Bad Software Architecture is a People Problem

**WHEN PEOPLE
DON'T WORK
WELL TOGETHER
THEY MAKE BAD
DECISIONS**

KATE MATSUDAIRA

It all started with a bug. Customers were complaining that their information was out of date on the website. They would make an update and for some reason their changes weren't being reflected. Caching seemed like the obvious problem, but once we started diving into the details, we realized it was a much bigger issue.

What we discovered was the back-end team managing the APIs and data didn't see eye-to-eye with the front-end team consuming the data. The back-end team designed the APIs the way they thought the data should be queried—one that was optimized for the way they had designed the schema. The challenge was that when the front-end team wrote the interface, the API seemed clunky to them—there were too many parameters, and they had to make too many calls. This negatively impacted the mobile experience, where browsers can't handle as many concurrent requests, so the front-end team made the decision to cache part of the data locally.

The crux of the issue

The challenge is that if you want your software to last, uniformity and predictability are good things—unique snowflakes are not.

was that the teams had not communicated well with each other. Neither team had taken the time to understand the needs of the other team. The result was a weird caching bug that affected the end user.

You might be thinking this could never happen on your team, but the reality is that when many different people are working on a problem, each could have a different idea about the best solution. And when you don't have a team that works well together, it can hurt your software design, along with its maintainability, scalability, and performance.

Most software systems consist of parts and pieces that come together to perform a larger function. Those parts and pieces can be thought out and planned, and work together in a beautiful orchestra. Or they can be designed by individuals, each one as unique as the person who created it. The challenge is that if you want your software to last, uniformity and predictability are good things—unique snowflakes are not.

One of the challenges of managing a software team is balancing the knowledge levels across your staff. In an ideal world, every employee would know enough to do his or her job well, but the truth is, in larger software teams there is always someone getting up to speed on something: a new technology, a way of building software, or even the way your systems work. When someone doesn't know something well enough to do a great job, there is a knowledge gap, and this is pretty common.

When building software and moving fast, people don't always have enough time to learn everything they need to

bridge their gaps. So each person will make assumptions or concessions that can impact the effectiveness of any software that individual works on.

For example, an employee may choose a new technology that hasn't been road tested enough in the wild, and that later falls apart under heavy production load. Another example is someone writing code for a particular function, without knowing that code already exists in a shared library written by another team—reinventing the wheel and making maintenance and updates more challenging in the future.

On larger teams, one of the common places these knowledge gaps exist is between teams or across disciplines: for example, when someone in operations creates a Band-Aid in one area of the system (like repetitively restarting a service to fix a memory leak), because the underlying issue is just too complex to diagnose and fix (the person doesn't have enough understanding of the running code to fix the leaky resources).

Every day, people are making decisions with imperfect knowledge. The real question is, how can you improve the knowledge gaps and enable your team to make better decisions?

Here are a few strategies that can help your team work better, and in turn help you create better software. While none of these strategies is a new idea, they are all great reminders of ways to make your teams and processes that much better.

One of the most important strategies is to think about how you will truly test the end-to-end functionality of a system.

Define how you will work together

Whether you are creating an API or consuming someone else's data, having a clearly defined contract is the first step toward a good working relationship. When you work with another service it is important to understand the guardrails and best practices for consuming that service. For example, you should establish the payload maximums and discuss the frequency and usage guidelines. If for some reason the existing API doesn't meet your needs, instead of just working around it, talk about why it isn't working and collaboratively figure out the best way to solve the problem (whether that is updating the API or leveraging a caching strategy). The key here is communication.

Decide how you will test the whole system

One of the most important strategies is to think about how you will truly test the end-to-end functionality of a system. Having tests that investigate only *your* parts of the system (like the back-end APIs) but not the end-customer experience can result in uncaught errors or issues (such as my opening example of caching). The challenge then becomes, who will own these tests? And who will run these tests and be responsible for handling failures? You may not want tests for every scenario, but certainly the most important ones are worth having.

When bugs happen, work together to solve them

When problems arise, try to avoid solutions that only mask the underlying issue. Instead, work together to figure out what the real cause of the problem is, and then make a decision as a team on the best way of addressing it going forward. This way the entire team can learn more about

how the systems work, and everyone involved will be informed of any potential Band-Aids.

Use versioning

When another team consumes something you created (an API, a library, a package), versioning is the smartest way of making updates and keeping everyone on the same page with those changes. There is nothing worse than relying on something and having it change underneath you. The author may think the changes are minor or innocuous, but sometimes those changes can have unintended consequences downstream. By starting with versions, it is easy to keep everyone in check and predictably manage their dependencies.

Create coding standards

Following standards can be really helpful when it comes to code maintenance. When you depend on someone else and have access to that source code, being able to look at it—and know what you are looking at—can give you an edge in understanding, debugging, and integration. Similarly, in situations where styles are inherited and reused throughout the code, having tools like a style guide can help ensure that the user interfaces look consistent—even when different teams throughout the company develop them.

Do code reviews

One of the best ways of bridging knowledge gaps on a team is to encourage sharing among team members. When other people review and give feedback, they learn

the code, too. This is a great way of spreading knowledge across the team.

Of course, the real key to great software architecture for a system developed by lots of different people is to have great communication. You want everyone to talk openly to everyone else, ask questions, and share ideas. This means creating a culture where people are open and have a sense of ownership—even for parts of the system they didn't write.

Kate Matsudaira is an experienced technology leader. She worked in big companies such as Microsoft and Amazon and three successful startups (Decide acquired by eBay, Moz, and Delve Networks acquired by Limelight) before starting her own company, Popforms (<https://popforms.com/>), which was acquired by Safari Books. Having spent her early career as a software engineer, she is deeply technical and has done leading work on distributed systems, cloud computing, and mobile. She has experience managing entire product teams and research scientists, and has built her own profitable business. She is a published author, keynote speaker, and has been honored with awards such as Seattle's Top 40 under 40. She sits on the board of acmqueue and maintains a personal blog at katemats.com.

Copyright © 2016 held by owner/author. Publication rights licensed to ACM.