54: Less Software

Keep your code as simple as possible

You'd think that twice as much code would make your software only twice as complex. But actually, **each time you increase the amount of code, your software grows exponentially more complicated**. Each minor addition, each change, each interdependency, and each preference has a cascading effect. Keep adding code recklessly and, before you know it, you'll have created the dreaded Big Ball of Mud.

The way you fight this complexity is with less software. Less software means less features, less code, less waste.

The key is to restate any hard problem that requires a lot of software into a simple problem that requires much less. You may not be solving exactly the same problem but that's alright. Solving 80% of the original problem for 20% of the effort is a major win. The original problem is almost never so bad that it's worth five times the effort to solve it.

Less software means you put away the crystal ball. Instead of trying to predict future problems, you deal only with the problems of today. Why? Fears you have about tomorrow often never come to fruition. Don't bog yourself down trying to solve these phantom issues.

From the beginning, we've designed our products around the concept of less software. Whenever possible, we chop up hard problems into easy ones. We've found solutions to easy problems are not only easier to implement and support, they're easier to understand and easier to use. It's all part of how we differentiate ourselves from competitors; Instead of trying to build products that do more, we build products that do less.

- Less software is easier to manage.
- Less software reduces your codebase and that means
- less maintenance busywork (and a happier staff).
- Less software lowers your cost of change so you can adapt quickly. You can change your mind without having to change boatloads of code.
- Less software results in fewer bugs.
- Less software means less support.

Which features you choose to include or omit have a lot to do with less software too. Don't be afraid to say no to feature requests that are hard to do. Unless they're absolutely essential, save time/effort/confusion by leaving them out.

Slow down too. Don't take action on an idea for a week and see if it still seems like a great idea after the initial buzz wears off. The extra marinating time will often help your brain come up with an easier solution.

Encourage programmers to make counteroffers.
You want to hear: "The way you suggested will take 12 hours. But there's a way I can do it that will only take one hour. It won't do x but it will do y." Let the software push back. Tell programmers to fight for what they think is the best way.

Also, search for detours around writing more software. Can you change the copy on the screen so that it suggests an alternate route to customers that doesn't require a change in the software model? For example, can you suggest that people upload images of a specific size instead of doing the image manipulation on the server side?

For every feature that makes it into your app, ask yourself: Is there a way this can be added that won't require as much software? Write just the code you need and no more. Your app will be leaner and healthier as a result.

There is No CODE That is More Flexible Than NO Code!

> *The "secret" to good software design wasn't in knowing what to put into the code; it was in knowing what to leave OUT! It was in recognizing where the hard-spots and soft-spots were, and knowing where to leave space/room rather than trying to cram in more design.*

—Brad Appleton, software engineer (from There is No CODE that is more flexible than NO Code!)

Complexity Does Not Scale Linearly With Size

> *The most important ruleof software engineering is also the least known: Complexity does not scale linearly with size...A 2000 line program requires more than twice as much development time as one half the size.*

—The Ganssle Group (from Keep It Small)

55: Optimize for Happiness

Choose tools that keep your team excited and motivated

A happy programmer is a productive programmer. That's why we optimize for happiness and you should too. Don't just pick tools and practices based on industry standards or performance metrics. Look at the intangibles: Is there passion, pride, and craftsmanship here? Would you truly be happy working in this environment eight hours a day?

This is especially important for choosing a programming language. Despite public perception to the contrary, they are not created equal. While just about any language can create just about any application, the right one makes the effort not merely possible or bearable, but pleasant and invigorating. It's all about making the little details of daily work enjoyable.

Happiness has a cascading effect. Happy programmers do the right thing. They write simple, readable code. They take clean, expressive, readable, elegant approaches. They have fun.

We found programming bliss in the language Ruby and passed it on to other developers with our framework Rails. Both share a mission statement to optimize for humans and their happiness. We encourage you to give that combination a spin.

In summary, your team needs to work with tools they love. We've talked here in terms of programming languages, but the concept holds true for applications, platforms, and anything else. Choose the fuse that gets people excited. You'll generate excitement and motivation and a better product as a result.

The kinds of engineers you want

> *The number one reason I wanted to build our app using Ruby on Rails is that it is so elegant, productive, and beautifully designed. It tends to attract the kind of engineers who care about just those sort of things...those are exactly the kinds of engineers you want on your team because they create the kind of beautiful, elegant and productive software you need to win the market.*

—Charles Jolley, Managing Director at Nisus Software (from Signal vs. Noise)

56: Code Speaks

Listen when your code pushes back

Listen to your code. It will offer suggestions. It will push back. It will tell you where the pitfalls reside. It will suggest new ways to do things. It will help you stick to a model of less software.

Is a new feature requiring weeks of time and thousands of lines of code? That's your code telling you there's probably a better way. Is there a simple way to code something in one hour instead of a complicated way that will take ten hours? Again, that's your code guiding you. Listen.

Your code can guide you to fixes that are cheap and light. Pay attention when an easy path emerges. Sure, the feature that's easy to make might not be exactly the same as the feature you originally had in mind but so what? If it works well enough and gives you more time to work on something else, it's a keeper.

Listen up

> *Don't worry about design, if you listen to your code a good design will appear...Listen to the technical people. If they are complaining about the difficulty of making changes, then take such complaints seriously and give them time to fix things.*

—Martin Fowler, Chief Scientist, ThoughtWorks (from Is Design Dead?)

If Programmers Got Paid To Remove Code...

> *If programmers got paid to remove code from sofware instead of writing new code, software would be a whole lot better.*

—Nicholas Negroponte, Professor of Media Technology at MIT (from And, the rest of the (AIGA Conference) story)

57: Manage Debt

Pay off your code and design “bills”

We usually think of debt in terms of money but it comes in other forms too. You can easily build up code and design debt.

Hack together some bad code that’s functional but still a bit hairy and you’re building up debt. Throw together a design that’s good enough but not really good and you’ve done it again.

It’s ok to do this from time to time. In fact, it’s often a needed technique that helps you do the whole Get-Real-ASAP thing. But you still need to recognize it as debt and pay it off at some point by cleaning up the hairy code or redesigning that so-so page.

The same way you should regularly put aside some of your income for taxes, regularly put aside some time to pay off your code and design debt. If you don’t, you’ll just be paying interest (fixing hacks) instead of paying down the principal (and moving forward).

58: Open Doors

Get data out into the world via RSS, APIs, etc.

Don't try to lock-in your customers. Let them get their information when they want it and how they want it. To do that, you've got to give up the idea of sealing in data. Instead, let it run wild. Give people access to their information via RSS feeds. Offer apis that let third-party developers build on to your tool. When you do, you make life more convenient for customers and expand the possibilities of what your app can do.

People used to think of rss feeds as merely a good way to keep track of blogs or news sites. Feeds have more power than that though. They also provide a great way for customers to stay up to date on the changing content of an app without having to log in repeatedly. With Basecamp feeds, customers can pop the URL into a newsreader and keep an eye on project messages, todo lists, and milestones without having to constantly check in at the site.

APIs let developers build add-on products for your app that can turn out to be invaluable. For example, Backpack supplies an API which Chipt Productions used to build a Mac OS X Dashboard widget. The widget lets people add and edit reminders, list items, and more from the desktop. Customers have raved to us about this widget and some have even said it was the key factor in getting them to use Backpack.

Other good examples of companies letting data run free in order to get a boomerang effect:

- The **Google Maps API** has spawned interesting mash ups that let people cull information from another source (e.g. apartment listings) and plot that data on a map.
- Linkrolls offer a way for people to get their latest **del.icio.us** bookmarks displayed on their own sites.
- **Flickr** allows other businesses access to commercial APIs so customers can buy photo books, posters, dvd backups, and stamps. "The goal is to open it up completely and give you the biggest variety of choices when it comes to doing things with your photos," says Stewart Butterfield of Flickr.

A Widget Makes the Difference

> *When Basecamp released Backpack a while back, my first impression was...eh.*
>
> *So it was around the time that Chipt Productions released a Backpack widget for Tiger — which was too cool not to download and try — that I gave Backpack a second look. The result? Quite a difference.*

> *Now whenever an idea hits, I pop up the widget, type, and submit — done. Email arrives with something I want to check out? Pop up the widget, type, and submit — done. The widget is an immediate brain dump readily available on each Mac I use. And because everything is web based, there isn't any version control or syncing — just the fluid input of content without having to be concerned about where it's going or how I'll access it later.*

—Todd Dominey, founder, Dominey Design (from Trying on Backpack)