Extracted from:

From Ruby to Elixir

Unleash the Full Potential of Functional Programming

The Pragmatic Bookshelf

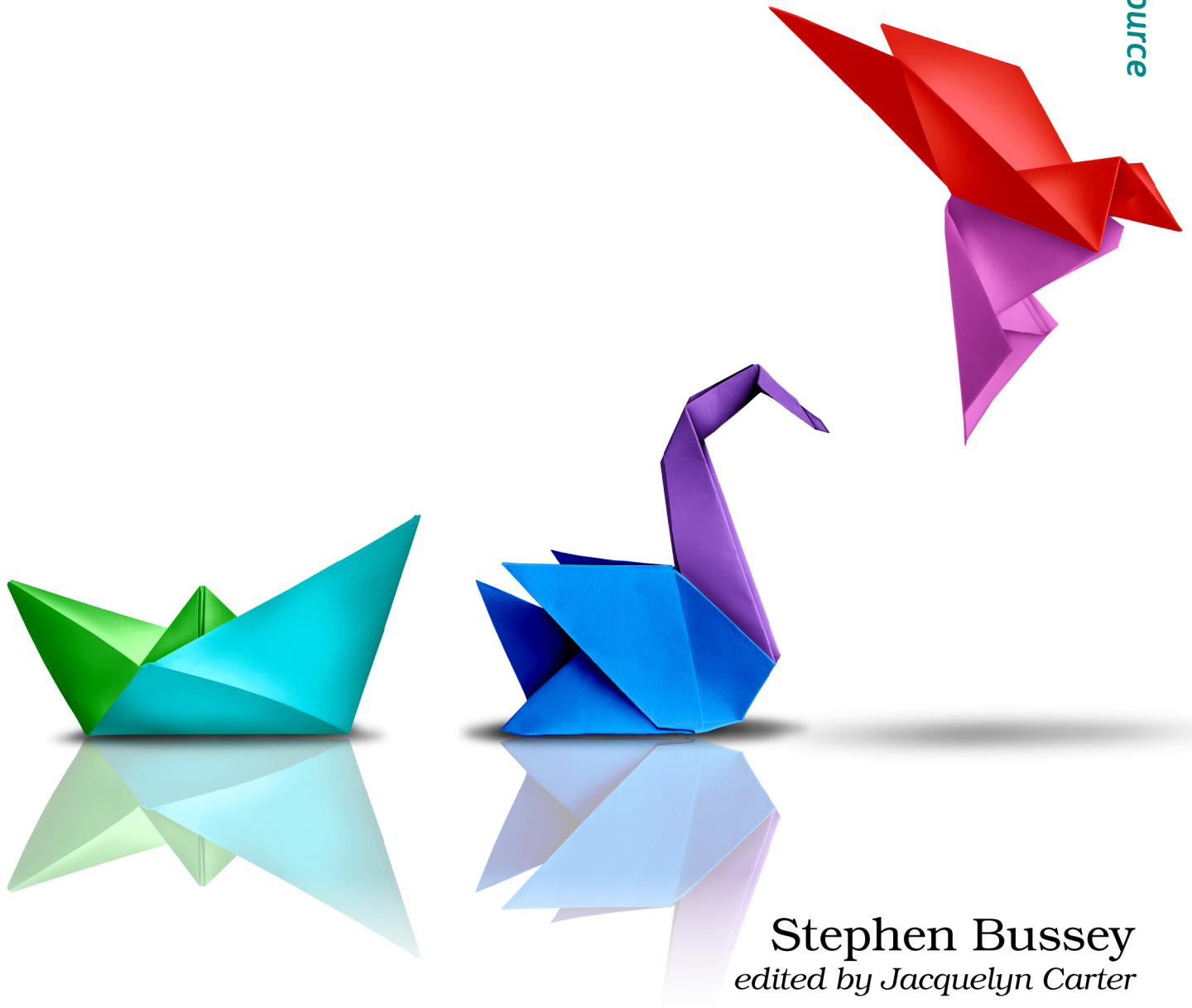
Raleigh, North Carolina

The Pragmatic Programmers

Your Elixir Source

From Ruby to Elixir

Unleash the Full Potential of Functional Programming



Stephen Bussey

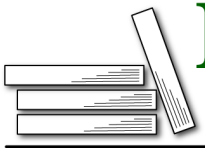
edited by Jacquelyn Carter

From Ruby to Elixir

Unleash the Full Potential of Functional Programming

Stephen Bussey

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*.

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

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

ISBN-13: 979-8-88865-031-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—May 31, 2023

Elixir has emerged over the past few years as a most loved language[1] that is used by many businesses and hobbyists to write reliable software systems. Many Elixirists consider it their superpower of productivity and stability. Hopefully, by the end of this book, you will see why this is the case.

It's hard to learn a new language, harder to become production-ready with it, and even harder still to win a business over that they should bet on the unfamiliar. The juice is worth the squeeze, though. Elixir opens a new way of thinking about programming that carries over into other languages as well.

Your knowledge of Ruby influences how you view and write in other programming languages. Similarly, as you develop an understanding of Elixir, it will also influence how you think when writing code. Even if you were to never use Elixir in production, you'll still benefit and grow as a programmer.

We'll start this chapter by taking a look at what makes Ruby such a great language. You'll see why Elixir is a similarly great language and why its future is bright. You'll learn about the technology that Elixir is built on top of: Erlang, OTP, and the BEAM. Finally, you will write a touch of Elixir code and run it on your computer.

Before we talk about Elixir, let's talk about Ruby.

The Joy of Ruby

Some people have pitted Elixir as "against Ruby," but that's not the case. When you learn a new language, it doesn't have to come with a reduced respect for what you used in the past. It also doesn't mean that you can't use that language anymore. Especially with the rise of microservices, it's very possible that you can program in Ruby and Elixir at the same company!

Ruby is a language based around joy, which leads to several non-technical aspects that make it an appealing language: a healthy foundational philosophy, a strong community, and continual improvement. Let's go over each point and how it benefits Ruby.

Solid Foundations

Yukihiro Matsumoto (Matz) created Ruby with a philosophy that continues to influence it to this day. The philosophy of Ruby is to create the feeling of joy for programmers that use it. This philosophy influences the design of the language, the way that libraries are built—really everything about Ruby.

1. https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted

The happiness of programmers is just as important as what those programmers create, because happiness affects all aspects of your life. Businesses benefit from this as well. A happy programmer is more likely to be happy with their job, more likely to stay with their company, and more likely to create a better product.

A Strong Community

Matz's philosophy is felt in the community with the phrase "Matz is nice, so we are nice" or MINSWAN.[2] The Ruby community is welcoming and helpful, which is critical for adoption of a language over time. New developers won't want to learn a language if they are pushed away due to negativity.

Another strength of the Ruby community is a culture of testing. This might be obvious to people that have spent a lot of time with Ruby, but it's certainly not true in all language communities. Having a culture of testing creates better libraries and applications. This testing culture contributes to joy over time.

Continual Improvement

Ruby is not a static language. The language doesn't push many breaking changes (this would certainly not elicit joy) but it has continued to evolve. There are many smart people and companies that deliver performance improvements, increased security, and even large projects like type checking.

Plus, the major libraries have continued to innovate over the years. Ruby on Rails 7[3] is the best release of Rails yet. It continues to set a high bar for programmer productivity and happiness.

The continued improvements to Ruby guarantee relevance for Ruby developers. Ruby isn't going anywhere, and it isn't a goal of this book to try to replace Ruby with Elixir.

Next, let's explore a bit of what makes Elixir special.

The Case for Elixir

Anytime that you pick a new technology, you will have to make a case for it. The first consideration is your personal decision of whether you want to spend time learning the language. Next, you'll need to bring it to your professional peers and get their buy-in that they also want to spend time learning it.

2. https://en.wiktionary.org/wiki/MINASWAN
3. https://rubyonrails.org/

Finally, your business will need to make the case of whether they want to invest in it.

The technical aspects of a language are obviously important in these decisions, but the non-technical aspects are just as important. In the previous section, we spent a lot of time talking about the non-technical aspects that make Ruby a joy to use and learn.

We'll briefly cover a few non-technical strengths of Elixir, but we'll also look at why the technology itself is appealing.

Non-Technical Strengths

Elixir is small and relatively new. These are two things that you'll be working against when you make the case for it. However, Elixir finds strength in its community, innovation, and philosophy.

Close-Knit Community

The Elixir Forums[4] and Slack Group[5] are a wealth of knowledge and friendly faces. It is rare that a question goes unanswered (and even unsolved) in these places because the community shows up to help.

Culture of Testing

Elixir, like Ruby, has a healthy testing culture that permeates every layer of the technical stack. It is rare to find libraries that are untested.

Language Design Philosophy

Elixir does not have a vocalized philosophy like Ruby does, but it's clear that there is an influence there. Elixir's creator José Valim[6] has talked about taking influence from Ruby, Clojure, and Erlang. The language syntax is pleasant to use and elicits joy much like Ruby. This is apparent when seeing how hard the Elixir core team works to avoid breaking changes—you can usually upgrade Elixir versions without too much work.

Continual Innovation

Over the past few years, there have been a ton of innovative features and libraries added to the Elixir ecosystem. These have been developed by the core team, but also by private companies developing edge-pushing libraries.

4. https://elixirforum.com
5. https://elixir-slackin.herokuapp.com/
6. https://twitter.com/josevalim

> The most well-known innovations in the space are Phoenix LiveView,[7] machine learning with Nx,[8] and hardware development with Nerves.[9]

A healthy foundation wouldn't matter if the technical aspects weren't also strong. Let's look at why Elixir is appealing for modern application development.

A Solid Foundation with BEAM

Elixir is a functional programming language that is built on top of an almost forty year old foundation, the BEAM. We'll be going into what that is in the next section, but the important takeaway for now is that it's a unique runtime that empowers parallelism and fault tolerance in a way that isn't common in other languages.

Fault tolerance and isolation is a premier strength of Elixir and the BEAM. Errors are going to happen, whether they are bugs, service outages, or anything in between. The BEAM provides ways for us to determine how errors should affect our application. Do you want an error to bring down the whole thing? Do you want an error to only affect the request that it was serving? Do you want to restart a piece of your system when an error occurs? Select the failure mode of your application, so your application behaves predictability when things go wrong.

It's human nature to focus on the happy path. When evaluating a programming language, that would mean evaluating how you code in it, the libraries you use, and how data flows through your system. But, the unhappy path is just as (or sometimes more!) important. Elixir and the BEAM provide a core foundation for programming the unhappy path you want, rather than taking whatever you get.

Technical Benefits

Elixir applications can fully utilize their host CPU cores (parallelism) without having to write complex code. It's possible for other languages to achieve this same level of parallelism, but the unique aspect is how simple it is to achieve with Elixir—it's nearly free. Elixir provides an easy solution for the opposite of parallelism—serial code.

The combination of parallel and serial code gives you complete control over your system run-time performance. This is unmatched in most every other

7. https://github.com/phoenixframework/phoenix_live_view
8. https://github.com/elixir-nx
9. https://www.nerves-project.org/

programming language today. Elixir (and BEAM languages) are equipped with these world-class capabilities out of the box.

Elixir is really well-suited for building reliable, modern systems. This especially applies for web systems, but Elixir can be used in hardware systems, communication systems, machine-learning, and more. You will write more performant systems that are cheaper to run when you use Elixir.

These are just a few of the technical benefits of Elixir. A lot of Elixir's biggest strengths come from its foundation with Erlang and the BEAM. Let's dive into what those are next.