# Make Better Software

A collection of interviews with makers about the craft of software development, hiring, and technical leadership

Volume 1

by Fog Creek & Friends

# We Had a Different Idea...

Fog Creek Software began in 2000, after our founders, Joel Spolsky and Michael Pryor, had trouble finding a place to work where programmers had decent working conditions and got an opportunity to do great work. At that time, developers were treated like typists, and yet companies still complained that they couldn't hire great software developers, and they struggled to make products people actually wanted to use.

So we came up with a different idea - what if we started our own business where we only recruited the best software people, we treated them well, and then got the heck out of their way?

Well, it worked. Since then we've been creating great software products that are used and loved by millions of people. These include:

FogBugz stackoverflow HyperDev Trello

# Let's Make The Future

We take a similar approach to our products too. Rooted in a deep understanding of the realities of building software, they're designed to help you get the job done. But otherwise, they get out of your way so you focus on what's really important - working together to make things. Whether that's making code work on Stack Overflow, organizing things in Trello, or building software with FogBugz or HyperDev.

And what we're making is the future. Far from just being typists, our whole world is now infused by the software that we're building. Software is waking us up at just the right time. It's hailing the cab and guiding us to our next meeting, and it's finding us a great place to eat at the end of a busy day. Our whole lives are informed by and run with software. And this is only the beginning. So let's make the future.

# The Interviews

## Part 1: Mastering the Craft

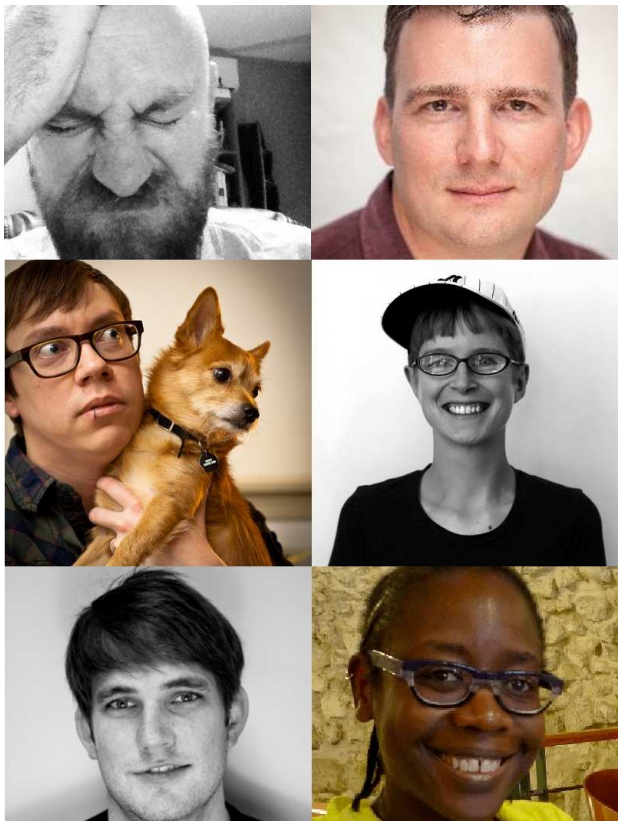


## Part 2: Hiring and Development



## Part 3: Technical Leadership

# The Other Stuff



## Ask the Experts

Sound advice from successful software developers



## Creeker Wisdom

Lessons learned at Fog Creek



# Foreword

If you follow developers around, here's what you'd see them doing. The first thing you'd notice is that about 5%, and I made that number up, 5% of their time is spent actually typing code into a computer. The other 95% is spent trying to figure out why the perfectly straightforward code that you just typed is not doing the perfectly obvious thing that you thought it should do.

This is why programming is hard. Programming is hard because computers are super fussy. Super literal. They need to have everything explained to them exactly, down to the smallest detail, and then they become maddeningly literal about doing exactly what you told them to do, especially if, it's not what you wanted.

Literally every other job is easy compared to programming.

I don't know, let's say you're a taxi driver. You get in the taxi, you turn the wheel to the right, the taxi goes to the right. Pretty much 100% of the time.

Imagine what it would be like if you turned the wheel to the right, and sometimes it goes to the right, but sometimes the radio goes on. Or maybe when you turn right out of the airport, the trunk pops open.

One day, you turn right, and the whole taxi literally disappears without a trace. Poof. And you're just sitting on your butt in the road wondering where the taxi went.

You don't know why this happened. Maybe it's because your passenger was wearing a shirt with stripes. And you will never, ever figure it out, because the taxi disappeared and the passenger disappeared.

This is what life is like every day for developers.

Now, how do we deal with this? Besides going crazy. Because I've known some high tech companies that literally hire staff psychiatrists.

The way you deal with this is that you ask people for help. There's tons of knowledge locked away in people's heads that can help you, and that's where this magazine comes in.

We've collected together some of the best interviews with smart developers, about software engineering, hiring developers and technical leadership. They've found ways to solve problems and hold back the crazy, at least for a little while.

So go, read, and help stave off the madness of software development for at least another day. Maybe two. You're welcome.

JOEL SPOLSKY
**Co-Founder of Fog Creek,**
**CEO of Stack Overflow and**
**Co-Founder of Trello**

# Mastering the Craft

Part 1

# How to Maximize Your Impact

## Lessons Learned from Silicon Valley's Software Engineering Leaders

> ***If you pick up the right habits, techniques, and mindsets when you're starting off, then you're significantly more likely to succeed***

### Start Out Right

Lau says that something he noticed early on in his career that was reinforced after speaking with engineering leaders is that, "if you pick up the right habits, techniques, and mindsets when you're starting off, then you're significantly more likely to succeed". This is most obvious when you look at the career progression of junior developers in organizations with and without effective onboarding programs, says Lau.

When starting at Quora early on, Lau says "it was sort of luck of the draw whether you would have a good mentor, or whether someone would explain to you core concepts in your first day or your first week or your first month. People who got lucky were much more likely to succeed and much more likely to be effective many months down the line". So Lau set work on revamping their onboarding to level the playing field.

### Edmond Lau

Edmond Lau is an experienced Software Engineer who has worked for the likes of Google, Ooyala, Quora, and Quip. He spoke with engineering leaders from companies like Twitter, Facebook, and Stripe, whilst writing his book 'The Effective Engineer'. We interviewed him to understand what he learned about maximizing impact as a software engineer.

## The Importance of Onboarding

"Onboarding is not something where you do it once and you're done. It's something you do very incrementally," explains Lau. "Start by documenting questions that new engineers are asking and make them reusable resources. If you are a new engineer, then as you are onboarding identify the things that are giving you problems. What is taking a long time to do that really shouldn't? Over time, you can build up a repository of knowledge that becomes much more useful for new engineers joining the company."

As an example of this Lau describes how things work at his current company, Quip. "Any time we are building new products we end up writing a short design doc that is then shared with the team" and then "everybody gets this new piece of information" and if "someone ran into an issue we document that into some best practices." What this means is that over time "we end up with lots and lots of documents that are really useful for new engineers."

## Common Themes

Having spoken to many engineering leaders about what worked well in their careers, a number of common themes emerged. Lau says that one such theme was "focusing on and investing in tooling". For example "when I talked with Bobby Johnson, who is the former Director of Engineering for Infrastructure at Facebook", Lau recalls, "one of the biggest time and investments that he made that had the biggest return was just building tools. Trying to automate as much of the mechanics of what they were doing as possible. Every time they got a problem and found that they were still repeating what they were doing, they would write a tool for it and automate it."

> ***"Every time they got a problem and found that they were still repeating what they were doing, they would write a tool for it***

Another theme was "just keeping things simple… when I asked them what's the biggest lesson or the most valuable lesson you've learned in the past year, they would say they just sort of made things a little too complex" and that things didn't work when "there were a lot of operational burdens to keep things going."

## Optimizing for Impact

Keeping things simple and automating repetitive tasks are great examples of work which optimizes the impact you can have. This is a key part of Lau's approach to engineering and is what he terms Leverage, and focusing on this helps engineers identify activities that have the most impact.

"Leverage basically means what is the output you're producing for the amount of time you're investing? It's a metric of the impact you're generating per amount of time that you spent." So to be "an effective engineer you want to focus on the high leverage activities." This might seem obvious, but as Lau explains it can be hard to actually internalize — "This is something that took me many years to learn… I used to work 70 to 80-hour weeks because I thought that was how hard a team had to work". But after "we built an analytics module for a client and they never used it, or when we would build a product and it would never get the user adoption that we wanted… those experiences made me feel like working hard, putting in those hours just isn't enough." Lau says that what he then learned to do was "prioritize the things that are actually generating a large impact for the amount of time you're spending."

So, how do you know what to focus on? "The key" explains Lau, is in "setting up feedback loops, and that applies to everything you do. For instance, when you're writing software, it means you send around a lightweight design doc to get feedback on your design before you even write any code. When you're writing code it means… sending your code to code reviewers that are the strictest. The ones who would do a good job of critiquing what you're doing. If you're working on a product it means… getting feedback from gatekeepers who sort of block or launch as soon as possible. Then when you're

building or iterating a product, it means running A/B tests to measure if this change is actually materially improving the product in some way."

## Metrics to Measure Impact

To monitor and gauge this feedback you want to "pick a metric that incentivizes the type of behavior that you want". Lau gives the example of a Performance team looking to optimize the performance of their product — "the metric that you want to improve could be something like the 99th percentile of all load times. Or it could be the average load time. Either of those metrics would improve the performance of your product, but they have very different outcomes. If you're improving the average load times, that would tend to make you focus on general server improvements, but if you were focusing on the 99th percentile… then you're focusing on the long tail of the application." So it's important to consider the metric you choose carefully, as "the choice of metric affects the type of work that you focus on," Lau says. So "pick the metric that is most aligned with the success of the business."

## Developing as a Software Engineer

Lau attributes success in his career to having purposefully sought out job opportunities where he could continue to learn new things and push himself. Without doing this, Engineers can end up "being stuck on a plateau where they're not really challenging themselves, they're not really learning day to day. Just from my own experience, when I was at Google, it was a very comfortable place and sort of after my first six months there I learned a ton… but after two years I felt like I wasn't really challenging myself anymore," explains Lau.

Another way he has developed himself is through meet-ups and swapping stories with other developers. Lau comments that "when you're working on a team, a lot of times the lessons we learn are just the ones that occur during the projects that you work on. People who have either had similar experiences or have gone through other projects… can teach you a ton."

---

Read more from Edmond at: [www.theeffectiveengineer.com](http://www.theeffectiveengineer.com)

> ***"Prioritize the things that are actually generating a large impact for the amount of time you're spending***

# More Effective Code Reviews

The key elements to a code review culture

> ***It's much more helpful to focus on the cultural benefits of code reviews***


Derek Prior

Derek Prior is Development Director at Thoughtbot, where they have a strong code review culture. In this interview, Derek shares the benefits of code reviews and the essential elements he thinks you need to make them effective.


When it comes to the benefits of code review "everybody's natural reaction is to say that they catch bugs. That is true. If I have code reviewed that's going to have fewer bugs than code that isn't reviewed. But I think that that puts too much importance on the 'finding bugs' part," says Prior. "It's much more helpful to focus on the cultural benefits of code reviews," which for him are many. From "sharing knowledge with each other, or keeping up with what everybody's doing" to "knowing what's going on in that other part of the code base and finding a really interesting alternative solution to a problem that they hadn't thought of."

## More Than Catching Bugs

For Prior, the benefits of code review go way beyond simply defect finding — "finding defects is actually, frankly, really hard. A lot of times I'll talk to people about code reviews, and they'll say, 'well we did code reviews, but we still had all these bugs'. But they're not going to catch all the bugs. You're looking at the diff, and to know exactly how that's really going to impact your system, you have to know the entire system". So whilst "code review is great for defect finding, it's not a panacea. Instead, I think it's much more helpful to focus on the cultural benefits of code reviews", says Prior.

## Make Them Part of Your Workflow

One way to make sure that once you get started with code reviews, you stick with them is to "take a really, lightweight approach to it," recommends Prior. For instance, "when I

finish up a PR, I will paste it into Slack, or whatever. We'll paste that in and say, 'can I get a review for this please?' and generally, that's enough, with the way we work, to get your code reviewed within the next few hours," Prior says.

A fast turn around on code reviews is important, and it doesn't have to be burdensome to stay on top of them. Prior suggests that there are in fact "a lot of natural breaks throughout your day that you can work these in". "For me, I come in in the morning" and catch up on any code reviews from overnight. "Then, right before lunch, or in the afternoon, if I'm going to take a coffee break… I'll look then". "There's plenty of times I find, that I can just work these in naturally. There's no need to schedule them. I've worked with teams that try to schedule them, or try to say, 'this person is going to be the one, that's going to be chiefly doing code reviews this week', and I've never seen that work particularly well".

Instead, you just need to "keep it lightweight and friendly," says Prior. "The biggest thing that makes this work is, keeping small, discrete pull requests, that are going to be much more easily reviewed if you provide some excellent context. That's the 'why' you're making this change, not necessarily the 'what'". But "ultimately, the big secret to this", confesses Prior, "is that most of these code reviews only take me five minutes. It's not a big commitment."

## What to Cover in a Code Review

"Part of what I think makes code review so great, is that it's a great place to have a technical discussion about your actual software. Rather than in the abstract," says Prior. He doesn't subscribe to the idea of using a checklist or specific items, suggesting a more relaxed approach — "just look at things that interest you about a change". "Maybe you just got finished reading this great book on design patterns. That's valuable. You're going to teach somebody something. Or maybe you're really interested in web security… or accessibility." For me, says Prior "I harp on naming a lot… I look for test coverage." Whatever it is, so long as you have "a good blend on your team, of people who are interested in different things" then you "can all learn from each other."

> ***"It really helps if you've been making several small commits along the way***

## Context is Key

"The number one thing is context," says Prior. "When you're submitting a pull request, you've been working on this thing for four hours, or eight hours, or two days, or sometimes even a week, or whatever the case may be. You have a lot of context built up in your head. Some things seem really obvious to you at this point". The thing to remember though is that "to the person reviewing it, they weren't there… they don't have that context". "It really helps if you've been making several small commits along the way, where you're describing what was in your head," but regardless, "as you prepare the change… make sure you give a nice description of everything."

This description should answer the question "Why are we changing it?," says Prior. Other considerations are "why is this the best solution? What other solutions did you consider? What problems did you run into? Is there an area of the code you're really unsure about?" These will all "help you get a much better review, by setting up everybody else to be on the same page as you", Prior says.

## Keep Reviews Positive

Prior warns that "when code reviews aren't done particularly well, or are overly critical, they can lead to resentment among

the team". One thing that you must be aware of though is that written communication is prone to negativity bias, says Prior. "If I'm talking to you and I give you some technical feedback, and I say, 'oh, why didn't you use this pattern here', you're going to perceive that in one way. But if I say that same exact thing written down, it comes off as more harsh. You're going to perceive that more negatively. It's much more subject to your particular mood. I can't influence it with the way I say something. It's just a fact that, the same feedback written, is going to be perceived more negatively". So this is something that should be considered in code review too.

Prior recommends an excellent way around this is to "give feedback in a manner that's more of a conversation. What I like to do is ask questions… Instead of saying, 'extract the service object here to reduce some of this duplication', I would say, 'hey, what do you think about extracting a service object here, to eliminate this duplication?' They're very similar comments, but now I'm opening it up to a conversation, by asking you a question". Another suggestion Prior makes is "clarifying how strongly you feel about a piece of feedback."

Nevertheless, from time to time conflict will still occur. But for Prior, "having conflict in your code reviews like this, is actually really beneficial… as long as they're the right types of conflict and nobody feels bad about them afterwards. If you're agreeing with your teammates all the time, and nothing interesting is happening in your code reviews, you basically have a monoculture. You want everybody bringing their own experiences, and their own expertise, and sometimes those are going to clash with each other". But if you can "handle those conflicts properly", says Prior, then that's "how everybody on the team is going to learn."

> ***"Having conflict in your code reviews like this, is actually really beneficial***

---

Derek co-hosts the Bike Shed podcast: www.bikeshed.fm

# *Creeker Wisdom*

"You have to learn to ask for what you want. Part of your manager's job is to have conversations with you about career development, your salary, etc. So don't be afraid to broach those subjects, even if you're at the bottom of the food chain, and know your value. Work hard, and be your own advocate!

… And if you're working for someone who doesn't value you, leave. Immediately. No job is worth feeling bad for every day. The odds are you won't do good work in that environment anyway. Take your toys and go, something better awaits you!"

- Allie Schwartz, VP People at Fog Creek

# Selecting Software Development Metrics

How to pick the right metrics for your team and workflow

"I don't actually find metrics to be an interesting topic," says Nicolette.

"But I think it is a necessary thing. It's part of the necessary overhead for delivering. If we can detect emerging delivery risks early, then we can deal with them. If we don't discover them until late, then we're just blindsided and projects can fail".

Through his work Nicolette has worked with countless technical managers, but he's observed that "a lot of managers, team leads and people like that, don't really know what to measure". What usually happens is that "when people adopt a new process or method", usually one of two things happens, says Nicolette. Either, they're "only using the process in name only, or they're trying to use it but they're not used to it yet, and the metrics don't quite work." Or despite changing methods, they continue to measure in the same way—"people tend to use the measurements that they're accustomed to. They've always measured in a certain way, now they're adopting a new process. They keep measuring the same things as before." So the metrics don't quite match the process and they wrongly interpret a problem with the process itself.

Another common mistake Nicolette has noticed with development metrics, is that "people either overlook or underestimate the effects of measurement on behavior. We can be measuring something for some objective reason, but it causes people to behave differently because they're afraid" that it will negatively impact their "performance review." So Nicolette says, "I think we have to be very conscious of that. We don't want to drive undesired behaviors because we're measuring things a certain way. That not only breaks morale, but it also makes the measurements kind of useless too when they're not real."

## Factors to Consider When Selecting Metrics

Nicolette suggests that when it comes to metrics, all we really need to do is "look at the way work actually flows in your organization and measure that." Now, of course, that's easier said than done. But he asserts that there are really just "three factors to judge which metrics are appropriate."

> ***"People either overlook or underestimate the effects of measurement on behavior***

"The first factor is the approach to delivery," says Nicolette. There's traditional, which is where you "try to identify all the risks in advance… lay out a master plan, and you follow

Dave Nicolette

Dave Nicolette is a consultant specializing in improving software development and delivery methods. We spoke to him about software development metrics—the factors to consider when selecting metrics, examples of useful metrics as well as common mistakes made in applying them.

> *Look at the way work actually flows in your organization and measure that*


that plan." Or there's another type, adaptive, where "you start with a direction and an idea of how to proceed. Then you solicit feedback frequently so you can make course corrections." According to Nicolette, the delivery type, "traditional versus adaptive… has the biggest impact on which metrics will work."

Nicolette says that "the second factor to look at is the process model." "Nobody does anything in a pure way, but if you boil it down I think there are basically four reference models we can consider. One is linear… the canonical steps that we go through from requirements through to support. He says that "the next one would be iterative, in which you revisit the requirements multiple times and do something with them. The third one that I identify is time-boxed. It's really popular nowadays with processes like Scrum and so on. The fourth one is a continuous flow. This is popular now with the Kanban method, and it's also being adapted into Scrum teams quite a lot. And it's where we're really interested in keeping the work moving smoothly."

But Nicolette is a pragmatist and knows that any "real process is going to be a hybrid of these, but it's been my observation that any real process will lean more toward one of those models than the others, and that'll give us some hints about what kind of metrics will fit that situation," he says.

Lastly, the third factor Nicolette says impacts which metrics are relevant "is whether you're doing discrete projects or continuous delivery". "I look at those three factors, and based on that you can come up with a pretty good starter set of things to measure, and then you can adapt from there."

## Example Scenarios

> ***If we can detect emerging delivery risks early, then we can deal with them. If we don't discover them until late, then we're just blindsided and projects can fail***

To understand how these factors can be applied, here are some example scenarios, detailing the relevant metrics.

**Metrics for Agile Teams**
*Scenario: An agile team, working in short sprints, who are struggling to ship a new product.*

Nicolette says, that "if they're using Scrum basically correctly, they could probably depend on the canonical metrics that go with that, like velocity and your burn chart. You might look for hangover, that's incomplete work at the end of the sprint. You might look for a lot of variation in story size, when you finish a story in one day and the next story takes eight days." Interestingly, "you can always use lean-based metrics because they're not really dependent on the process model… so they could look at the mean cycle times as well as the variation in cycle times and get some hints about where to go from root-cause analysis. Metrics don't tell you what's wrong, but they can raise a flag," says Nicolette.

**Metrics for Hybrid Waterfall-Agile Teams**
*Scenario: A hybrid waterfall-agile team working on a long-term project, wanting to know what's possible to deliver by a certain date.*

"To know what's possible to deliver you can use a burn chart, burn up or burn down as you prefer," suggests Nicolette. This would help them see "according to their velocity, how much scope they can deliver by a given date. Conversely, you could see by what date approximately they could deliver a given amount of scope. It depends on what's flexible. In this kind of a project, usually neither is flexible, but at least you can get an early warning of delivery risk."

"One thing that might surprise some people," says Nicolette, "is the idea that agile methods can be used with traditional development. We need to decouple the word 'agile' from 'adaptive' because quite often it is used in a traditional context."

**Optimizing Kanban Processes**
*Scenario: A team wanting to optimize their working practices to stay on top of a bug queue.*

Typically "you want to be somewhat predictable in your service time," says Nicolette, "so that when a bug report comes in people have an idea of when they can expect to see it fixed." To be able to calculate this "you can use empirical information from past performance with fixing bugs, and your mean cycle time will give you approximately how long it takes to fix one."

However, "I like to use the Kanban method for these kinds of teams because it defines classes of service. You'll find that every type of bug doesn't take the same amount of time to fix," cautions Nicolette. So "based on your history, pick out different categories. You can identify the characteristics of particular kinds of bug reports that tend to fall together, and you can track cycle times differently for each of those classes of service."

**Rolling-up Metrics for Development Management**
*Scenario: A CTO who wants to monitor how teams are performing and ensure code is of high quality.*

Nicolette recommends in order to measure "how the teams are performing, you need measurements that are comparable across teams and comparable across projects." To to this end, he recommends "tracking throughput, cycle time" and "process cycle efficiency" as "those roll up nicely." However, he warns that "some other metrics don't roll up so well" — "like agile metrics, particularly velocity, which is really different for each team".

"The other question about code quality I think should be a team responsibility. Then they can use metrics, and static code analysis tools to help them spot potential quality issues. But I wouldn't share details outside the team, because then members will feel like they're getting judged on that and they'll game the numbers."

---

Read more from Dave at:
[davenicolette.wordpress.com](davenicolette.wordpress.com)

# Go Beyond Code to Become a Better Programmer

**Many of us want to become better programmers, but how exactly do you go about doing that?**

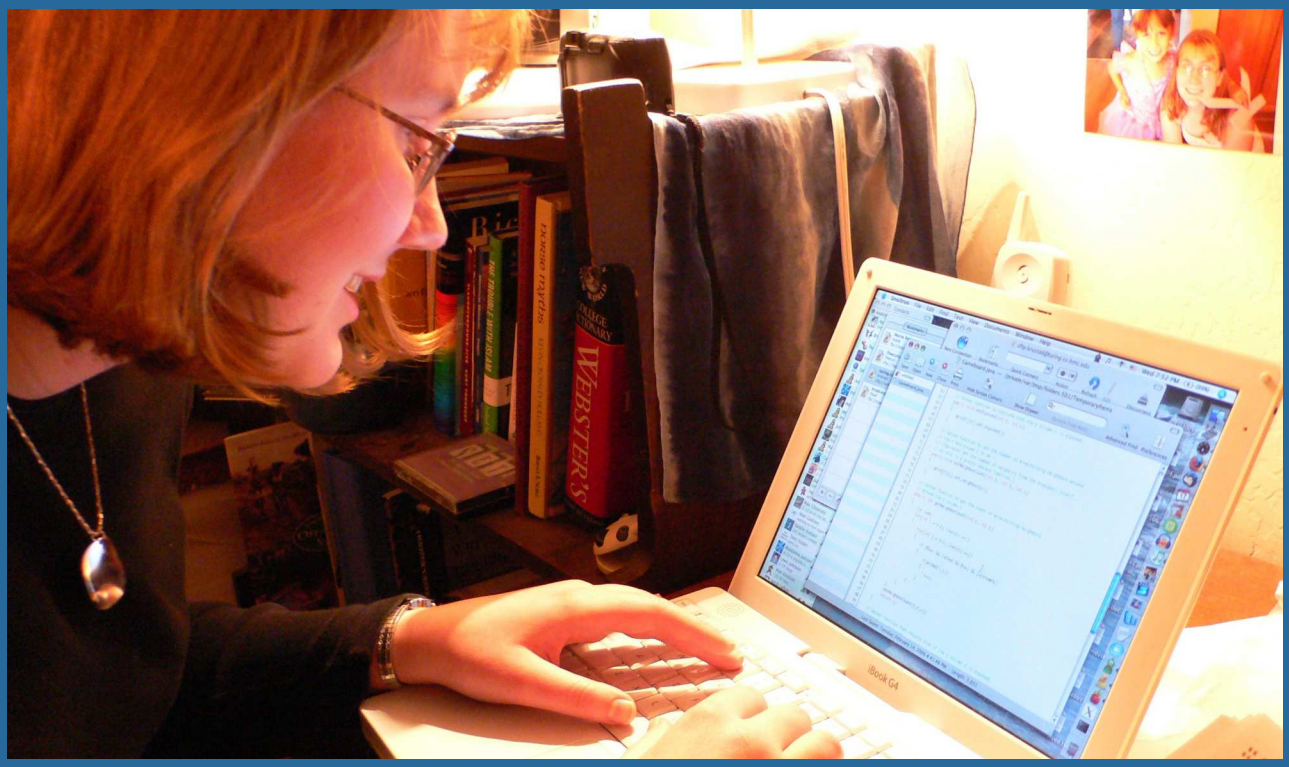



Photo by **Skyfaller**

So where to start? For Goodliffe, it starts with having the right attitude — "The standout difference between the really good coders that I've worked with and the ones that aren't so great, is attitude," says Goodliffe.

"It's not a hand-wavey thing. I've worked with people who know technology, who know the idioms, and how to do all this stuff. But, if they don't have the right attitude, they're just not effective programmers and they're not great people to work with. The kind of stuff I'm talking about here is humility. You don't want to work with people who think they know it all but don't. Being humble is the key thing. It doesn't mean that's an excuse to not know stuff, but it's just not believing that you're better than you really are".

You can develop this humility by "being in a state of constant learning… so looking for new stuff, absorbing new knowledge, wanting to learn off of other people," explains Goodliffe. "It doesn't necessarily mean be a perfectionist and wanting to make everything perfect before you ship. It's doing the best you can in the time you have, with the resources you have. But that kind of attitude really drives through to great code".

## Write Less Code

Another way he suggests that you can improve your code is to write less of it. "It seems kind of counter-intuitive for a coder… but it's entirely possible to write thousands of lines of code and achieve nothing in it. Think about it — no unnecessary logic, don't write stuff that doesn't need to be said, don't write verbose code. Sometimes you can stretch out boolean expressions into massive If statements

### Pete Goodliffe

Pete Goodliffe is a programmer and software development writer who has authored two books, 'Code Craft' and 'Becoming a Better Programmer'. Both provide advice to Software Developers on how to hone their skills and become better programmers. We interviewed him, and he told us why you have to go beyond code to become a better programmer.

which just hide what is being said". Ultimately for Goodliffe this means writing "simple but not simplistic code. If you don't write enough code, it doesn't do what it's supposed to do. But by avoiding all points of needless generality, like making abstract interfaces, or deep hierarchies that don't need to be extended," then you can write much more effective, concise code.

## Communicate Effectively

Goodliffe is also adamant about the need for programmers to communicate well. "Code is communication. You're communicating not just with the computer, you are communicating to other people writing the code. Even if you are working by yourself, you are communicating with yourself in two years time when you pick up the same section of code". And this is something that needs to be kept in mind, says Goodliffe. "It's a skill, and it's something you learn, and it's something you need to consciously practice. And yet, I don't know of courses… that really focus on something that's really quite an important skill for programmers," and so it can often be overlooked.

Of course, we aren't all the same and our aptitude for communication differs. But it's important to recognize your communication strengths, suggests Goodliffe. "Some people can talk well, some people are shy and retiring, but that doesn't necessarily mean you are stuck like that. That doesn't necessarily make you a bad communicator. Some people communicate better in different media and it's worth bearing that in mind. Some guys are really great on email, they can write concise, clear descriptions, they can follow a line of argument by writing and explaining something really well. Other people struggle to put it together in words". So he suggests that you should "learn how you communicate well, play to your own strengths and pick the right medium".

## Handle Complexity

"The reason people pay us to write software, unless we're doing it for fun, is because there's a complicated problem that needs to be solved," says Goodliffe. "There is some necessary level of complexity in software engineering and we have to embrace and understand that". Yet many of us fall into the trap of writing code that produces unnecessary complexity. So Goodliffe suggests that what we should always aim for is to produce code that "when you look at it, it looks obvious. That is the key hallmark of some excellent code… you know it wasn't simple to write. But when you look at it, the solution's simple, the shape is simple. All I can say is, that's what we should strive for".

Sometimes though, this is taken out of our hands and we're landed into the middle of a messy codebase. When this happens, "the most important thing… is to ask people," advises Goodliffe. "I see so many developers who just won't sort of swallow their pride and say, 'I don't quite know what this is doing, but I know Fred over there does. I'll just talk to Fred about it.' It's often those little bits of insight that can give you a super-fast route through something intractable," Goodliffe says. What's more, all too often we "pick up some code, look at it, and think 'that's a bit dodgy isn't it… what were they thinking?" What we forget though is this happens with our code too. "Somebody else picks it up, and they'll make that same judgement call on my code," says Goodliffe. So what we need to remember is that "what I might think is some terrible hack, is probably some pragmatic thing they did for a very good reason." After all, "nobody goes out of their way to write messy code… I can't say that I've found anyone that really tried to ruin a project. So approach code with that attitude," cautions Goodliffe.

## Sit at the Feet of Great Coders

Regardless of our own skills though, Goodliffe says that "the biggest thing for me… is to sit at the feet of great coders. I have learned the most in my career when I have been around excellent people who I can learn off of. Whose skills can rub off on me and I have moved jobs. I have moved physically to be able to work with those people". This can be motivating, and it's important to keep stoking our passion for programming. "I'm enthusiastic. I love this stuff," says Goodliffe, and "if you have an enthusiasm, that passion for programming, it tells out in the code that you write".

---

Read more from Pete at:
www.goodliffe.net

# Expert Advice for New Developers

We asked some old pros their top tips for those starting out in software development

"Work on a bunch of personal projects and contribute to Open-Source. Consuming information is one thing, but if you don't build things you won't remember it."

Brian Bondy, Co-Founder at Brave, prev. Senior Engineer at Khan Academy and Mozilla

*"Write code you don't know how to write, tackle problems you don't know how to solve. More importantly, learn things you don't want to learn. If it looks boring or if it looks too hard, study it."*

Dusty Phillips, Software Engineer at Facebook

"Don't fragment your forces into too many pieces. Focus on the intersection of things that at the same time you find valuable, and many people find valuable."

Salvatore Sanfilippo, Creator of Redis

*"Don't try to learn without a purpose. The desire to learn a new skill is driven by curiosity. Try to build something, find a problem, and learn only to solve it."*

Dayle Rees, Prev. Head of Engineering at JustPark, now at Crowdcube

*"Figure out what you're passionate about and do it. That might sound obvious, but focus and persistence are important for success, and both require passion."*

Lindi Emoungu, Senior Software Engineer at Google

"Get involved with the developer community: contribute to an Open-Source project, answer questions on StackOverflow, join a local User Group, etc. Constantly engaging with other people will help push you to grow as a developer."

Jared Parsons, Principal Developer on the C# Language Team at Microsoft

*"Be rigorous! When I was younger, I thought the only thing that mattered was having a working project. That attitude slowed me down because it kept me from understanding some things deeply."*

Mary Rose Cook, Makers Academy

"Good developers will quickly reach a point where just writing correct code is not enough; they've got to teach others how to use it, or convince them that the change is correct. Communication is key."

Eric Lippert, Software Engineer at Facebook and author of 'Essential C#'

# Hiring and Development

Part 2

# How to Onboard Software Engineers

## The essential elements to effectively onboard developers

There are 2 ways to get great engineers at your company. You can steal them or you can make them." says Heddleston. "In this day and age, you'd probably better have outlets for both", but you should at least have "a sustainable program of bringing on junior engineers".

### The Benefits on Onboarding

"We spend a huge amount of money recruiting and sourcing engineers" and then "we pay them huge sums of money to work" for us. But all too often when it comes to onboarding these valuable engineers, the approach is just — "you'll figure it out," says Heddleston.

The result is that "we're underutilizing people, which is expensive for companies, and people are unhappy when they aren't fulfilling their potential and that can lead to attrition". The answer to this problem for Heddleston is onboarding. "The return on investment is incredible… you get so much more out of employees who are happy and productive and feel integrated into the team."

### Getting Started with Onboarding

"The goal with onboarding is what I call reliable independence", says Heddleston. This is when "someone is able to reliably and independently build software on your team. For someone who is really senior, that might take 2 weeks… for someone really junior, that might take more like 6 months."

However, according to Heddleston, the time-frame for reliable independence "varies hugely depending on the size of the company and the quality of their internal tools." A common issue among fast-growing startups, for instance, is that "everyone has to come in and manually set up everything… and that's just going to bottleneck your company." So before

you even start to think about an onboarding program, first take stock of your tools. Heddleston recommends that new hires "shouldn't spend a lot of time having to do all these installations that you do once and that have no learning value".

Once your tooling is sorted, "the second thing is to put together a Trello board and come up with some goals of what you want to see. You can section it basically by the rough seniority level of someone coming in: senior, mid-level, junior". This helps because "someone who is junior is going to need a little bit more hands-on attention and someone who is senior is probably going to want freedom earlier. Then just set up goals of what you want to see them doing in the first day, the first week, the first month."

So an example goal that Heddleston suggests is "being able to ship something on the first day". "This new engineer comes onboard and in their first day, they actually push something to production. Even if it's just a small bug fix or… some config files that you might need for something, it's a really nice thing to feel like you can contribute on your first day."

Beyond that, goals should fall into what Heddleston says are the "3 major categories that people need to develop in order to become reliably independent. They're each about a third of what someone needs to know. We focus a lot on technical knowledge… but another third is company structure, the internal tools that you have, the way that you build, the way that your code is set up." The final third "is personal development, things like confidence, the ability to research problems, the ability to debug independently and judgment."

Your program should also take into account the fact that "people are going to come in stronger in different categories," says Heddleston. "Everyone is going to come in not knowing that much about your internal company structure, but some people might have more confidence, more debugging skills. Some people might know a lot more about the technologies that you use". So the goals should be flexible and focus on "filling in the gaps in the areas that they aren't as strong in."

## Who Should Onboard New Hires?

Who is involved in onboarding new hires is often an area where mistakes are made. According to Heddleston, often "the common first approach to onboarding is to place new employees with really senior mentors, but mentoring is actually really hard. It's a lot like teaching in the sense that it's very emotionally draining." What happens then is that you can "burn out all your senior mentors". So instead she recommends that you "spread out the load, and instead of pairing every junior who comes

Kate Heddleston

Kate Heddleston, an independent Product Engineer, provides us with advice about onboarding new developers. In this interview, she covers the benefits of onboarding, its essential elements, areas to focus on when getting started and common mistakes made.

> ***"There are 2 ways to get great engineers at your company. You can steal them, or you can make them***

in with a super senior engineer, you pair them with the last people who joined". Heddleston says this is a much better approach anyway, because "the best person to teach something is usually the last person who did it."

What's more, "really senior people are not necessarily that great at teaching junior people. They've forgotten what it was like to learn things for the first time so it can be really painful. It's nice to have the intermediate people turning around and teaching because they grow a lot."

Regardless of who is involved, one pre-requisite that Heddleston recommends is "executive level sign-off" — "there's nothing worse at a company than fighting a Director of Engineering" who doesn't believe in the goal.

## Creating a Learning Environment

When setting goals you should understand that "one of the tenets of expertise is the ability to recognize boundaries and scope really well. Whereas, one of the tenets of being a beginner is that you cannot recognize boundaries and you are unable to scope problems". So Heddleston cautions against making the mistake of "expecting a junior engineer to be really good at scoping a feature. That's one of the skills that they have to learn. Whatever you give them to do, just scope it. Then let them go play. Give them a feature that's really well defined, that has a clear area where they're working on and then let them go and fumble around with it."

"The final thing for junior engineers, and beginners, in general, is helping to bolster confidence," says Heddleston. "People think that confidence follows skills, but it's usually the other way around where skills follow confidence. If someone feels good about what they're doing, they're more likely to explore it and ask questions and to believe that they're able to solve the problem".

---

Read more from Kate at:
www.kateheddleston.com

# *Creeker Wisdom*

"When I started in tech there was a notion that the way to be great was to lock yourself in your parent's garage until you emerged with a unicorn – the one perfect thing that the world really needed, but hadn't yet realized was even a possibility. That worked for a few people but, by in large, money and talent were more often squandered than transformed by following this course.

It took me a long time to come around to the notion that the method for developing life-altering tools might have at least as much science as art in it. To be sure, there is plenty of that ineffable stuff that makes art and innovation, in building great products, but the pure light of inspiration is no substitute for being methodical about your process and open to the reality depicted in your results."

- David Miller, VP Product at Fog Creek

# How to Find, Hire, and Retain the Best Developers

> *Understand that they have lives, and the main point of their lives is not to build you the next 'Uber for Penguins'*

Cal Evans

Cal Evans has been a developer for more than 35 years. A well-known PHP community leader, he's experienced in building and managing development teams and is the author of 'Culture of Respect'. We interviewed him about finding, hiring and retaining developers.

To hire the best developers, you first have to know exactly what you're looking for. For me, says Evans, "the number one thing that I look for is competency. If you're not competent then you're going to be a hindrance to the team."

Then next is culture fit. "These days, you have to be very careful when you say cultural fit because everybody says you're only looking for 'white guys that graduated from your college'." But Evans explains that this isn't what he means, "cultural fit does not mean that you look like me, it does not mean that you think like me. What cultural fit means is that I've got a team that is currently working together and you've got to be able to play a part on that team." This is paramount for Evans because as he sees it, "building that team culture is so important" as is "keeping the team cohesion".

"I've seen great teams make a bad hiring decision and totally destroy the culture of the team and the momentum of the team. Just because they got one person who dragged the whole team down."

## A Community Approach to Hiring

So once you've settled on what exactly it is that you're looking for, you need to consider how you're going to find that person. Evans suggests that if that's finding a developer in the local area, then "the first thing is to get involved in your local community. That means start attending your local developer groups on a regular basis. Go 5 or 6 months to build your relationships before you start doing any real recruiting. Even then, once you start, don't just walk in and say 'I'm here, I'm collecting Resumes', but be the sponsor — buy the pizza and the beer…" It's important to get this right,

because “your local community is where you’re going to find your best resources.”

## Writing Great Job Ads

When it comes to writing a job ad, Evans stresses the need to put yourself in the developer’s position. When a developer is looking for a job, what they want to know is “‘can I afford to take the job? am I qualified for the job? how much does the job pay?’ That’s the important stuff. If you nail those, then you have a good chance of finding the developer that you’re actually looking for.” So, “number one, and I cannot stress this enough”, says Evans, “is put a salary or salary range on the job ad. You’re not limiting yourself… you’re not giving away any secrets because I guarantee that if you’ve already got developers, they’re talking to their buddies and we know how much you’re paying.” Then, it’s equally important to only put the skills that you really need on the ad. The tendency for some is to list everything they can think of, but if Evans sees that then “that’s a red flag immediately. What it tells me is that you don’t understand what you want, so you’re just throwing everything out there”.

## Hiring the Right Developers

Once you’ve got some applications for a position coming in, then you’re going to want to make sure they’re a good fit. Evans explains the approach he takes to make sure the team can work well with a candidate. “I get the entire team that the person’s going to be working with in one room and I let the team ask questions until everybody is out of questions”. He does this regardless of the level he’s hiring for — “I don’t care if you’re Junior, Senior, an Architect or whatever, you’re going to get to ask any question you want of this person, as long as it’s a legal question”.

Another key aspect of this approach is that as the team manager, he doesn’t ask any of the questions. “My job is to sit back and see how the team interacts with this person and how this person interacts with the team. If I see a lot of friction, I’m thinking this might not be a good fit.” What’s more, everyone has to agree on who to hire. “It has to be a unanimous vote. I’ve actually had to walk away from candidates because I had one person say ‘I don’t think this person is going to fit well on my team’, and so we said ‘I’m sorry it’s not going to work’, because the team is the one that’s going to have to work with them. I’m just going to have to manage them, that’s the easy part. The team has to work with them on a day to day basis. If that team is not sold on the fact that this person is the person that they want on their team, I’d rather just walk away and find another candidate.”

## Honesty and Salary Negotiations

“The final thing,” Evans says that he looks for, “once I have determined competency, once I know it’s a good cultural fit, is ‘can I afford them?”’ However, unlike in a lot of companies when it comes to salary discussions, Evans is clear about one thing — “I do not negotiate salaries. I ask the person ‘what do you need?’ I expect that person to be honest with me and tell me the number they need.”

Then regardless of what that number is, “if it fits within my budget and it’s a reasonable price… then I say okay. I don’t come back with a counter offer. I don’t negotiate this at all because my feeling is… if they’re being honest with me then they’re going to tell me what they need to be happy with this. I don’t mind side projects, but I don’t want people to have to do side projects to make ends meet. Quite honestly, if they’re not being honest with me on their salary needs, I’m probably not going to want them on my team to begin with because they’re not being honest with me, period.”

> ***“If you've already got developers, they're talking to their buddies and we know how much you're paying***

> *I don't mind side projects, but I don't want people to have to do side projects to make ends meet*

## Competing for Top Talent

"For most companies these days, their in-house development team is actually what drives the company," says Evans. They're the ones that "build the products that the company sells, or makes it possible for the companies to sell. So, while I'm never an advocate for setting developers up on a pedestal and saying we need to treat them like Demigods, I do think you have to respect the fact that you have to treat them differently. You cannot just say, 'we have a cookie cutter, this is how we treat people', Developers have different needs," Evans says. "If I start a job and they hand me a laptop that has been used by three other people and I have to figure out how to make things work, then that's a smell test that doesn't pass. Companies have to understand that you've got to invest in good tools for developers."

But how can you compete for top talent when Google and Facebook are offering top salaries with great perks? Evans suggests that you instead compete on respect. "I understand the mentality of free lunches and dry cleaning and all of this. Those exist so the company can keep the developer's butt in their seat longer, and they exist for no other reason. It's just a way to keep that developer on campus, focused on the job, for as long as possible." Whereas, "if a small company builds a culture of respect, where they respect the developer, they understand that their job is a portion of their life and that their job is not their whole life, then you can attract better developers than companies that have the free lunches, dry cleaning, etc."

## Retaining the Best Developers

The best way that you can show developers respect is to respect their time, says Evans. "I practice what's called Servant Leadership. It was not unusual for me, when I was running teams, to first thing in the morning, brew up a huge pot of coffee and walk around to each of my developers around 9am and refresh their coffee… this was just my way of saying my job is to hire good people and get out of the way and to make sure they have everything they need. My job is to serve them and also be what I call the 'poop shield'. Any poop that comes in from above, my job is to keep it off my team. If a manager needs a meeting with somebody on my team, I'll go. I will sit in that meeting," explains Evans.

Ultimately, Evans says, you just need to "treat developers with respect, and understand that they have lives, and the main point of their lives is not to build you the next 'Uber for Penguins'. You don't have to put them on a pedestal, but they aren't galley slaves either. You hire good people, you treat them with respect, you give them the tools that they need and let them do their job. That is the secret to a happy, productive, and awesome development team."

---

Read more from Cal at: [blog.calevans.com](blog.calevans.com)

# We're Bad at Interviewing Developers (and how to fix it)

Tips on running interviews and evaluating technical candidates

> I like to think of my software teams as little ecosystems, or tiny arcologies that exist in a bottle.

They're not entirely a closed environment, and like any ecosystem anytime you introduce anything new to that realm there's going to be changes… Any time you hire somebody, you're changing that ecosystem," says Miller. "A lot of teams and companies don't do a really great job of understanding that".

As a result, one mistake all too many of us make according to Miller, is just "hiring from our friend networks. The friend network is such an important part of how we get jobs, but it also tends to reinforce our monocultures. We tend to be friends with people who are mostly like us, and so those are the people that we're going to be recommending, and so those are the ones that get hired." But, "it's important to have some diversity… and not just the diversity we talk about in terms of gender or ethnicity or race, but age, class, looking at people's technical backgrounds, do they come out of CS programs versus being self-taught or a boot camp? Industry backgrounds… Were they at startups versus large enterprise companies, or somewhere in between? All those pieces of diversity are going to be influential and improve the health of the ecosystem of your team."

It's important to consider each team in this wider context. As Miller explains, "in soccer, they say, 'run to where the ball will be, rather than where the ball is'" and we all need to hire like that as well. You can start by having "early conversations about who you need to hire, and what you want to look for, what sort of energy and person do you want to add to your team, to influence it into a good direction and then go to those people, find them, whether it be through meetups, or user-groups… and not just your immediate friend network."

## Structuring the Interview

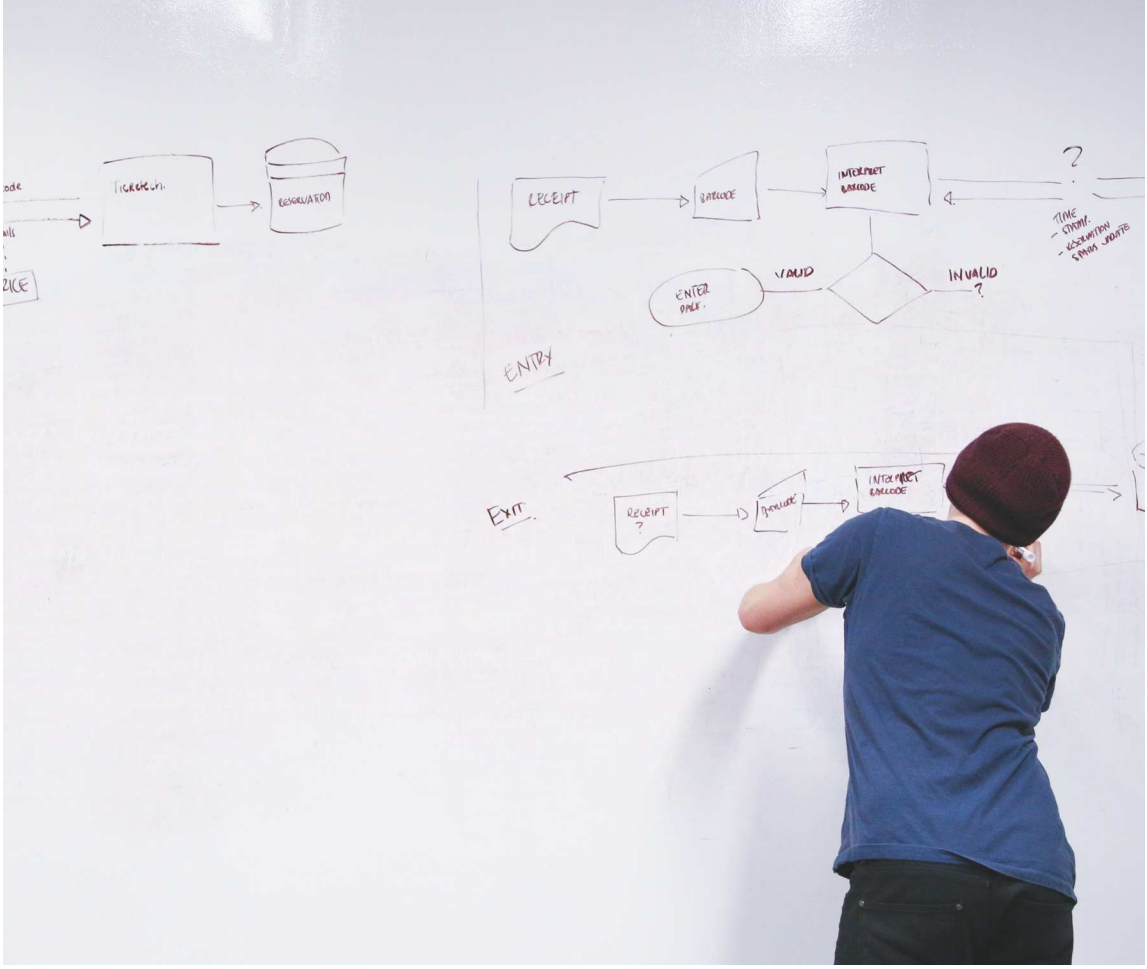
When it comes to interviewing technical candidates, Miller suggests that you start by "splitting up the interview into topics" and then working out "the questions you're going to ask" about that topic. However, it's important that "you're going to consistently ask these to all of your candidates". It may feel "a little bit like reading a script, but it really lets you compare apples to apples as much as possible," says Miller.

Another thing she recommends is that, if there are multiple people involved in interviewing a candidate and lets say "you're

Kerri Miller

We typically don't get trained how to interview, and we've all experienced the haphazard approaches of those new to it—poor organization, repeated questions, fizz-buzz... In this interview, Kerri Miller, Application Engineer at GitHub and former Lead Software Engineer at LivingSocial, tells us how to run interview days. She covers the types of questions to ask, how else we can evaluate technical candidates and what to do after the interview.

> ***Get at the heart of, not necessarily what somebody knows, but what they're capable of***




> ***Communication is such a big part of what we do in this job, testing for that essential skill can be really useful***

hiring for a front-end developer, you should have one person ask about JavaScript. You should have one person ask about browser interaction, or working with designers. Just split up the interview so that you're not asking the same questions over and over again". This helps because it means you're able "to get really solid signal on a person's skill sets, what they're comfortable with, and what their concerns are," explains Miller.

## Good Interview Questions

So what are good types of questions that we should be asking? Miller suggests that we "focus on questions about decisions that they've made, what choices have they made, and what choices would they make again in the future?" You want to understand whether "they're reflective about mistakes that they've made. Is a candidate looking for opportunities to improve, and how do they actually go about it?"

Then you also want to get a feel for whether "they make plans for themselves, like how they would improve a certain skill set, whether that be a technical skill set or a more soft-skill set, for example, management, or project shepherding for example. Those are the kinds of questions that I think really get you at the heart of, not necessarily what somebody knows, but what they're capable of."

"I'm not a big fan of whiteboarding," says Miller. "I think that's something that we just automatically do, and we don't think about what questions we are trying to answer by asking a candidate to solve a problem". Instead, she recommends "pairing on projects, like actually working with somebody. It doesn't have to be a formal or traditional pair-programming situation with

one computer and two people, talking through the technical choices that they would be making as they programmed on something".

> *"Immediately go back to your desk and not get back to work, but write down what your impressions were*

## Beyond Whiteboarding

Miller explains that "at LivingSocial, we do a code challenge… but as a launching pad to having a discussion with a candidate." Then as they're undertaking that exercise you can ask questions, like "why did you choose this technique? How would you do it better? What if we sat down and refactored? That's one really good way to get to the heart of why they are making the decisions they've made."

Another approach Miller suggests is "asking the employee to explain something to me". This is great for understanding "how well they communicate about something that they're a local expert in but their intended audience is not. Could they then go off and learn a new framework, or go have a meeting with, perhaps, a stakeholder, or a client, and come back and explain what the actual requirements are to me?" After all, Miller says that "communication is such a big part of what we do in this job, testing for that essential skill in a really clear and explicit way can be really useful and get you a really good signal about who that candidate is and how they're going to fit into your organization."

## After the Interview

"Once you're done with your little section of the interview, you should immediately go back to your desk and not get back to work, but write down what your impressions were. What were the pros and cons, the bullet points, and find something good about the candidate and something not-so-good about the candidate, something that you wish they did have. Don't pass this feedback back to the group but pass it back to a central person, like the hiring manager, so you're not coloring the impressions of other people."

For Miller, this immediate but individual feedback is important to capture so that "when you get back into that room with everybody else, whether it's virtual or real, to really discuss your opinions… you can't be swayed by the impressions of somebody else. For example, if you were supposed to interview them about JavaScript, and the senior developer, who's got twenty years of experience in JavaScript, just really did not like that person, how would that color your opinion if you had to give it in that moment? If you wrote it down previously then you've captured that honestly and you can really give honest feedback."

## Measuring Your Hiring Process

When it comes to measuring and improving your hiring process, Miller says that "it's really hard to look at who you hire and decide whether you have a good or bad process". Instead, she recommends taking a "look at who you don't hire. You can look at that in terms of what were the false negatives? Did we bounce this person out of the process for a specific reason and then it turns out that that reason wasn't good, based on where they ended up going to work?" For developers in particular, "it's really easy to LinkedIn stalk people, and peek into their GitHub profiles… to see what they're doing a few months later. So go back and look at the candidates that you passed over."

Another aspect that Miller thinks you should consider is "understanding what your pipeline of candidates consists of. At each step, you have a certain amount of leakage, because people just simply don't make it through the process or they abandon the process. How many people are you losing at each step, and is there one step that you're losing a lot of people at? Maybe you need to refine that step, remove it, or move it in the process. I think it's also important to look at who you're losing as well. Are you losing junior developers at a particular step? Are you losing more diverse candidates? And then understand, or question at least, your process."

---

Read more from Kerri at: [www.kerrizor.com](http://www.kerrizor.com)

# Building a Culture of Learning in Development Teams

In a lot of companies, people just learn on the job. But Mastey says that "when people try to learn on the job, they really only learn a narrow subset of all the things that they need to know about a technology. What that leaves is a big gap in their skills where maybe they know a lot about one particular section of the platform, but then they miss entire other sections. That ends up being a hindrance." What's more, people can pick up bad habits and practices from learning this way too. After all, not all of the code in our codebases is good, but people see that code and "pick up these terrible patterns and so they internalize the wrong thing instead of learning the right thing," says Mastey.

## Benefits of a Learning Culture

The answer to this problem is to establish a learning culture. "The thing that interested me the most was that by creating a culture of learning, we actually increased retention, not only of the younger people but also of the more experienced engineers," says Mastey. "There's this thing where when you're at a technology company for maybe a couple of years, right at the beginning you're learning new things. It's very exciting, you get a new platform… but, a couple of years in, and you are basically done learning parts of it. You feel like you slow down in that learning. This is the part where a lot of people end up dropping off. By building a culture that continually moves forward and rewards that kind of learning, we can attract really great engineers. And we can also retain really great engineers too."

Another benefit of a learning culture is that "it actually reduces waste," says Mastey. "One of the things that we've seen a lot of is if you aren't really great with your tools, you're not leveraging them completely. Because of that, you're spending more time than you need to. One of the things that's maybe counter-intuitive is that by spending this extra time, we actually end up saving time in the long run."

And anyone who is interested in establishing a learning program can do so. "I have no teaching background whatsoever," confesses Mastey, but "one of the things that I always tell

> ***"People pick up these terrible patterns and so they internalize the wrong thing instead of learning the right thing***

Joe Mastey

Joe Mastey is a developer who previously led internal learning at Enova where he built out their internal learning program. In this interview, we hear what worked, what didn't and how you too can kick-off an internal learning program for your development team.

people is that when you're trying to build this part of your culture, you don't need to have a teaching degree. You don't need that expertise. Really, if you're interested, you can figure it out."

## Key Phases

So there are three key stages that Mastey outlines to create a learning culture. "The first phase is really building up credibility. In that phase, we are trying to just find these quick wins and things to help people prove to themselves that what we we're doing is going to be valuable and going to provide value to the business".

You can progress to the second stage once you've built credibility and people are starting to understand the positive value a focus on learning can have. Mastey says that "in the second phase we can start to expand outside of the couple of people and some of those quick wins. We can start to invest time and things that take a little bit longer and now we're involving entire teams of people. We're trying to build links between teams. We can actually invest a little bit more money because people have started to understand what we're going to get back for that money."

Then by the third stage, it's "essentially making that a permanent part of the culture… you can make it a part of people's job". By this stage people "just don't even remember that it was ever something that they didn't do. At that part, you have the ability to make really big changes that are incredibly helpful to the company but really need to have everybody on board," explains Mastey.

## Activities to Try

To get started "you really want to focus opportunistically on places where you can have a lot of impact for only a little bit of effort," says Mastey. "Don't try upfront to change the world right out of the gate, because you don't have enough buy-in for that yet." You should also bear in mind that it's an iterative process, cautions Mastey. "A lot of things don't work. That's okay. Like that's part of the process… we don't know upfront what's going to work… so try a thousand things and monitor how they're working, but don't be afraid to throw them out."

"Some of the really easy stuff" to start with, suggests Mastey, "is have a brown bag lunch. Have a book club. Do code review. Literally, you can have two people on different teams, just have those two people do code review between each other. Already, you're starting to create the links within the organization that is going to support those changes later on".

Another suggestion is to "post in your company's chat program that you want to go to a meet-up and see if some other people will come with you," says Mastey. You want to think of "things where you only need a couple of people to buy-in. It doesn't really cost you anything, but you can come back and say, 'Okay. I now know about this new JavaScript framework. I now know about this tweak that I didn't understand before'".

Then once it's working, you can think of things that require a little more time investment. "Cross training is one of the big ones that I love," says Mastey. For example, "your UI people should understand the database. Your database people should understand how testing works" and through cross-training you can encourage this learning and reduce your bus factor.

Something that worked well for Mastey were weekly tech talks. "In the first company that I did this at, we had an hour long tech talk program, where it was four slots every single week. Every week, we got four new ideas into the organization". Of course, "that does take time, and it does take preparation, but when you can start to do that, the impact on the organization is really obvious and it's really good".

## Next Steps? Just go do it!

There is a lot that you can read on the subject. Mastey recommends Dave Hoover's Apprenticeship Patterns book in particular, but warns to "not get stuck on reading and learning everything beforehand, because everyone else's organization is really different than yours. Whatever you learn you're going to have to take with a grain of salt anyway. The best thing to do is really just get in and start getting feedback from your own organization and from your own stakeholders. Don't think that you're going to perfect it by reading."

---

Read more from Joe at:
www.josephmastey.com

# What Makes Developers Happy?

We asked some passionate developers when they are at their happiest whilst coding

"I get really excited about optimizing performance and experiences. I get frustrated very easily, so anytime I can use programming to take away my frustration is a golden opportunity."

Richard Schneeman, Software Engineer at Heroku

*"I'm at my happiest when I get to make the code itself more beautiful. I love producing APIs and code that other programmers will be happy to live in. I could refactor all day. I find cleaning up messy code deeply soothing."*

Bob Nystrom, Software Engineer at Google

"I love the “eureka” moment when I've gotten something to work, particularly when the solution was not obvious initially. Removing code when refactoring is always satisfying. I also love reading code that is beautiful and concise."

Lindi Emoungu, Senior Software Engineer at Google

*"I like making code beautiful. Performance matters and code has to do what it's supposed to do, but for me, the fun part is finding the most elegant way to solve a problem."*

Dusty Phillips, Software Engineer at Facebook

*"When my language and tools don't get in my way, and help me instead. Also, when my dependencies have good documentation."*

Steve Klabnik, Rust Community Team Lead at Mozilla

"I'm happiest when I know what I'm doing and I enter this zen mode. I love when I'm building a feature that I already know how to build, and I get to focus on getting things just right, and paying attention to the details of the code rather than just getting the feature to work."

Saron Yitbarek, Founder at CodeNewbie

*"I'm happiest when I've solved a problem to my satisfaction. Sometimes the people asking me to solve the problem are satisfied too."*

Chris Hartjes, QA Engineer at Mozilla

"The kinds of problems that I really enjoy coding-up are those where first, I have a clear specification of the problem, and second, I can solve the problem by creating subsystems that solve simpler problems and then fortifying the heck out of those subsystems."

Eric Lippert, Software Engineer at Facebook and author of ‘Essential C#’

# Technical Leadership

Part 3

# From Developer to Tech Lead

## Making the Leap into Technical Management

Pat Kua has been working as a Technical Leader for eight years, working alongside Agile development teams. "One of the interesting things I've observed is a gap in leadership skills for technical people", Kua says. Part of the cause of this is a lack of learning resources on the subject. "There are lots of books that teach you how to be a great developer, that teach you how to learn refactoring skills, things about good clean code, how to unit test stuff", but there's not much out there about becoming a Tech Lead.

That's not the only problem new Tech Leads have to contend with. It's also a lonely role — "you're kind of by yourself", Kua says. "There's not a lot of support. You're kind of thrown into this world between business people and this frustration of wanting to write code and there's nobody there to help you". You're taken away from the familiar, "where you're surrounded by compilers, test frameworks, new technologies and then suddenly you're having to worry about who on your team is currently a bit upset, about trying to negotiate an agreement between two very opinionated developers" and all "while you're still trying to get something delivered".

Anyone who has been in any sort of leadership position will know that "you won't have enough time to deal with all of the things that you'd like to do, and this is true when you're being pulled in quite a few different directions". And so of paramount importance is the ability to "really prioritize time to its best use possible", Kua says. As you develop in the role, you also have to broaden your perspective and take a more holistic view. "So thinking about longer-term plans, are there any bigger, nagging architecture issues that need to be dealt with", have we made "the right technology choices or should we be looking at a new technology and if so how do you kind of

introduce that".

A common theme in Kua's writing on the subject of Technical Leadership is that of the paradoxes the role throws up. As Tech Leads "we have to be really focused on delivering something. At the same time, we kind of want the team to be learning new technologies, to… spend time on training, to go to conferences" and "focus on removing problems and the technical challenges that we have in our environment. But, it's… hard to balance out those two kinds of aspects", Kua says. "I don't think you can ever really say that's it's a perfectly unbalanced equation, it will wax and wane depending on what things are going on. But it's a key skill for a Tech Lead to sort of accept the paradox and embrace the paradox and to find a way to do both at the same time".

## Should Tech Leads Code?

"When I talk to developers going into the Tech Lead role, the biggest thing is 'do I still get to code?'," says Kua. "My answer is, to be effective, I think you do need to code". "Developers really respect technical ability. And it's very difficult for people who want to follow a leader", who "want to respect their leadership decisions if they don't have that respect for their ability". This probably stems from "the classic ivory tower architect, that makes decisions far removed from the context of the problems that we experience on a day-to-day basis". So that's why "Tech Leads, to be effective, still need to code. Or at least, be at the coal face, where they're reading code, being able to work with people on code" to help ensure that they really "understand what the key problems are. To understand the language that developers use, so that they can talk at the same sort of level, and also know when people are maybe making a bigger deal out of something".

"I think 30% is a good minimum", says Kua, about the amount of time you should actually still spend coding. Whilst at times this might not be possible due to more pressing concerns, "it's important that the Tech Lead thinks about spending some time in the code, so you're getting the respect" and maintain "a true awareness of what's actually going on."

## Dealing With Interruptions

"Tech Leads will always be interrupted", says Kua. He recommends a few tactics to help cope with this. "Some Tech Leads simply block out time in their calendar so that they can actually spend time without interruptions. I think that's easier in some environments than others. In a sort of more Agile environment, where the team is all co-located, there are lots of things going on, even blocking out your calendar doesn't really solve a lot of that problem because people will just interrupt you".

Pat Kua

Pat Kua knows what it is to be a great Tech Lead. Not only has he worked with hundreds of clients, helping to solve their engineering problems, as part of his role as Principal Consultant and Tech Lead for Thoughtworks. But he's also learned a lot from his conversations with other experienced Technical Leaders. Kua spoke to more than 35 whilst researching his book, 'Talking with Tech Leads'.

> ***"There will always be some tense situation at some point in a project… it's kind of natural, people are unpredictable***

So Kua suggests "having some sort of visible sign that says 'actually I need some quiet time'". "What I've used personally, for instance, is a flag on the desk to indicate when it's no interruption time". "Other Tech Leads take themselves out of the working space. So they go off to a quiet room, they book a meeting room and then they sit in the room by themselves. Just so that they can go through a few things in quiet time. Another way is to maybe offset your day as well. So either, start early or end late, but do the same sort of hours. And this kind of helps you to get through all of those things before everyone else gets there into the workplace and interruptions start to come".

## Common Mistakes

"I've worked with a lot of developers, trying to coach them into this new role", says Kua. "When you watch new developers go into this role for the first time, I think there are two streams". "There's the Tech Lead that believes that the entire team should be self-empowered, every developer should know what they should do, and the Tech Lead wants to have trust in everyone". "So they sort of just sit back and hope for the best. And I think this is where things get a bit dangerous if the team hasn't gotten into that mode and they don't already have that strong vision". Typically then, in the absence of true leadership, "the most opinionated developers sort of pipe up and they form the direction themselves" and this can "quickly fall apart", says Kua. When this happens "people spend a lot of time arguing and the same things come up over and over again" and are never truly resolved.

However, Kua has also seen those that think they "need to demonstrate that I'm a Leader, they'll want to still make all of the heavy technical choices, so they'll want to be involved in every technical discussion". And while "I think these two extremes are elements that good Tech Leads should be able to play", "they shouldn't be playing that mode of Tech Lead all of the time". Kua recommends an understanding of the situation leadership model, "which kind of talks about situations where more directive behavior is useful. Where maybe more targeted coaching is useful, and where perhaps pure delegation, and just sitting back and trusting the team to do the things that need to be done makes sense". But it can take "time to develop an awareness of when to use which mode".

> ***"Tech Leads, to be effective, still need to code or at least be at the coal face***

## Skills to Develop

Kua says that the people side of the role, is "one of the hardest things for a Tech Lead to learn as it's a skill that you don't necessarily practice when being a developer". Another key part of this is the need for the tech Lead to act as a linchpin between engineering and the business. "A lot of the skills from Tech Leads, that end up being developed is this kind of translation, where you're kind of trying to avoid all of the technical terms and trying to express it in ways that your normal family members might be able to understand who don't understand technology whatsoever. But you kind of need their buy-in to understand why you're going a certain direction with things".

To help build those skills Kua recommends the book 'Getting To Yes'. "The book is about negotiating, and it's about trying to come up with a good way forward while you're trying to appease both sets of interests. I think it has some really great stories that help people to understand some good skills".

He also recommends the books 'Crucial Confrontations' and 'Crucial Conversations.' "They're useful because there will always be some tense situation at some point in a project" — "it's kind of natural, people are unpredictable". "I think that the books help people to understand how to have a discussion around emotionally sensitive or difficult topics."

---

Read more from Pat at:
www.thekua.com/atwork/

# Become the Leader Your Engineers Need

Practical tips for Programmers who want to lead

> *If you want them to follow your authority as a manager, you have to make sure you know how to build the trust of others*


Oren Ellenbogen

Oren Ellenbogen is the author of 'Leading Snowflakes', and in this interview, he discusses some important tips for those moving into Developer team leadership.


"I would definitely advise you to earn your teammates' trust first," begins Ellenbogen. "If you want them to follow your authority as a manager, you have to make sure you know how to build the trust of others, starting with your teammates. Figure out a way to earn their trust - maybe it's your technical skills, maybe it's your business understanding, maybe it's a mix of everything. But don't take it for granted and think that just because you have a title, that everyone should do what you say."

With your credibility established, Ellenbogen suggests you then "clear the path to get more feedback". He explains that "as engineers we have the luxury of a really fast feedback cycle. We have code review, we can deploy code and see tests running, and run it in production." But we don't have such established mechanisms in management decision making. However, Ellenbogen says that this should not stop you — "great engineering management develop good ways to bring in feedback, rather than just sitting in the dark and feeling alone."

"A trick that has really worked well for me", says Ellenbogen, is code reviewing management decisions. Ellenbogen took "the concept of code review and applied it to management." By this he explains that he would "capture one or two of the dilemmas that I had during the day" and then he would ask some trusted colleagues "what they would do in this situation". What he found was that by "just having a discussion around the dilemmas you have, you can really open up the way you are thinking," explains Ellenbogen.

Another key aspect for any engineering leader is "to know the business inside out," says Ellenbogen. "I know there are many engineers and engineering managers that feel that business is not our job — it's the sales guys, it's the marketing guys… but you can spend many, many days at the office writing software just to find out that nobody uses it." So if you want

to do what's in the best interests of your team, then figure out the business so you can "understand the risks, understand the current capabilities that will help your business scale and how you can convey the message to your teammates," suggests Ellenbogen.

## Managers Should Code

A concern for many considering a move into management is whether they still get to code. But Ellenbogen suggests that at least at first, "I would advise you focus on feeling comfortable with your role as an engineering manager." Coding is a safe activity for any developer, so "if you are finding yourself always shutting down and getting back to writing code all the time, that is kind of a red flag. That means you should probably just avoid writing code and focus more on the people aspect, or on the business aspect, and stop writing code," says Ellenbogen. "You have to feel like your teammates are being productive and you are not being called on to save the day. Once you are feeling more comfortable with that, then you can go back to writing code."

But it is important that you do still write code in the long-term, says Ellenbogen. At the very least you want "to keep your technical skills at a high level" so that you can "make sure you are a part of the conversation when your teammates need you". When you do so, "focus on fixing bugs, dealing with nasty performance issues, reducing technical debt… or work on tools that will increase the productivity of your teammates." But regardless of what you do, "you should avoid being on the critical path." Interruptions are common and you don't want to be your team's bottleneck.

## How to Stretch Your Team

To get the best out of your team, sometimes it's necessary to push members out of their comfort zones. But what's the best way to go about this? "Start by setting an example that people can emotionally connect with," advises Ellenbogen. "I shared my struggles with the team… I was just honest with my teammates… but later on when I talked with them about pushing them out of their comfort zone, just being able to relate to the way that I have been honest about my struggles just made the conversation a lot easier."

Beyond that, Ellenbogen suggests that you provide "examples from inside the organization so they can see how it could be. For example, if I want to push someone out of their comfort zone regarding the way they are communicating their progress, I would forward emails from other teammates or other teams in the organization who are doing a brilliant job at it," explains Ellenbogen. They can then pick a few things up from these and we can then "talk about it next time in our one-on-one and have a discussion around that."

## Scaling an Engineering Team

"Often engineering managers try to own everything," says Ellenbogen. "They try to own the people aspect, try to own technical aspects, strategy, business". There's just one problem with this approach — it doesn't scale. So what you must do, according to Ellenbogen, is "leverage the fact that you have great people working with you". He recommends that you delegate ownership of key areas like code quality, to team members with a passion for them. Regardless, "I make sure that every time someone comes to me and says, 'hey, we have a problem that we need to fix', I write it down. I never ignore it," says Ellenbogen. Then "I make sure we pick a few things we want to work at every couple of weeks or every month… so the team feels like things are not being ignored."

## Working with Other Teams

Finally, a key part of the success of any team isn't just individual team brilliance, but your ability to work well as part of the wider organization. To this end, Ellenbogen suggests that a great first step is to "share your plan for the next few months or the next few weeks with your peers". "It might sound obvious," says Ellenbogen, "but from my experience, very few do it." He suggests that you "try to share as much as you can with others so they can understand what's the plan and where you are heading." Then follow this up by "asking for their opinion" and getting feedback from their perspective. Then "if things change, and that is alright, make sure that you keep everyone in the loop."

---

Read more from Oren at: [www.lnbogen.com](http://www.lnbogen.com)

# Growing Self-organizing Software Teams

## How to develop teams with different phases of leadership

> *If you get hit by a bus tomorrow, could the team continue to function without you?*

Often developers can be reluctant to put themselves forward for leadership roles, but Osherove says that whilst it can be scary, "it's not something that should block you". In fact, if you want things to change, then you should actively seek it out. "Remember all those things that you've always wanted your manager or leader to do? You can actually get to do them… It's a great chance to change the environment, and not just bitch about it," says Osherove

### Growing Team Members

In his book, Osherove explains that "a team leader grows the people on their team," and he thinks that this should be the key thing driving your behavior as a lead. "It's almost required," says Osherove. "If you don't do it, you're basically stopping in the same place and you're not actually changing anything". Furthermore, "if you want the team to do unit testing or test-driven development, but the team itself thinks it's not such a great idea, how do you actually get them to do it without growing them in the right direction?"

"Another thing that happens is that a lot of teams are actually not in a position where they can actually make good decisions," says Osherove. So "as a leader, you are the bottleneck. They come to you with every little decision", but "what we want is for a team to grow their skills so they can handle the current reality that they're in".

Osherove says you should ask yourself "are you the bus factor? In other words, if you get hit by a bus tomorrow, could the team continue to function without you? If the answer is no, then you don't really have a team. What you have is a bunch of people to help you".

To gauge whether you're moving in the right direction with this, each week ask yourself whether "the team need me more or less than last week? If they need me less, if I make myself less needed, if I remove myself from the equation, not just by disappearing but by enabling the team to do the things that I know how to do, then I've

Roy Osherove

Roy Osherove, author of 'Notes to a Software Team Leader', discusses how to grow self-organizing software teams. He covers how to develop team members, different phases of leadership, and some common mistakes made by new tech leads.

basically made them better".

## Phases of Leadership

In Osherove's view, there are three main phases of leadership. There's "survival mode, where you don't have time to learn, you don't have time except only to react; learning mode, where you're supposed to do things and fail and learn and do things slower; and self-organization, where the team can handle their situation without requiring the leader".

Osherove explains that "you know you're in survival mode if you don't have any time to do the things that you would like to do, and you keep reacting instead of planning. Even if you have time to send people on a two-day course, it doesn't mean that you have time to implement what they learn. If you don't have time to do slow practice, you're in survival mode."

Of course, life is rarely as simple as being in a specific binary state. But Osherove explains that "it could be that a part of the team is in survival mode and a part of the team is self-organizing. If you have a large enough team, you might start to get cliques where some people are doing okay, whilst some people are definitely buried deep". But "the way I see it is that it always hunkers down to the lowest common denominator. That means that if a part of the team is in survival mode, your team is in survival mode".

## Mistakes Made By New Tech Leads

"I think one of the most common mistakes," says Osherove, "is that they tell people what they want to hear instead of telling them what the reality is, because they're really scared and they want to make a good impression". But, "we get paid to do good work and to say if there's a problem," although it really "takes a while to learn that."

Another common mistake that Osherove highlights is not considering the mode their team is in, and the appropriate phase of leadership to apply. For example, "in learning mode, you definitely want to be a coach. You want to teach people. You want to give them time to do it. In survival mode, you usually want to be command-control. You don't want to start coaching. You don't have time to coach. You have to get people out of survival mode and into learning mode. If you mix those things up, you have a problem. Suddenly, you're not really helping the situation. Similarly, if you have a self-organizing team that already knows how to work, already knows how to do something, and you treat them as if they're in survival mode with micromanagement and command-and-control, you're going to lose your team very, very quickly." The particularly difficult part though, says Osherove, is that "the mode can change from day to day. You have a new project, and suddenly you are back in survival mode."

---

Read more from Roy at: [www.5whys.com](http://www.5whys.com)

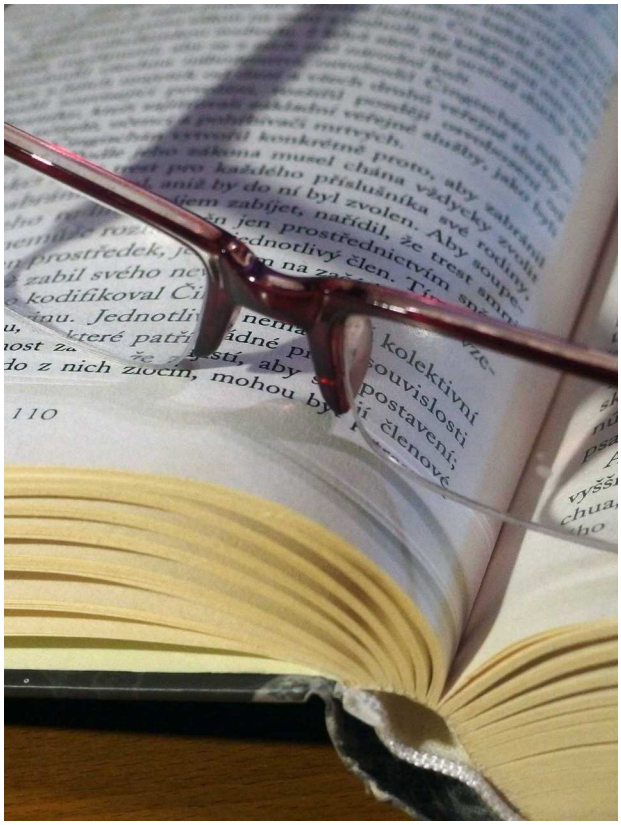
# *Creeker Wisdom*

"Clout disrupts meaningful discussion. If you're the one with clout (say, you're some sort of manager or lead) be mindful that until you build a strong rapport with someone, going against your opinion is scary and accepting your criticism constructively can be hard. There will be some people with whom you'll never gain sufficient rapport. There will be people who bottle up their opinion instead of confronting you. There are many ways to help this, but I have no single recipe. Mindfulness is key, however, because if you're ignorant of it then your clout can drown out the very voices you depend on. If you're talking to someone with clout, be fearless, ask questions and have a meaningful presence in the discussion."

- Jude Allred, CTO at Fog Creek

# Recommended Reading

We asked all contributors for their favorite books about programming and technical leadership. Here's what they recommended.



‘The Pragmatic Programmer’, Hunt and Thomas

‘The Mythical Man-Month’, Brooks

‘The Art Of Computer Programming’, Knuth

‘Effective Java‘, Bloch

‘Code Complete’, McConnell

‘Design Patterns’, Gamma, Helm, Johnson and Vlissides

‘Purely Functional Data Structures’, Okasaki

‘Extreme Programming Explained’, Beck

‘C# In Depth‘, Skeet

‘Thinking In Java‘, Eckel

‘The C# Programming Language’, Hejlsberg, Torgersen, Wiltamuth and Golde

‘Apprenticeship Patterns’, Hoover

‘Metaprogramming Ruby’, Perrotta

‘Becoming a Technical Leader’, Weinberg

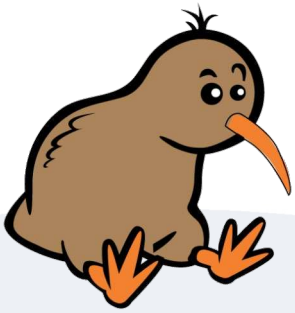
‘Management 3.0’, Appelo

‘Ada’s Algorithm‘, Essinger

‘Peopleware’, DeMarco and Lister

‘Agile Management From Software Engineering’, Anderson

‘Getting to Yes’, Fisher and Ury



# FogBugz

Trusted By

VOX MEDIA · TESCO · nielsen · Adobe · Zoopla · Deloitte. · unity · CHEEZburger · swiftype · 53 · bandcamp · KHANACADEMY

# Work Tracking and Collaboration

Used to manage over 20,000 software-led businesses and teams

## Four products that work together

- **Tasker**: Task Management
- **Issue Desk**: Helpdesk and Issue Tracking
- **Agile**: Agile Project Management
- **Dev Hub**: Software Development Management

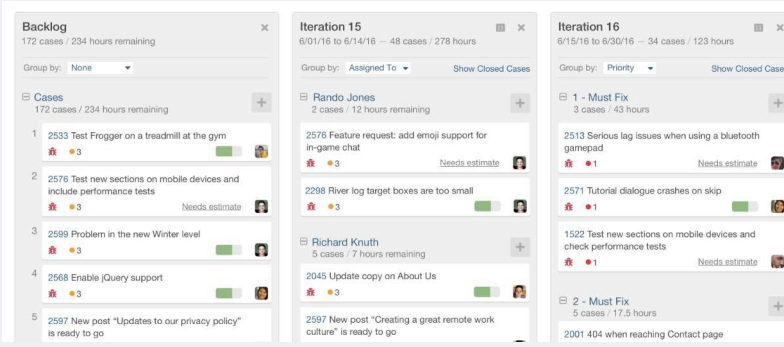


## On Demand or On Site

With On Demand, FogBugz is hosted by us in the Cloud for ease of access and flexibility. Or with On Site, you can host it on-premises for enhanced control. Either way, you get the same powerful features and world-class support.

## Focus on *real* work, not managing it

There's no time-consuming setup or confusing customization. Workflows are simple yet flexible. FogBugz provides everything you need to plan, build and ship software.




Learn more at [fogcreek.com/fogbugz](fogcreek.com/fogbugz)


Fog Creek
SOFTWARE