Extracted from:

Testing Elixir

Effective and Robust Testing for Elixir and its Ecosystem

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Testing Elixir

Effective and Robust Testing for Elixir and its Ecosystem

Andrea Leopardi and Jeffrey Matthias

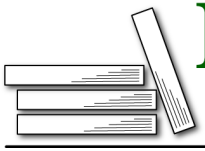
edited by Jacquelyn Carter

Testing Elixir

Effective and Robust Testing for Elixir and its Ecosystem

Andrea Leopardi
Jeffrey Matthias

The Pragmatic Bookshelf
Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Series Editor: Bruce A. Tate
Development Editor: Jacquelyn Carter
Copy Editor: Molly McBeath
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-782-9
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—July 2021

Acknowledgments

Books don't get published without people reviewing them first. We're deeply grateful to all the folks who spent their time and effort carefully reading and reviewing this book. Thank you to Amos King, Ayomide Aregbede, Devon Estes, Doyle Turner, Geoff Smith, Jonathan Carstens, Justin Smestad, Pedro Medeiros, Solomon White, Todd Resudek, and Zach Thomas.

A special thank you goes to Karl Matthias, who provided thorough and thoughtful comments that challenged our assumptions and pushed us to rethink, reshape, and reword concepts and sentences all over the book. The book's quality wouldn't have been the same without Karl's input.

Another special thank you goes to Kim Shrier, whose reviews were so detailed and neatly organized that it made addressing his comments a breeze.

Most importantly, this book would never have gotten to print without the guidance (and patience) of our Development editor, Jackie Carter. Thank you.

Andrea Leopardi

The biggest thank you goes to the most important person in my life, my wife, Kristina. I can't imagine any other human being so full of support and encouragement. You're the light of my life.

Thank you to my parents for setting up a life for me that made it possible for me to write a book about something. That's pretty crazy. Thank you to my other family, Contrada Cavalli, for filling that life with enough love and fun to give me the energy to learn, work, grow, and write this book.

Thank you to José Valim for creating a beautiful programming language that works the same way my mind does. José, you have been (possibly without even knowing it) an incredible mentor in my career, both in programming as well as in interacting with people.

Thank you to everyone at Community.com who supported me and gave me time and mental space to work on this book while giving me chance after chance to learn and grow professionally.

Thank you to Jeffrey for dragging me into this adventure. You were a friend before this book, but now we share something that will, if you think about it, outlive both of us. Spooky but pretty cool.

Jeffrey Matthias

Before I ever started on this book, several people helped set me up for success. Josh Kaiser and Ted Coleman gave me a trial by fire, making me the testing expert for our team only months into my career. That formed the foundation of everything that I know. Bradley Smith gave me my first opportunity to get paid to work in Elixir and taught me a lot of the foundation of my software knowledge, as well as tolerated my insistence on testing all the things.

I'm thankful to Ben Tan for helping me catch the Elixir bug in 2014 and then years later for supporting me in getting this book out, from pitch to final feedback. Moxley Stratton planted some seeds in my head that showed up as content in this book. The Denver Erlang and Elixir meetup has endured more testing talks than any group should have to. John Unruh deserves a special callout for providing plentiful, helpful feedback as a beta reader.

Getting through this book while working at a quickly growing startup turned out to be a team effort in a lot of ways. I'm grateful to everyone at Community.com. Matt Peltier and Josh Rosenheck created an awesome place to work and supported me from the day I was hired. Tomas Koci and Barry Steinglass consistently made time for me to write. Joe Merriweather-Webb has served as a partner to develop a lot of my Elixir testing techniques. Roland Tritsch helped me climb out of burnout, prioritize the book, and get it over the finish line. Every member of every team I've worked with has worked around my schedule. You're all the coolest.

Thanks to my co-author, Andrea, who let me talk him into this project. I'm glad we're still friends.

My parents have supported me through so many career changes that they likely have whiplash. Thanks for sticking with me until I found one where I am constantly learning and able to give back.

My older brother, Karl, has had an outsized impact. After getting me into software in the first place, he's supported me throughout my career. He

inspired me to write a book and then gave me lots of feedback and encouragement the whole way through.

Most of all I want to thank my wife, Amy, and my kids, Noe and Brock, who have been the most patient and supportive of all.

Introduction

Charles Kettering, an American inventor and the longtime head of research for General Motors, has a quote that applies well to our approach to software tests:

> A problem well stated is a problem half-solved.

Think of your tests as stating the problem that your code solves. Once that's done, writing that code becomes easier. Moreover, tests not only state the problem, but they also *verify* that the solution is correct. They're an almost indispensable part of software engineering, and we believe it's important to understand why and how to use them.

Some developers write tests for their software. Some developers don't. There is code running all over the world that doesn't have a single test behind it. So, why do we test? We do it for a plethora of reasons, but they fit into just a few categories.

First of all, we test to increase confidence that our software does what it's supposed to do. Testing gives us *confidence* that our code works as expected. This is true for all kinds of testing, whether for automated tests performed by a machine or for manual tests performed by a human.

The other main reason for testing is to *prevent breaking changes* (also called *regressions*). Imagine you have an existing codebase that you have to work on in order to add a new feature. How can you feel confident that adding the new feature won't break any of the existing features? In most cases, testing is the answer. If the existing codebase is well tested (automated, manual, or both), then you'll feel safer making changes if the testing suite reveals that nothing broke.

In this book we'll focus on *automated testing*. Manual testing, such as QA (Quality Assurance), is fundamental for many reasons, but as developers we often get more value from automated testing. A good automated test suite

allows us to have a fast feedback cycle during development and gives us tight control over how we test specific parts of our applications.

Why Do We Need a Book for Testing in Elixir?

Elixir is an interesting and fairly unique programming language. It provides features and concepts that can be hard to test if you've never dealt with anything like them. Some of those features are common in other programming languages but are more prominent in Elixir (and Erlang), such as concurrency and immutability. Other features, such as resiliency or the OTP framework, are more unique to Erlang and Elixir and can be challenging to test effectively.

From a more practical perspective, Elixir is a great language to write a testing book about because the tools and patterns we use when testing Elixir code are pretty consolidated in the Elixir community. One reason for this is that Elixir comes equipped with its own testing framework, *ExUnit*. We'll explore ExUnit inside and out and we'll learn how to use it in many different situations in order to test our applications on different levels.

Elixir is closely tied to its "parent" language, Erlang. As you likely know, Elixir compiles to the same bytecode as Erlang and runs on the Erlang virtual machine (commonly known as the *BEAM*). Elixir code often seamlessly calls out to Erlang code, and Elixir applications almost always depend on a few Erlang libraries. However, testing seems to be an area where the two languages have a bit less in common. The sets of tools and libraries used by the two languages don't intersect much. For these reasons, we won't really talk about testing in Erlang and will focus exclusively on testing in Elixir. We feel this statement is worth clarifying since the two languages are so close to each other.

Who This Book Is For

This book was written for people with a basic Elixir background who want to get better at the testing tools and practices in the Elixir ecosystem. We will skim over most Elixir concepts, such as the language constructs and data types, OTP, Ecto, and Phoenix. Instead of covering those, we'll learn how to *test* those concepts and tools. Whether you've used Elixir just for a bit or you're an Elixir expert, we think you'll learn a few new things throughout the book.

How to Read This Book

Each chapter in the book addresses a different aspect of testing in Elixir.

In Chapter 1, Unit Tests, on page ?, we'll get going and learn about the "smallest" kind of testing: unit testing. We will cover how and when to write unit tests, the tools to write them in Elixir, and techniques to isolate code under test.

In Chapter 2, Integration and End-to-End Tests, on page ?, we'll move on to testing different components of your system that interact with each other. We'll learn how to test components together, as well as how to isolate components to run more focused integration tests. We'll also touch on end-to-end testing, that is, testing the whole system from the perspective of an outside entity.

In Chapter 3, Testing OTP, on page ?, we'll learn about testing one of Erlang and Elixir's most unique features, OTP. OTP and processes in general present quite a few challenges when it comes to testing. We're going to talk about those and learn techniques to make testing these abstractions easier.

In Chapter 4, Testing Ecto Schemas, on page ?, and Chapter 5, Testing Ecto Queries, on page ?, we'll talk about testing code that uses the Ecto framework to validate data and interact with databases. Ecto is a widely used library in the Elixir landscape, and the community has created patterns on how to test code that makes use of it.

In Chapter 6, Testing Phoenix, on page ?, we'll cover Elixir's most used web framework, Phoenix. Phoenix provides several moving pieces. We'll learn how to test those pieces in isolation as well as how to test that the pieces of your Phoenix application work correctly together.

In the last chapter, Chapter 7, Property-Based Testing, on page ?, we'll explore a technique for introducing randomness in your testing suite in order to cover large amounts of inputs to your code and increase the chances of finding inconsistencies.

Note: The chapters in the book don't have to be read in the order they're laid out. For example, if you're particularly interested in testing code that uses the Ecto framework, you can jump directly to Chapter 4, Testing Ecto Schemas, on page ?. Most chapters are self-contained. However, we recommend that you read Chapter 1, Unit Tests, on page ?, and Chapter 2, Integration and End-to-End Tests, on page ?, in order: these chapters lay the

foundations for the terminology, tools, and techniques that we'll use throughout the book.

About the Code

A testing book is a strange beast. Most programming books show "application code" when discussing examples and often omit or give little attention to tests. In this book, we want to focus on testing, but we need application code since there's no point in testing if you don't have anything to test. At the same time, we don't want to focus on the application code since it would take away from what we want to talk about, which is testing. As we said, it's a strange beast.

Throughout the book we'll work on two main applications. In the first three chapters, Chapter 1, Unit Tests, on page ?, Chapter 2, Integration and End-to-End Tests, on page ?, and Chapter 3, Testing OTP, on page ?, we'll work on Soggy Waffle. Soggy Waffle is an application that reads the weather forecast for a given area off the Internet and can send SMS alerts in case rain is expected in the next few hours. It's not a broadly useful application, but it helps illustrate many Elixir testing concepts.

In the next two chapters, Chapter 4, Testing Ecto Schemas, on page ?, and Chapter 5, Testing Ecto Queries, on page ?, we'll use a very basic application, called Testing Ecto, to illustrate how to test applications that use the Ecto framework.

Chapter 6, Testing Phoenix, on page ?, will have a single application with examples covering the different interfaces provided in the standard Phoenix library.

The last chapter, Chapter 7, Property-Based Testing, on page ?, won't follow a particular application in order to focus on different concepts related to property-based testing.

We'll have to continuously "escape" these applications over and over again. We made this choice because our focus is testing, and many times we would have had to come up with artificial and forced features for these applications in order to talk about some testing topic. In those cases, we'll just go with self-contained examples that allow us to directly address some testing topics without making our main application a total mess.

Online Resources

You can get the code from the book page on the Pragmatic Bookshelf website.[1] We hope that when you find errors or think of suggestions that you'll report them via the book's errata page.[2]

One online resource we highly recommend is Elixir's excellent documentation. You can find that on Elixir's official website.[3] Particularly interesting for this book is the ExUnit documentation since we'll get to use ExUnit a lot.[4]

If you like the book, we hope you'll take the time to let others know about it. Reviews matter, and one tweet or post from you is worth ten of ours! We're both on Twitter and tweet regularly. Jeffrey's handle is @idlehands and Andrea's is @whatyouhide.[5] [6] You can also drop notes to @pragprog.[7]

Andrea Leopardi and Jeffrey Matthias

July 2021

1. https://pragprog.com/book/lmelixir
2. https://pragprog.com/titles/lmelixir/errata
3. https://elixir-lang.org
4. https://hexdocs.pm/ex_unit/ExUnit.html
5. https://twitter.com/idlehands
6. https://twitter.com/whatyouhide
7. https://twitter.com/pragprog