your knowledge more effectively and fill in the gaps.

NOTICE THE REFERENCES

We cite the books, papers, and presentations:

- That we relied on in writing this book.
- Or that you might find useful for understanding or applying this material.

The reference format (name, date) can be a little distracting at first. You'll get used to it. It carries information that you won't get as easily in any other way. When you pay attention to wh says what, who agrees or disagrees with them, who cites their work and who ignores it, you'll start to see patterns. These patterns can teach you about the social networks that structure communication in our field. Understanding these networks can give you a much deeper insight into the context and meaning of the papers you read and the talks and classes you attend (Moody 2001; Price, 1965; Small, 1978, 2003; Upham, Rosenkopf & Ungar, 2010).

A field's social networks are the fabric on which the field's data are written.

OTHER TESTING BOOKS AND COURSES

Over the past 30 years, there has been tremendous growth in the number and quality of books an courses on software testing.

BOOKS FOR PRACTITIONERS

As I wrote this note, I skimmed through some of my favorite books that address black-box testing: Beizer (1995), Black (2002), Copeland (2004), Craig & Jaskiel (2002), Graham & Fewster (2012), Kaner, Bach & Pettichord (2002), Kaner, Falk & Nguyen (1993), Koskela (2008 Myers (1979), Nguyen, Johnson & Hackett (2003), Page, Johnston & Rollison (2009), Perry (2006), Perry & Rice (1997), Rainsberger (2004), Weinberg (2008), and Whittaker (2003).

- Each of these has been valuable to practitioners. They explain important testing concepts an controversies, suggest ways to think through testing problems, consider the goals and management of the testing effort.
- These books teach by describing and explaining. They offer excellent examples, but they don't set many exercises for the students or provide feedback to help the students develop

their skills.

BOOKS FOR UNIVERSITY STUDENTS

The last decade has seen a new generation of university textbooks for testing, such as Ammann & Offutt (2008), Jorgensen (2008), and Tian (2005). I like all three of these books, have taught from them, and expect to teach from them again. As you would expect from university texts, these have more exercises than the books for practitioners. The exercises were designed to support the instructional objectives of the books.

These books assume knowledge of programming and discrete mathematics. They work well for students with those strengths. Many of these students would benefit from a supplementary collection of problems that emphasize practical applications.

WHY WRITE AN ENTIRE BOOK ON DOMAIN TESTING?

The main ideas of domain testing are easy to explain. In a lecture, I can teach the key concepts in 15 minutes. There are many good, easy-to-read explanations. Glen Myers (1979) did an excellent job of laying out the basics. Jorgensen (2008) and Kaner, Falk & Nguyen (1993) are two other popular examples of clear introductory presentations.

In my classes, most students can give me a description of domain testing and use it in simple situations after a short explanation with an example or two. *That doesn't mean they understand domain testing or that they can use it on the job.*

The most common problem in training is that students can go to a class, learn a technique, but can't apply it on the job. This problem presents itself in several ways:

- Testers recognize the opportunity to apply a new skill and should be able to use it, but it is harder than they expect. It takes them more time. They make errors. *This illustrates the problem of skill.*
- Testers recognize an opportunity to apply a new skill but it contains elements different from what they've studied. Maybe the type of data are a little different, the context is a little different or the risks are a little more complex. They have trouble extending what they've learned to this new case. *This illustrates the problem of transfer of learning.* We offer additional notes on this in the Appendix.
- Testers face a testing problem and don't recognize it as an opportunity to apply a skill or technique that they've learned. *This also illustrates the problem of transfer of learning.*

What good is training if you can't apply it with enough skill to do your job?

When you work as a tester, it doesn't matter whether you can define a test technique or apply it to a simple textbook example. It doesn't matter whether you can answer multiple-choice test questions about it. What matters is whether you can apply the technique to the software you're trying to test. If you can't do that, you didn't learn what you needed in your training. Too much teaching about testing doesn't give you the practice you need to apply new skills and techniques.

I saw these problems for many years, through the eyes of a test manager, a test-management

consultant, and a commercial trainer.

As a trainer, I started developing collections of examples and exercises for several test techniques. I could use *some* of these materials in general survey courses on software testing, bu these types of courses didn't allow enough time for enough examples or practice.

My models for this work were the *Schaum's Outlines* books and more advanced practice books like Sveshnikov (1968). These books present basic ideas and many worked examples, running from simple to complex, from straightforward applications of theory to more challenging word problems.

I gained experience with these types of books as a student, tutor, and teacher of mathematics. Th books were effective when students created their own solutions for the examples and checked their own work against the book. For those students, the diversity of the collection was importar

My goal in creating *The Domain Testing Workbook* was to collect a set of examples:

- Broad enough for you to learn the scope of the technique.
- Detailed enough for you to develop skill, which I think requires trying something, getting feedback, and trying it again until you can do it well.

At first glance, *The Domain Testing Workbook* looks like a long book. However, I urge you to think of it instead as a collection of worked examples. Work through them at your own pace, getting feedback as you go.

Cem Kaner

August, 2013

ACKNOWLEDGMENTS

We got a lot of help from a lot of people:

Thanks to our many reviewers, who waded through drafts from 2010 through 2013: Scott Allma Dani Almog, Ajay Balamurugadas, Richard Bender, Ajay Bhagwat, Lalitkumar Bhamare, Rex Black, Cole Boehmer, Michael Bolton, Pat Bond, Laurent Bossavit, Paul Carvalho, Ross Collard Adriano Comai, Mike Dedolph, Tom Delmonte, Joe DeMeyer, Rikard Edgren, Jack Falk, Rebec Fiedler, Keith Gallagher, Markus Gärtner, W. Morven Gentleman, Paul Gerrard, Dorothy Graham, Sam Guckenheimer, Michael Hackett, Jon Hagar, Dick Hamlet, Linda Hamm, Parimala Hariprasad, Jean-Anne Harrison, Dan Hoffman, Justin Hunter, Martin Jansson, Karen Johnson, Jesse Johnson, Paul Jorgensen, Adam Jubaer, Nawwar Kabbani, Geordie Keitt, Chris Kenst, Vip Kocher, Jonathan Kohl, Darko Marinov, John McConda, Iain McCowatt, Pat McGee, Grigori Melnick, Zoltán Molnár, Hung Nguyen, Ray Oei, Brian Osman, Jane Owen, Louise Perold, Dale Perry, Erik Petersen, Curtis Pettit, Ron Pihlgren, Dee Ann Pizzica, Meeta Prakash, Maaret Pyhajarvi, BJ Rollison, Rob Sabourin, Huib Schoots, Parimala Shankaralah, Aleksander Simic, Ben Simo, Ajoy Kumar Singha, Michael Stahl, Keith Stobie, Andy Tinkham, Aleksis Tulonen, Peter Walen, and Christin Wiedemann.

Thanks also to students who worked in Florida Tech's Center for Software Testing Education & Research and helped us with projects directly related to the content of this book: Ayuba Audu, Pushparani Bhallamudi, Thomas Bedran, Karishma Bhatia, Nathan Christie, Tim Coulter, Sabri Fay, Ajay Jha, Kishore Kattamuri, Jiahui Liu, Pat McGee, Tauhida Parveen, Amit Singh, Umesh Subramanyan, Andy Tinkham and Giridhar Vijayaraghavan.

This book's approach was informed by our instructional research and course development at Florida Institute of Technology. That work was supported by National Science Foundation research grants *EIA-0113539 ITR/SY+PE: Improving the Education of Software Testers* and *CCLI-0717613 Adaptation & Implementation of an Activity-Based Online or Hybrid Course in Software Testing.* (Any opinions, findings, conclusions or recommendations expressed in this book are ours and do not necessarily reflect the views of the National Science Foundation.)

Much appreciation to *Panera Bread* at 245 Palm Bay Road in West Melbourne, Florida, for providing warm hospitality, unlimited good coffee and a comfortable table where we wrote muc of this book.

Special thanks to BJ Rollison for a Chapter 24 better than anything we could have written, to Pa Bond and Morven Gentleman for freely sharing their knowledge of mathematics and their wisdom about presenting technical ideas to readers who are more interested in the application o the concepts than the theoretical underpinnings, and to Bill Shoaff, Chair of Computer Sciences at Florida Tech, for his ongoing encouragement and support over so many years.