Extracted from:

Adopting Elixir

From Concept to Production

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Your Elixir Source

Adopting Elixir

From Concept to Production

Ben Marx, José Valim, Bruce Tate

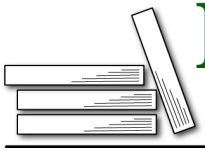
edited by Jacquelyn Carter

Adopting Elixir

From Concept to Production

Ben Marx
José Valim
Bruce Tate

The Pragmatic Bookshelf
Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

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

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

Printed in the United States of America.
ISBN-13: 978-1-68050-252-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—March 2018

Introduction

Elixir is a rapidly growing functional language and the first production applications are emerging. Still, early adopters may find some important details missing. As with any emerging language, common questions arise:

- At what point do the promising rewards of the Elixir platform outweigh the risks?
- How do you recruit and train teammates who might never have used a functional language before to build consistent code?
- How do you wrestle with the trade-offs with distributed systems?
- What tools do the pros use to test, deploy, and measure your applications?
- What critical but less popular tools are available to solve issues like integrating external code or measuring performance trade-offs in production?

We wrote *Adopting Elixir* to change that. Rather than write another Elixir book about some narrow aspect of the language, we decided to write an experiential book containing the kinds of details that are typically difficult to find for an emerging language. It's a daunting task.

Said another way, you can find plenty of books out there to cover known topics. This book is more of an exploration of the expansive field of resources for new Elixir developers. Most often, we won't give you all of the answers. Instead, we'll help you know what you don't know, help you build a limited foundation with the basic trade-offs between possible solutions to a problem, and point you to the community for more answers.

Who This Book Is For

Each adoption story has many actors, and many of them have broadly different roles and responsibilities. Our book cuts across all adoption stories.

Such a book will have many different types of readers, and no single reader will find everything covered here relevant to them. Team leads and technical managers will want to know how to recruit and train, but pure programmers will probably find such details distracting. Beginners from traditional languages like Java or Ruby will crave more information on making the transition to a more functional, concurrent language.

We decided to write this book anyway, because the information is important right now for the greater community. We hope you'll agree.

Still, this book is not for everyone. If you're the type of reader who is likely to be frustrated when you find content that is not specifically for you, we don't think you'll be happy. The Pragmatic Bookshelf has the biggest selection of Elixir books in the industry and we'll gladly help you find one that's right for you, but you may want to pass on this one.

If you are a CTO looking for a book to help you build a business case for using Elixir, we don't believe such a book exists. The first couple of chapters will introduce you to a few stories that you may find instructive, with some hints toward financial justifications beyond "It scales well." But in the end, we decided we did not want to build a full business case in this book. This book may help some technical managers who code, but is probably not for the C-level executive.

We're writing this book for those in the technical Elixir community who find themselves adopting Elixir (or who plan to in the near future). Look, Elixir adoption can be hard because the collective problems we're solving are demanding. It's a functional, concurrent, distributed language. Any one of those concepts is difficult to understand. Many of our readers will be learning all of them at once. Have courage, though. We also know that many teams are making the successful transition.

Our combined experience suggests there is a growing segment of Elixir programmers who need to walk with successful Elixir practitioners. That list includes day-to-day developers looking for help making the transition from other languages. Experienced programmers may be deploying their solution from the experimental staging servers into production for the first time, or learning to scale their solution, or beginning to dabble in distribution for better fault tolerance. We can't promise you'll like everything in this book, but we can guarantee that you'll find something you'll like, something you've not seen before.

About the Authors

Ben Marx is one of the first developers to use Elixir at scale. As a lead developer at Bleacher Report, he was intimately involved in their transition from Ruby on Rails. His involvement spans the whole development cycle, from the initial plans for the Elixir migration through recruiting, development, production, and debugging that live system.

José Valim is the creator of the Elixir programming language and was once a member of the Rails Core Team. He graduated with an engineering degree from São Paulo University, Brazil, and has a Master of Science from Politecnico di Torino, Italy. He is also co-founder and Director of R&D at Plataformatec, a consultancy firm based in Brazil. He is the author of *Programming Phoenix [TV16]* and *Crafting Rails 4 Applications [Val13]*. He now lives in Kraków, Poland, with his wife and children.

Based out of Chattanooga, Tennessee, Bruce Tate is a father of two, as well as a climber and mountain biker. As CTO, he helped grow icanmakeitbetter.com, the insight community platform, from a cocktail napkin drawing in 2010 to its acquisition in 2016. As of this writing, he is leading the company through an Elixir migration. He's an international speaker and the author of several other books, including *Programming Phoenix [TV16]* and *Seven Languages in Seven Weeks [Tat10]*.

Now you know us. Let's get to the book.

How To Read This Book

We'll cover the whole adoption lifecycle in three parts, from concept to development and finally into production. As you progress through the book, the chapters will go deeper into technical details. This is intentional, as we're working to fill the many holes we've seen our teams and customers encounter. We'll try to provide enough detail to lay the right theoretical foundations and point you toward the right solution, and then move on.

Feel free to read the individual parts in any order, or not at all based on your needs. We've designed the code and prose so that you can do so. We'll tell you if there's something from an earlier chapter we think you should know. For example, if you have already built and trained your Elixir team, you may find more value in the development and production lessons explored in Parts II and III. On the other hand, if you are early in your adoption journey and

you are still deciding if Elixir is the right tool for you, you will likely get more mileage from the first chapters and you can revisit the production discussion once development starts.

We understand that's not the typical experience, but we made that trade-off because each adopting developer has a different set of needs. No simplistic grouping of technologies can cover everything that needs to be said. We're willing to take our lumps in the review process because we believe in the need for this book in the greater community.

With those bits of housekeeping done, let's dive into the parts of the book. We'll also highlight which parts will be of particular interest based on your individual use case.

Part I: Concept

Adoption is in part a social problem. Adopting an emerging technology means becoming a salesperson. Helping stakeholders, teammates, and potential new hires understand that our technical decisions are wise and in the best interests of the companies we serve is a critical part of the process. New languages often mean new hiring and team-building strategies. Adoption *is* changing habits and practices to take best advantage of our new platform. This book will give you tools to help automate those things you can and lay out experiences to help you handle the things you can't. We'll walk you through the discussion in three chapters:

Team Building

Building a team for an emerging language is a little different than ramping up for a well-known, established technology. We'll tell you what we did to train and recruit talent, set expectations, and keep folks motivated. If you have an established effective team and a good handle on recruiting and training, you will probably want to skip this chapter. If you find yourself ramping up or training for a new adoption or if you're concerned about finding talent, this chapter will help you in your efforts.

Ensuring Code Consistency

It's easy to fall into old habits when you're learning a new language, but that would limit the benefits you'd reap. This chapter will show you how to use automated tools to gently nudge your team toward a more beautiful idiomatic coding style. If you have a comfortable cadence and are already well versed with lexers like Credo, Elixir's testing ecosystem, and Elixir's documentation tools, you may not give this chapter more than a quick

skim, but most developers will appreciate the concepts outlined in this chapter. You *can* automate many different aspects of code quality.

Legacy Systems and Dependencies

Many of the companies adopting Elixir are choosing it to replace a legacy system, and we'll cover that topic. Dealing with legacy systems also means carefully considering each new line of code you write and each new dependency you add. Sometimes code becomes legacy because of business needs, other times because the code becomes less healthy. In this chapter we will talk about how to replace legacy systems, building tomorrow's friendlier legacy systems as you write code today, and working with internal and external dependencies.

If you're a manager, an executive, or a team lead tasked with building a team and establishing culture, Part I is for you. The chapters in this part will also be of interest if you are starting your own project and need to understand what tools can help you build consistent code.

When you're through with Part I, you'll understand how to build a unified tool that writes uniform code. You'll then think about the best ways to think about taking that old system apart to adopt the new, should that be your chosen path.

Part II: Development

This part will show you how others have successfully built Elixir applications using new development teams or retooled teams who wrote in some other language. We'll focus on how to write code to do things beyond what you'd find in most typical technology books. We'll focus specifically on ideas and tools we've found difficult to find elsewhere. In particular, we've divided this part into the following chapters:

Making the Functional Transition

Object-oriented developers sometimes have trouble learning functional languages. This chapter gives advice to help make that transition. It is tailored specifically for beginning and intermediate Elixir developers, especially those who have come from other ecosystems.

Distributed Elixir

Adopting a new language is hard enough when you're only concerned about one system. It takes experience to learn to split concepts across the wire. This chapter provides exactly that. We'll talk about how to name things, the role of OTP, and how to think about distribution. If you're a technical

lead or looking to break Elixir out of a single box, this chapter will help you reason about the next level of challenges you're likely to face.

Integrating with External Code

Sometimes, Elixir is not enough. When your code needs to step out of its universe, Erlang and Elixir provide several tools to do exactly that. This chapter covers those tools, whether you decide to stay in the same memory space, or use different processes or different machines altogether. If you plan to stay within the Elixir ecosystem for all of your application needs, you'll likely want to skip this chapter. Just give it a quick skim so you'll know the techniques available to you should the need arise.

If you're a new Elixir developer making the transition from OOP to FP, the early chapters in this part will help. If you're an experienced developer but struggling with what it means to write a distributed project or the approaches to integrating external code, the later chapters in this part will have something you find useful.

When you've completed Part II, you'll have more tools to think about the things that trip up early adopters from functional programming to concurrency, even distributed systems. Then, in the final part, we'll worry about deployment.

Part III: Production

Every new language community has to work out what to do with deployment. Elixir is no different. It's not surprising that one of the most common questions Plataformatec received was how to deploy, and how to monitor the system once deployed. This part of the book will point to some prevailing wisdom in these areas. In particular, you'll see chapters for:

Coordinating Deployments

Deploying a simple system on a single server is a pretty easy problem, but modern applications no longer fit that profile. This chapter will show how successful DevOps folks think about releases and the tools they use to deploy. In some teams, the folks that write the code are the same ones that deploy and support it in production. If you're involved in any way in deploying Elixir or packaging your code for deployment, this chapter is for you.

Metrics and Performance Expectations

Typically, new languages have some lesser-utilized areas with less documentation than others. Since performance measurements often happen

after production applications are close to ready, this topic is one of the last to develop. When it's time to push Elixir to the limits and test performance, you'll need to know how to measure your performance and report those results. This chapter will tell you what you need to know. The techniques are advanced and the topics specialized, so you'll want your Elixir foundations to be pretty solid to attack this material.

Making Your App Production Ready

Once a system reaches deployment, the debugging and monitoring tools change. That throws many developers off, but Elixir has some unique capabilities that greatly simplify this process. This chapter will focus on instrumenting, measuring, and monitoring production systems. Just about all developers need to know about the debugging, logging, and reporting techniques in this chapter.

If you're in operations or responsible for seeing that your application is easy to deploy and manage, this part will be invaluable to you. Leads and architects will also want to understand how the deployment story fits together.

When you've finished this part, you'll know how others deploy with confidence. You'll learn what to measure, how to instrument your code, and how to monitor for the best possible reliability and information.

About the Code

The sample code and examples included in this book are written using the Elixir programming language, and will walk you through many broadly different Elixir concepts. Some of those are just fragments and some are full working examples. We'll show you how to use each example in the context of the book.

This book is about showing you a wider picture of the evolving greater Elixir ecosystem. Keep in mind that the nature of this book is that some of these code examples are just snippets, or segments of a fully working system. We can't possibly show you whole working systems for all of the examples in this book. Such a book would be many times the size of this one and take much longer to write.

Instead, we will let the prose and the code work together. Let the prose guide you through the proper use of the code examples. If you find any concepts that are not clear, just let us know in the forums. We will help you the best we can.

Online Resources

You can find all the example code for this book on its Pragmatic Bookshelf website,[1] alongside a handy community forum if you'd like to reach out for help along the way.

While we've worked hard to make our code examples and explanations bug-free and clear in every way for our readers, we've written enough software to know that we're fallible. Thanks in advance for reporting any issues that you find in the book code or text via the errata form, also conveniently found on the book website.

Thank you for joining us in your Elixir adoption story! We are excited to have you with us.

Ben Marx, José Valim, & Bruce Tate

March 2018

1. https://pragprog.com/book/tvmelixir/adopting-elixir