Extracted from:

Design It!

From Programmer to Software Architect

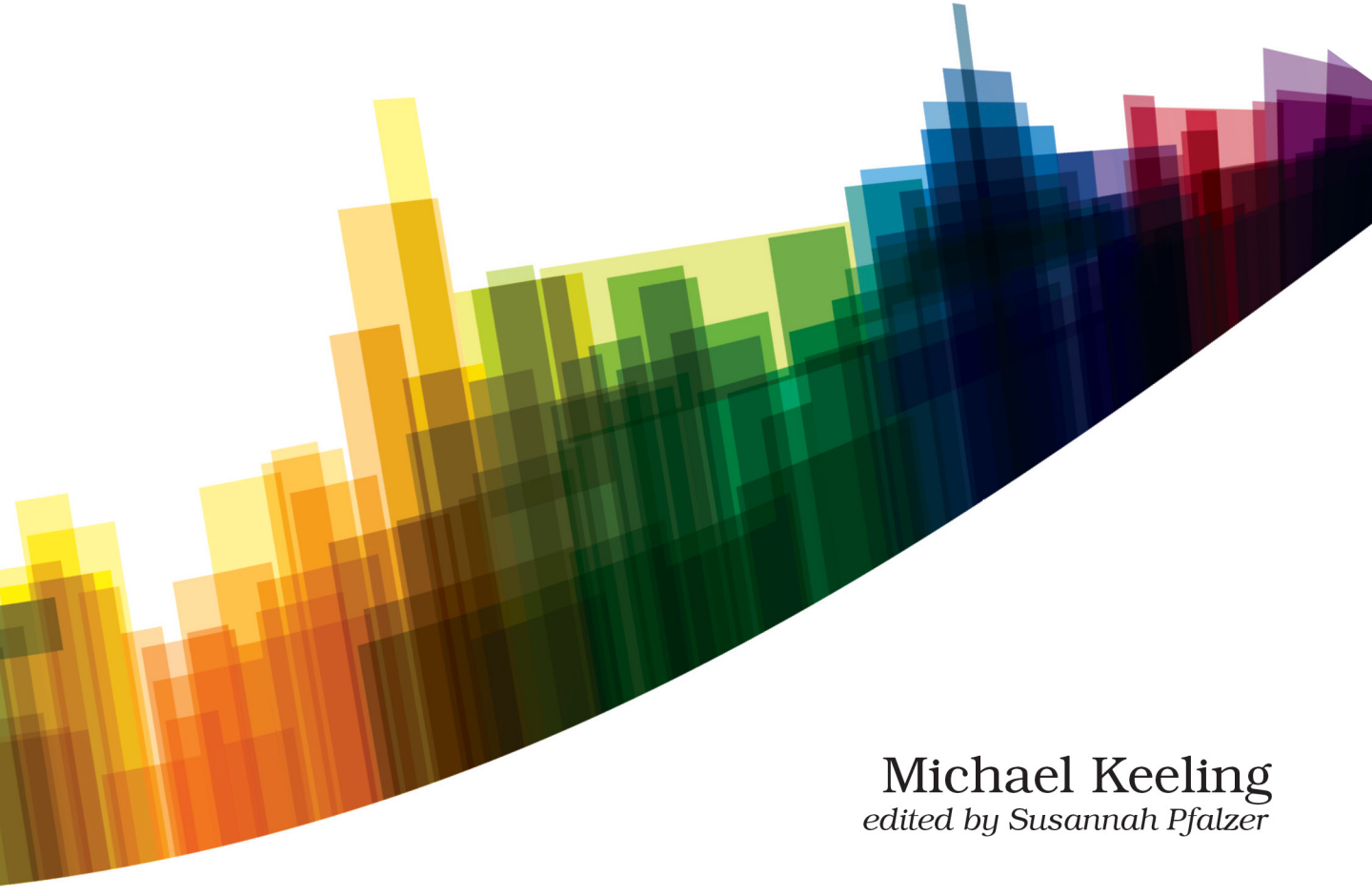
The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Design It!

From Programmer to Software Architect



Michael Keeling

edited by Susannah Pfalzer

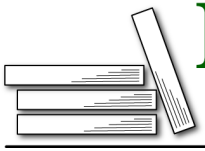
Design It!

From Programmer to Software Architect

Michael Keeling

The Pragmatic Bookshelf

Raleigh, North Carolina



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
Development Editor: Susannah Davidson Pfalzer
Indexing: Potomac Indexing, LLC
Copy Editor: Liz Welch
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: 978-1-68050-209-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—October 2017

Become an Architect for Your Team

On some teams, *architect* is an official team role. On other teams, there is no explicit role and teammates share the architect's responsibilities. Some teams say they don't have an architect, but if you look closely, someone is fulfilling the architect's duties without realizing it.

Architects are leaders, but being a *software architect* also implies a person who thinks about software design in a certain way. No matter what the title on your business card reads (mine still reads *software engineer*, my choice), you can be a software architect. Every team has at least one architect. The best teams have several.

If your team doesn't have an architect, congratulations, you've got the job! You don't need permission to inject architectural thinking into your team's design discussions. Start asking questions about quality attributes. Point out when the team makes trade-offs. Volunteer to write up design decisions and begin accepting more architecture design responsibilities.

If your team already has an architect, then ask that person how you can help. When possible, work closely with your architect and take advantage of every learning opportunity you can. Developing a software system is a big job. The more people who pay attention to the details, the greater your chance of success. Every team should be so lucky as to have many knowledgeable software architects!

Make the Move from Programmer to Software Architect

An average software architect has developed three to five software systems with increased technical responsibility on each software system. Depending on the software you build, as your architecture responsibilities grow you may find you have less time for programming. This is normal, though software architects should never stop programming altogether.

To measure your growth from programmer to software architect, create a project portfolio. For every software system you build, no matter your role, briefly describe the software system and what you learned during your time developing it. This kind of reflective practice is essential for all technical leaders but especially software architects.

Here are some questions you should answer about each project in your portfolio:

- Who were the stakeholders and what were the primary business goals?
- What did the high-level solution look like?

- What technologies were involved?
- What were the biggest risks and how did you overcome them?
- If you could do it all over again, how would you do it differently?

Whether your goal is promotion or simply professional growth, be patient. You might have the chance to design a software system of meaningful complexity only every three to five years. If you are lucky, you will see between 8 and 15 software systems throughout your entire career. Be prepared to take advantage of architecting opportunities as they arise. Work with your teammates to give everyone a chance to grow their skills. I promise there is more than enough interesting architecture work for everyone!

Always remember, *software architect* is a way of thinking, not just a role on the team. When you're wearing your programmer hat, you'll make dozens of design decisions daily. Some of these decisions have architectural significance. Anyone who makes a decision that influences the structures of the software system becomes the *architect pro tempore*. It's up to you to make good decisions and uphold architectural integrity no matter what the title on your business card reads.

Build Amazing Software

There are lots of things that have to go right when building a software system. Architecture connects them all together and provides a foundation for success. Here are six ways software architecture helps you in your quest to build spectacular software that your stakeholders will love:

1. Software architecture turns a big problem into smaller, more manageable problems.

Modern software systems are large and complex, and they have many moving pieces. The architecture precisely explains how to partition the system into smaller, bite-sized chunks while also ensuring the system as a whole is greater than the sum of its parts.

2. Software architecture shows people how to work together.

Software development is as much about human communication as it is technology. Software architecture describes how the whole system comes together, including the people who build it. When you know the architecture, then you can see how people can collaborate to develop software. The larger the software system, the more important this becomes.

3. Software architecture provides a vocabulary for talking about complex ideas.

If I don't understand what you're talking about, then we won't be able to collaborate. Instead of spending all our time inventing vocabulary and concepts, we can use the essential concepts and core vocabulary of architecture as the starting basis for collaboration. Now we can spend our time solving our users' real problems.

4. Software architecture looks beyond features and functionality.

Features and functionality are important, but they are not the only thing that determines whether or not software is awesome. When designing architecture, you'll consider not only the features but also costs, constraints, schedules, risk, the ability of the team to deliver, and most importantly quality attributes—things like scalability, availability, performance, and maintainability.

5. Software architecture helps you avoid costly mistakes.

In *Who Needs an Architect? [Fow03]*, Martin Fowler defines software architecture as "…the important stuff. Whatever that is." The important stuff is nearly always what we think will be difficult to change without significantly increasing complexity. Grady Booch echoes Fowler's sentiment by defining architecture as the "…significant design decisions (where significant is measured by the cost of change)."[1] Software architects are not omniscient, but designing an architecture will help you discover the challenging (and interesting) parts of the problem that might cause big trouble later.

6. Software architecture enables agility.

Your software should respond to change like water, by bending around obstacles with ease. If software is like water, able to take any shape, then software architecture is the container that holds it. That container can be rigid like a box or flexible like a plastic bag. It can be thick and heavy or lightweight. Without an architecture, software, like water, follows the path of least resistance and sprawls uncontrollably. A software system's architecture provides the structure within which change is possible.

We'll expand on these ideas throughout the remainder of the book.

1. Grady Booch. *Abstracting the Unknown*. SATURN 2016. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=454315

Case Study: Project Lionheart

As we cover new ideas in each chapter, we'll apply them to a case study, Project Lionheart. The case study is based on a real system, but the names and situations have been changed for teaching and legal purposes.

Design an Architecture to Solve This Problem

The City of Springfield is facing budget shortfalls and needs to cut costs. Mayor Jean Claude van Damme (no relation to the action hero) has hired our team to streamline the city's Office of Management and Budget (OMB).

When a city employee needs to purchase something for more than a few thousand dollars, the OMB issues a Request for Proposals (RFP) in the local newspaper. Businesses bid on the RFPs and the OMB awards a contract based on the competitiveness of the bid and other factors. The OMB monitors more than 500 active contracts and RFPs for everything from toilet paper to medical supplies to basketballs. The OMB manages all this data in spreadsheets.

Mayor van Damme hopes modernizing the OMB will improve a few strategic areas.

- Over half of all RFPs have a single bid. The city is potentially overpaying for lower-quality services.
- Finalizing a contract takes months. Many businesses get lost in the multistep process.
- Publishing a new RFP takes up to 6 weeks. This process must be faster.

Throughout Part II, we'll flesh out this case study and work together to design a plausible architecture to solve some of these problems.

Next Up

Software architects are responsible for quite a lot. Designing interesting, complex software systems and working with different people feels good and is well worth the effort. Becoming a software architect is not an overnight journey. If you focus on the architect's core responsibilities and do your best to apply the architectural fundamentals, mainly selecting structures to promote desired quality attributes, then you'll do great.

In this chapter, you learned what architecture is and what architects do. In the next chapter, you'll learn how to use design thinking to figure out what should go into the architecture.