Extracted from:

Design and Build Great Web APIs

Robust, Reliable, and Resilient

This PDF file contains pages extracted from *Design and Build Great Web APIs*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit http://www.pragprog.com.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2020 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

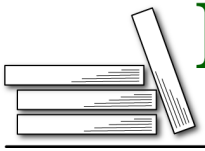
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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
Executive Editor: Dave Rankin
Copy Editor: Molly McBeath
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*.

Copyright © 2020 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 978-1-68050-680-8
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—October 2020

One of the challenges of designing APIs is that so many results are possible. Design, like art, is pretty much in the eye of the beholder. What some find beautiful or correct, others find lacking. If you get a group of five API developers together to review the same user interview information we gathered in the previous chapter, you're likely to end up with five different API designs. And that's normal. We don't all need to agree on the exact details of every API you and your company release. However, we do need to agree on some things and on a general process to use to design the APIs.

In this chapter, we'll explore the notion of design thinking using patterns like "jobs to be done," and we'll look at a simple API design method you can apply to make sure your API designs are consistent. Finally, you'll learn how to render your API designs with sequence diagrams to help visualize the API before you start writing the code.

The Power of Design

Art, architecture, and design all rely on the unique traits of the individual in order to turn out the final result—the building, the canvas, the musical score, or the API. Even though the final results rarely look the same from person to person, artists and designers often share a similar method or style when they go through the process of creating their work. And it's this process or method that can be described and standardized. When we all use the same general method, even when we come to different final results, those results are very likely to be compatible—they work together easily. This is the power of a good design process. It allows people to continue to think for themselves, to contribute in unique ways, and still come up with results that are compatible across the organization.

Creating compatible solutions is an important element of a healthy API program. In most companies, the API program will span many years, have lots of different people involved, and over time use several different technologies and products. Throughout all those changes, one of the things you can keep as a constant is your design process or method. By teaching your developers the same design method, you gain consistency without overconstraining their creativity.

A good design method also doesn't assume any single technology stack or tool set. That means you'll be able to grow an API program over time that's built on the power of consistent design methods and not reliant on any one API format or protocol or other technical elements. In fact, good design process makes it easier to change tooling over time without introducing incompatible implementations.

• Click HERE to purchase this book now. discuss

So design helps you bring consistency and compatibility to your API program. But how does that happen? Two key elements of any successful design program are these:

- Design thinking: The way people think about the process of designing an API
- Jobs to be done: A focus on the tasks your API users are trying to accomplish

Before jumping right into our API design method, let's take a moment to review design thinking and jobs to be done in a bit more depth.

Design Thinking

The phrase "design thinking" is used quite a bit in product design to identify a point of view and a collection of practices that help people think differently about the way they design and build products. Tim Brown, one-time CEO of the design firm IDEO, describes design thinking as "a design discipline that uses the designer's sensibility and methods to match people's needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity."[1] The key words in that quote are "match people's needs" and "viable business strategy."

While we may not think about it much, APIs are really meant to do both those things. They need to match people's needs and they need to support a viable business strategy. The people we need to pay attention to when designing APIs are often not just end users but also developers. Knowing your audience well is key to creating APIs that do a good job of solving the problem. For example, designing APIs for internal use by your own team is different than designing APIs for developers working for your customers or partners. Each is likely to have a different point of view, different set of skills, and different priorities.

Creating something that supports a viable business strategy or goal is important too. It's not a good idea to set out to design and build a bunch of APIs without knowing there is an audience for them and a business problem to be solved. Just creating APIs to say you *have* APIs is not a viable strategy. At the same time, building APIs that no one asked for, that no one really needs, is also a bad idea. I've seen API teams work hard to get funding from leadership, then work hard to build and release generic APIs that no one requested, only to find that company leaders complain that the new APIs

1. https://hbr.org/2008/06/design-thinking

aren't what they need and thus go unused. This isn't a viable business strategy.

What does it take to build APIs that "match people's needs" and support a "viable business strategy"? That is where the concept of "jobs to be done" can help.

Jobs to Be Done

An important reason to design and build an API is to solve a problem—to "match people's needs." In Chapter 3, Modeling APIs, on page ?, we worked to discover the problem (onboarding new customers). In our case, the customer onboarding work is a job that needs to be done. That's what Harvard professor and author of the books *Innovator's Dilemma* and *Innovator's Solution* Clayton Christensen calls "jobs to be done."[2] He says, "People don't simply buy products or services, they 'hire' them to make progress in specific circumstances." Translated into API terms, developers want to make progress (solve problems) and they'll use (or hire) whatever APIs will help them do that.

This means that designing APIs to solve existing problems is essential. Every organization, large and small, has problems that need to be solved. Focusing on them and figuring out how APIs can help is a good strategy for a number of reasons.

First, when you uncover problems to be solved, you're tapping into an opportunity to make things better. When you can create an API that makes work easier, safer, faster, and so on, that's a win. Second, when you complete an API that solves a problem, it's easy to know if you reached your goal. The truth is that sometimes our APIs don't work as we expected and may not actually do a good job of solving the stated problem. When that happens, we need to learn from our attempt and then move on to design and build an API that *does* solve the problem. When you start with the problem, it's easier to measure your success.

And measuring success is how you make sure your API supports a viable business strategy.

Business Viable

As mentioned earlier, it doesn't make sense to invest a lot of time and money into an API no one really wants or needs. That's why "matching people's needs" is an important element of good API design. Another important element to

2. https://www.christenseninstitute.org/jobs-to-be-done

• Click HERE to purchase this book now. discuss

good API work is making sure the design and implementation work makes good business sense. This can be a challenge since it's common for API designers and developers to be a bit "out of the loop" when it comes to business details. In fact, I know many programmers prefer it that way. But in reality, all of us working on IT projects for our companies have a responsibility to make sure the work makes good sense both technically and business-wise.

While developers aren't often the ones who decide business strategy, we influence what time and money goes into executing on that strategy. In my experience, there's more than enough opportunity to improve the way companies use APIs in their business. In fact, the challenge is not in finding an API opportunity, the challenge is in prioritizing the many existing opportunities. And that's a place where developer teams and API designers can help.

Starting an API project often represents a risk. It requires a commitment of time, money, and personnel. Working on one project can mean *not* working on other API projects. And the more time and money your API project consumes, the less is available for other projects. For this reason, one of the best things you can do when designing APIs is to reduce the risk incurred. You can do this by building an API that matches people's needs *and* has enough flexibility built in to allow for small changes in the future without big costs. That's a lot to think about. But as time goes on, I've found most API developers get pretty good at balancing matching needs versus anticipating changes. We'll see examples of this trade-off throughout the API design and build we'll be doing in this book.

Basically, those who are tasked with designing and building the APIs—the glue that makes services work well together—have the job of uncovering important needs and figuring out how to implement the APIs in a way that makes sense both now and for the foreseeable future. We started on that path in Chapter 3, Modeling APIs, on page ?, by uncovering the job that needs to be done (the OnBoarding API), and now we need to turn what we learned into a clear, concise API design that developers can use to successfully build the final product.

To make that possible, we'll first use a repeatable method—an API design method—to convert our information into a design. Then we'll take the output of that method and produce a helpful diagram (a sequence diagram) that documents our design and provides developers with added information they can use to actually build the API we need.

First, let's look at the API design method.

• Click HERE to purchase this book now. discuss