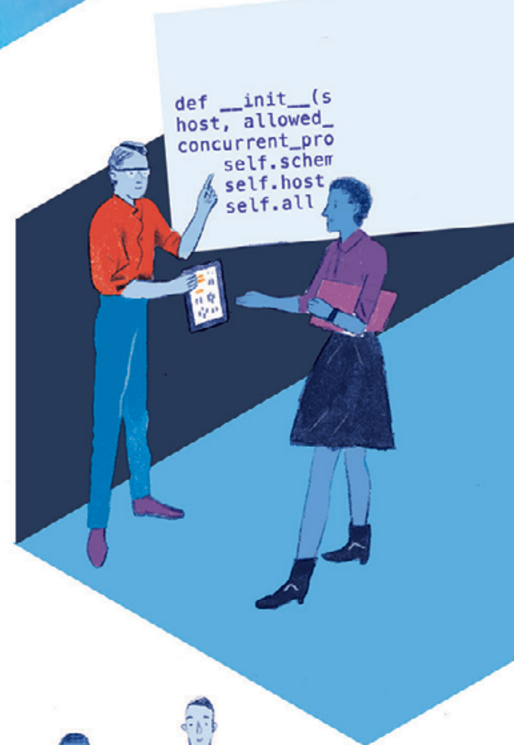
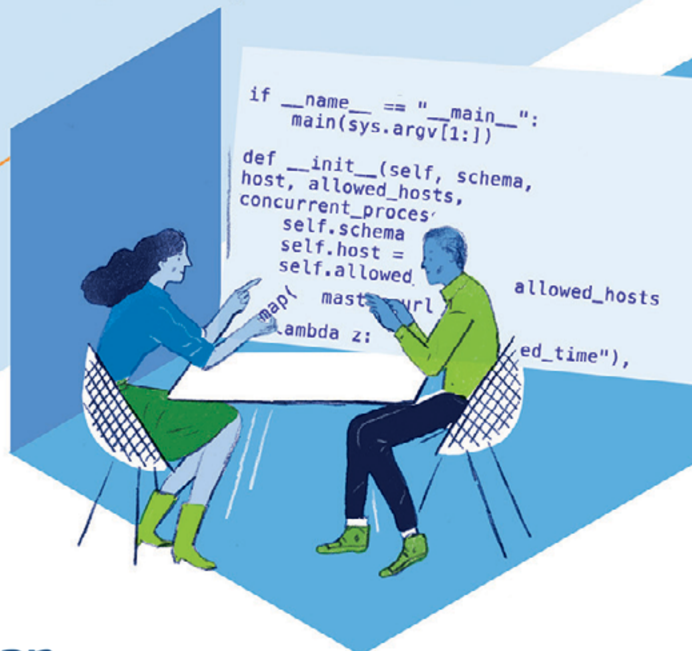
Hello World!

A new grad's guide to coding as a team





Atlassian

CONTENTS

Foreword 3
Scott Farquhar, Co-Founder/Co-CEO

Full-time school to full time life 5
Jamie Georgeson, Developer

Why I was wrong about code reviews 11
Steve Haffenden, Lead Developer

Try pair programming 14
Lucy Bain, Front-end Developer

Letter from an @ignored test 20
Mauri Edo, QA Engineer

Functional programmers vs. functional engineers 23
Sidney Shek, Developer

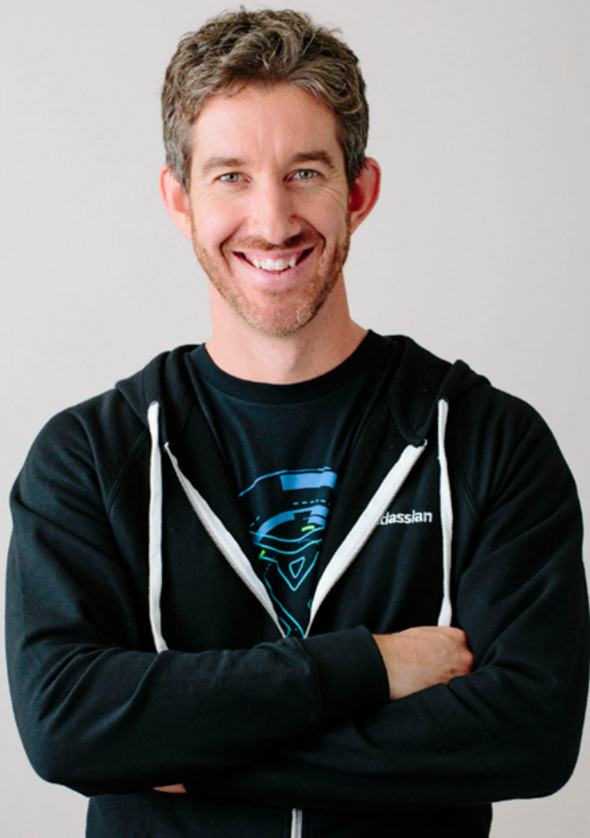
When I grow up, I want to be... a team lead? 28
Agnes Ro, Senior Team Lead

On being a woman in tech 33
Denise Unterwurzacher, Developer

Maintaining a growth mindset 37
Steve Haffenden, Lead Developer

Just bloody do it 43
Gilmore Davidson, Developer

Resources & further reading 45



Congratulations!

The hard work you've put in and perseverance you've shown over the past few years has paid off. Now the fun begins (and I don't just mean a summer of post-exam partying, though that's a lot of fun, too).

Amongst the myriad transitions you're making is one that doesn't often get mentioned in commencement speeches: you're about to go from being a student to being a teammate. Even if you worked on dozens of group projects in school, you'll find that working on a professional software team is different.

Really different. And in a good way.

Chiefly, you'll be able to accomplish far more with your team than you ever could on your own.

We often celebrate the "lone genius" because they're not only accomplished, but highly visible. It's easy to forget that they don't work alone. Mark Zuckerberg and Elon Musk have massive teams of engineers, designers, and advisors helping bring their visions to life. Derek Jeter, Steph Curry, and Misty Copeland are surrounded by teams that include coaches, trainers, and doctors, in addition to their fellow athletes. Even musical acts like Taylor Swift and Lorde have teams behind them: sound engineers, producers, and other musicians who support them in the studio and on the stage.

Talk to anyone in any industry and they'll tell you that the best work of their lives was (or is) as part of a team.

My first professional team was a team of two. Just me and my best mate, Mike Cannon-Brookes. When we graduated in 2001, most programming jobs in Sydney were at banks, which was a bit too buttoned-down for our tastes. We figured if we could code for a living without having to wear a tie, that'd be pretty sweet. So we teamed up and started a company.

Now our team has grown to over 1500 Atlassians. Developers, designers, customer support engineers, accountants, recruiters... you name it. They are talented, dedicated, delightfully weird, and together we're building something I'm truly proud of. Something bigger than the products we make. Together, we're proving that trust, transparency, empathy, and collaboration are the currency of the modern workplace. There's no way I could do this on my own. Not in a hundred lifetimes.

This book comes from our team of developers. They think a lot about what it means to be a great developer and a great teammate, and they wanted to share their thoughts. As you'll see, they believe life-long learning, humility, courage (including the courage to humbly accept critiques), and openness are every bit as important as being a killer coder. I hope these essays give you a head start in building a set of life skills to complement the technical skills you learned in school.

The energy and fresh perspective computer science graduates bring each year is an indispensable asset to the tech industry. We've got a million dreams to turn into reality, and only one lifetime to do it in. Welcome!

Scott Farquhar
Co-Founder/Co-CEO, Atlassian
Class of 2001

P.s.: Atlassian is hiring! A lot. Check out **atlassian.com/careers** to learn more.

From full-time school to full-time life

Jamie Georgeson
Developer
JIRA
Class of 2015

Hi, I'm Jamie, one of 80 new graduates to recently join Atlassian as a developer at our headquarters in Sydney. I'm in the middle of transitioning from "full-time study, plus part-time work" to "just full-time work". Seems like that should be easy, right?

It's not.

It's hard in all kinds of ways I didn't expect. Habits and attitudes that worked well for me and my peers during university (we call it "uni" down here in Australia, btw) are now unproductive—possibly even toxic.

Unlearning the behaviours I worked so hard to develop over the last four years is both mechanically difficult and a little bit soul-crushing. Luckily, I have experienced peers who tolerate my confused questions, share their wisdom, and generally help me make the move from uni life to real life. They've been good to me.

> Maintaining a healthy work/life balance isn't lazy. It's actually harder than working 80 hours a week.

Now it's my turn to share what I've learned with other new and soon-to-be grads. Maybe this will give you a head start on the massive brain rearrangement you're about to undergo.

Toxic Habit #1: overwork and brute-force timing

I've heard friends, family, lecturers, and even bosses stress the importance of work/life balance. And I quietly rejected them as lazy.

If you want better marks in uni, the recipe is simple: sleep less, study longer, and work ahead. So everyone who's even slightly motivated keeps calm and quietly soldiers on. And this has always

worked really, really well—regardless of how tired your Nana says you look.

What's different now? I don't actually know (brute force will probably still work for a while). But I have a new perspective, regardless. *Maintaining a healthy work/life balance isn't lazy. It's actually harder than working 80 hours a week.*

Saying "no" in the name of protecting your off-hours time is harder than saying "yes" and taking on more work than you can handle. Eating right and getting enough sleep requires discipline, but boosts your long-term productivity. And it's about more than your own well-being: you're an important part of a team, and you all rely on each other.

Maybe you (ok: I) genuinely think burnout won't apply to you (i.e., me). But if we keep moving at the speed of uni, one day *it will*.

We're playing the long game now. So if you really care about getting $#!т done, go to bed on time and eat your peas. Call your Nana this week, too. Take it on as a personal challenge. It's not a lapse in work ethic.

Toxic habit #2: competition and one-size-fits-all evaluation

I'm not a competitive person in that I don't care about being the best. I do, however, desperately strive for adequate.

It's easy to evaluate yourself against your peers in uni. Everyone completes the same prescribed work at the same time under the same conditions, and everyone gets a nice, clean, quantifiable outcome at the end. If your marks are above the mean, you're doing OK, and you can wallow in something other than self-loathing for the next week (I recommend fear of the future).

What's different now? One-to-one comparisons are totally invalid.

How would you, even? We were all hired for different reasons, we all have different skills, and we're working on different problems. You can't get down on yourself because someone knows more Scala than you or because they're "more creative". They're equally in awe

of a different trait in someone else. The awe goes around like links in a chain and links back to you eventually, whether you know it or not.

In uni, we were expected to be on top of *everything*. But the products, services, and systems we work on with our post-uni teams are too big and too complicated for one person to be across it all. So relax for two seconds and adjust your perspective.

More importantly, grading yourself against your teammates is irrelevant. Who cares how capable you are relative to everyone else? What's important is to figure out what you can contribute to the team, do that hella hard, and never stop trying to be awesome. Growth mindset FTW.

This is by far the hardest adjustment for me to make, and I'm still struggling with it. How do you know that you're progressing fast enough as a developer if you can't compare yourself with your peers in similar positions? How do you know that your personal goals are ambitious enough? Where is the baseline? AND FOR THE LOVE OF GOD, WON'T SOMEBODY GRADE ME, PLEASE?!?

The advice I got was to maintain open communication with my manager and let them help guide my progress. It's definitely helping. I recommend it.

Beyond that, I'm replacing "compare myself to other new grads" with "aspire to someone way above me." My team has some pretty smart people—people who are *incomprehensibly capable* in my squishy little grad mind. Your team will be no different: it's full of people who are clear examples of "what to be". Work with them on a project or sit with them at lunch and figure out how they've achieved what they've achieved.

> Working full time is like learning to walk again. Except my legs are made out of pudding and the floor is lava and there's a shark in the lava...

New habit: take a different road for your extra mile

Maybe you scoff at a 40-hour work week as "casual", and are still desperate to commit more time to your craft. But how? It used to be easy: if I had a quiet week at uni, it became a busy week at work (and vice versa). I used to think putting my head down and doing more of whatever it is I'm already doing was the best use of my excess energy.

What's different now? I've discovered loads of auxiliary activities that make me more productive because they *make me more balanced*. Here are a few ideas:

Organise volunteer work for your team. A day at a soup kitchen? A longer-term engagement with the tutoring program at that nearby school? A hackathon in which you help a local non-profit jazz up their website? You will feel so good for it. And speaking of hackathons...

Make the most of 20% time and hackathons. Maybe your team allows people to work on passion projects 20% of the time. Maybe your company does 24-hour events like Atlassian's ShipIt days. Use these opportunities to fill in gaps in your knowledge, explore your crazy new ideas, and/or tackle a hard problem.

Help a team member. Or me. If you're in Sydney, come help me specifically. (I'm at 341 George Street, 8th floor, near the service

elevators.) But seriously: someone somewhere is struggling with something and could use a hand.

Write a blog ☺ Share something on your company's intranet or LinkedIn Pulse or just your Facebook page. Somebody somewhere will learn something from it. Promise.

And if you really insist on doing more work after you go home, at least work on your own projects. Learn a new language (whether spoken or coded), build that sculpture you've always dreamed of taking to Burning Man, write a short story. I hear about people who get into software development or design or architecture because they love it... only to hate it once it becomes a career. Let's not let this happen to us.

I have no idea what I'm doing

I may have tricked you in to thinking I've nailed down how to survive in the workplace long-term, but I'm still struggling to put all this into practice.

Working full time is like learning to walk again. Except my legs are made out of pudding and the floor is lava and there's a shark in the lava (I think his name is Bruce). But that's cool. I'll get into my comfort zone soon enough, and then I'll get bored with that zone and move outside it again. It's the circle of life for us career-minded kids.

The people around us are awesome at what they do (literally inspiring awe), and we're properly inadequate (in a good way, because aspirations). For now, it's enough to be thankful for the opportunities in front of us and the support we've had so far. Good luck out there, everybody. ◎

Ask your teammates questions rather than trying to solve it all by yourself.

"When you get stuck on a problem, demonstrate your brilliance by asking your teammates intelligent questions rather than trying to solve it all by yourself (you will take too long and will still be wrong)."

Malcolm Purvis, JIRA Developer | Class of 2015

Why I was wrong about code reviews

Steve Haffenden
Lead Developer
JIRA
Class of 1996

If you've ever had to go through the process of a code review then I'm sure you'd be familiar with the time-honored tradition of raising a pull request and having your hand-written code critiqued. This is something that I've traditionally disliked. It's slow, can be distracting, and if I'm being completely honest, I don't like hearing that I'm wrong. Who does?

However, as a new member of my team, I thought it best to do things the right way and I found myself relying heavily on the code review process. For me, what I had once considered a mere a rubber stamp on my work became a process that helped me identify inconsistencies in my code style and validated that I was still capable of writing Java.

The code review became such a positive experience for me as a JIRA Software engineer that I started thinking I'd been unfairly harsh on the whole code review process and that I needed to change my perspective.

Better than yesterday, not as good as tomorrow

There's a UK hiphop artist called Scroobius Pip and you'd be forgiven for not being familiar with him, but in the context of code reviews he's important for penning the following lyrics:

If your only goal's to be as good as Scroobius Pip
Then as soon as you achieve that your standards have slipped
If your goal is always to improve on yourself
Then the quest is never over no matter how big your wealth

You see, it's important that we all strive to be better at everything that we do. That doesn't mean comparing ourselves to others, but rather taking the time to look at ourselves to compare the way we do things now against the way we did things yesterday and the way

we want to do things tomorrow. Improving our own skills and techniques is a good practice and makes every day a challenge.

Check your code review technique

Code reviews are an integral part of development culture at Atlassian and many other companies, but it can be easy to treat them with a certain amount of disdain. Us devs often find ourselves on the receiving end of a code review feeling tempted to give it only a cursory glance before clicking the approve button, or maybe waiting until someone else completes it before submitting our own approval without checking the changes. And when a code review is created, we may not always provide the necessary context around our changes or set up an instance to allow our reviewers to validate our changes easily. Lastly, we often fall into the trap of taking feedback on our code as a criticism of our abilities rather than an opportunity to hone our skills.

The thing is, different people will identify with the role a code review plays, well, differently. Some may view it as a method of identifying errors in code, a form of testing, a rubber stamping process, or even human linting. To make the most out of each code review it's worth considering the benefits it offers to your own development, and ask yourself: How does this review to help me grow as a developer?

Keep coding, keep growing

The crux of the matter is that the code review process is a great opportunity to take stock of your work. You might find better, more effective ways of implementing solutions, or practice the valuable art of providing concise and easy-to-understand feedback. Code reviews could also help you improve your depth of knowledge in a specific product, library, or language; or even just increase your ability to read, understand, and reason about other team members' code.

There's always something new to learn in even the most mundane of code reviews. Next time you're starting a code review—whether it be as a reviewer or as someone creating a review—start by asking yourself how it can help you and your team grow. Hopefully it will lead to a more fruitful experience for you and everyone else involved and, as we all grow, the quality of what we produce will grow too. ◎

Be active on open-source software forums even if you aren’t an official contributor.

“Being active and helpful on open-source software forums will help you find jobs and grow professionally even if you aren’t an official contributor on the project.”

Andrew Swan, Senior Developer | Class of 1985

Try pair programming

Lucy Bain
Front-end Developer
Class of 2011

It's not easy to start pair programming. Most of us don't get much of a chance in school, and often our first programming job is on a team that hasn't done it much (or at all) in the past. Still, it's worth making the effort.

So here's a quick-start guide to help get your pairing efforts off the ground: the benefits, the vocabulary, and different formats you can try as you're figuring out which style works best for you and your teammates.

Why pair?

It's a great way to share knowledge. You'll learn, teach, and work with code you might have missed. Often you understand the code better at the end of a pairing session because each of you were asking "why?" more than when you work alone.

You'll get some good instruction. Pairing can be particularly useful for new hires. It's a great way to get to know your team, learn about coding styles and expectations, and find who's the right person to ask about a given topic.

It helps you stay focused. Checking your social media site of choice is much less appealing when there's a person right there to talk (and code!) with.

You'll write better code. You don't write code much faster with two people, but you write it with fewer bugs, which gets you to "done" faster in the long run. There isn't a lot of research to back this up—purely anecdotal.

Personally, I think the other reasons are more compelling.

If you're thinking "aren't these the same benefits of code review?", then yes! You caught me! Pair programming is indeed a lot like code review. However, pair programming has one major advantage over

code review: you review in real time. That means you get to make corrections to your code before you've added a bunch more code on top of it.

If that all sounds amazing, that's because it is. So let's get started!

Finding a pairing partner

The first thing you'll need is a pairing partner, who can be anyone who wants to pair with you. A willing, happy pair is better than the person who is "best matched" in terms of experience or domain expertise. And the best way to find partners is to just start asking. If your teammates don't take the bait when you bring it up casually at lunch, try sending a meeting invite. They might take your pairing invitation more seriously when there's a super-official-looking calendar entry for it ☺.

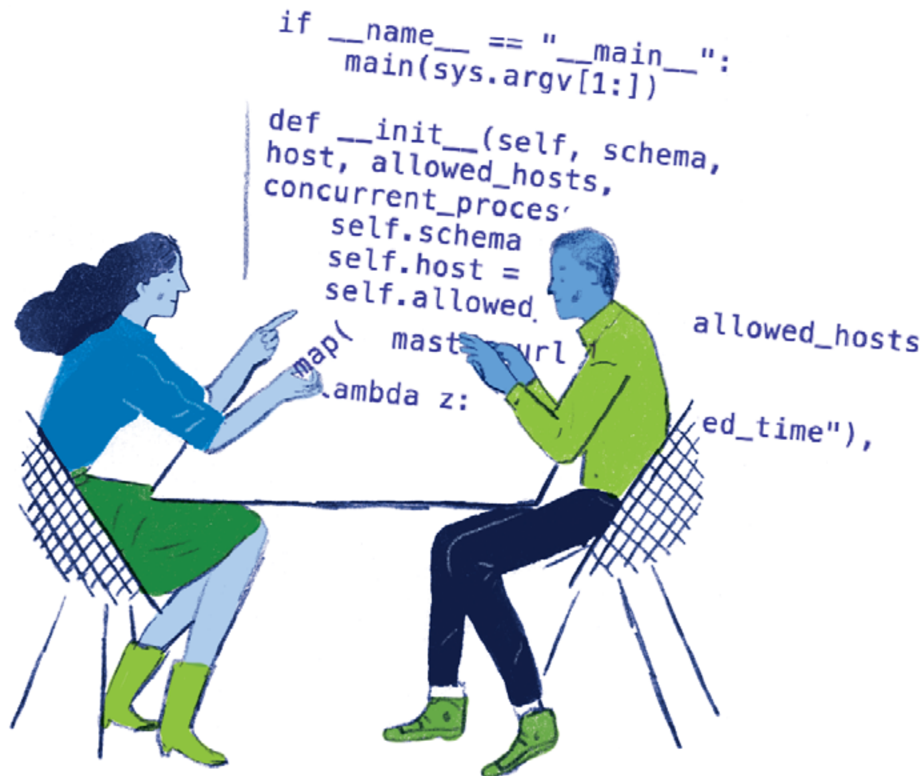
> A willing, happy pair is better than the person who is "best matched" in terms of experience or domain expertise.

Offering to pair up on the other person's task can be a good way to entice people who've never paired before. They may politely say they don't want to "waste your time" by working on their task, but push back on that. If the task is worth their time do in the first place, it's probably worth your time to pair on it. Besides, when you're new to a team, pairing is an excellent way to get up to speed on the code base. The time they invest in pairing with you will pay off quickly in terms of the value you add to the team.

That said, some people just don't love pairing. It's alright (and sometimes necessary) to push a little bit, but ultimately, be willing to gracefully accept a "no."

Once you've got a partner in (crime) code, how often you pair is up to you and your team. Just make sure you're all on the same page since pairing plans may influence how many tasks your team takes on in a given timeframe. If possible, begin with two sessions a week with different people on your team. That way you'll see what it's like to pair with different people.

Pairing sessions can range from 90 min one-offs to pairing all day, every day for a full sprint. One team found many short pairings was better, but your milage may vary. I recommend starting with 90-minute sessions and adjusting from there.



Finding a pairing style that works for you

Understanding the terminology and different styles of pairing will help you get off on the right foot. Let's start with the vocab:

Driver: Does the typing; bounces ideas off the navigator; gets the up-close view of the code.

Navigator: Looks for logic problems, bugs, and better implementations; acts as a sounding board, and thinks ahead to potential problems; gets the macro view of the code.

As for styles, I've tried or witnessed four of them. Try starting with "the noob" and fall back to "the distracted" when you need to look something up. Build up to "the classic." *Note: the names below are entirely my own invention.*

THE CLASSIC

Bring your keyboard and mouse to your partner's desk so you can find answers to questions as they come up ("What other libraries might be useful here? "When was the last time anybody worked in this area of the code?"). My team has recently set up a pairing station with two keyboards, mice, and monitors for "plug n' play" pairing—pretty handy!

Keep swapping the "driver" role with your pair—usually every 20-60 minutes. For beginners, I recommend swapping frequently.

Pros: You get to contribute as you go along.

Cons: Limited desk space for extra keyboard and mouse.

THE LAZY

Same as above, but don't bring your keyboard or mouse.

Pros: Won't be playing mouse-stealing games; setting up at the beginning of your session is really easy.

Cons: You have to move your arm to point at the screen (heaven forbid!).

THE NOOB

It's generally a good idea for the less experienced person to drive, even though that adds a bit more stress since someone is watching you type (and typing stage-fright is a real thing, no matter how experienced you are!). I love "the noob." It opens opportunities to ask questions, learn how the team does things, and learn what your pair is particularly good at. Excellent for new hires like me!

Pros: Not as stressful; more opportunity for the driver to ask questions.

Cons: Usually not as satisfying; harder to stay focused (especially for the navigator).

THE DISTRACTED

Like the noob, but you bring your laptop with you. The navigator keeps their laptop closed most of the time—only use it to check syntax, Google solutions, or settle a debate. Don't disengage from your pair for more than a few minutes.

Pros: The navigator can look up things easily.

Cons: It's very easy for the pair to get distracted!

Random tips

Good hygiene goes a long way. Make sure you've showered, put on deodorant, brushed your teeth, eaten a mint, and skipped the garlic.

Eat before or after pairing—not during. Filling your pair's keyboard with donut crumbs is a great way to make them think twice about pairing with you again.

When sharing a monitor, position yourselves so the screen is between you. It's tempting to sit such that the driver's nose is aligned with the middle of the screen, but that tends to shut the navigator out. Better to be inclusive.

Talk a lot while you're pairing, even if you're just "thinking out loud" sometimes. Seek first to understand. It makes your pair feel welcome.

If your first pairing experience isn't everything you dreamed it would be, don't give up! Pair with a few different people before you decide whether pairing is for you. Not everyone likes pairing, but it's something you have to try first to know.

Happy coding! ◎

Not all legacy code needs to be set on fire immediately.

"Not all legacy code needs to be set on fire immediately. Assume the team who wrote it are smart people who did things for a reason. Understand the reason before sharpening pitchforks."

Andrew Semple, Senior Developer | Class of 2009

4 MINUTE READ

Letter from an @ignoredtest

Mauri Edo
QA Engineer
Class of 2002

Dear Developer,

I've been wanting to talk to you for a while now, but words don't always come easy. We've been a great team and had some really fun times together. I still remember the first time I warned you about a minor bug in your code, and how happy you were for having me in your life! Do you remember it? I also remember the first time you refactored me to make me more efficient and how well-written I felt afterwards... ah, great times!

I owe you everything, I know. And I'm thankful for it. I wouldn't exist if it weren't for you. You thought that I was needed so you created me, and from that moment on I am at your service, and I am glad to be, as you gave me a purpose. I want to catch bugs. I want to give you assurance that things will continue to work after your changes. I want to make life easier for you and your team, and you know I can do all those things—I know you do.

But then, with no clear explanation, I started to fail sometimes, for no specific reason. Something broke a little inside of me. I was able to continue functioning almost normally, but I couldn't avoid causing red builds from time to time, it was simply out of my control. I became... flakey.

My flakiness upset you, and I am not angry about that, as it upset me too. I was not reliable anymore. I lost my purpose. At this point, I have to say, it hurts me to remember how you reacted after some weeks of flakiness: instead of investing some love and dedicate a couple of hours to fix me and get me back to a good state, **you annotated me as @ignore and abandoned me** in an immense and desolate codebase.

My statements and assertions can't help but shed a tear when I think of this. For an automated test, being flakey is bad—but at least I passed successfully from time to time, and my failures were a

reminder that I needed some of your magic. But being ignored? My friend, that is simply terrible.

If there is a hell for automated tests, it definitely is being annotated as @ignore and forgotten, being surrounded by successful tests that go green and not being able to join them, watching builds pass by and not pick me up, sitting between infinite lines of code, hopelessly waiting, needing to be fixed and not being taken care of... I would never wish that even to my worst automated test enemies.

Don't get me wrong, I understand that automated tests have a life-cycle, and eventually they get replaced by other automated tests —better and more modern. Sometimes our flakiness can't be resolved, so we need to be removed or replaced, and that's ok. Sometimes the code we are testing is simply retired, so we have no purpose anymore, and that's ok as well. It's part of who we are. But hey: I am code too, you know? I need attention! I need to be implemented and refactored properly so your whole team can benefit from me! I need to be code reviewed by your teammates to spot issues that I might have, because tests can have bugs too!

It's simply unfair to only look after feature code and, when forgotten tests start to fail, annotate them as *@ignore* and continue your day as if nothing happened. It's outrageous!

All I am asking is for you to make up your mind about me. Either fix me or delete me, but do not forget about me! Humans have issues with decisions, as we lines of code know, so if you need to get away with a green build and ignore me for a couple of runs, it's fine. Really!

But if you are not going to come back immediately and find what's wrong with me and why have I been flakey recently, have some decency at least: raise an issue in your bug tracker, so that someone else on your team can give me the attention I need to get back on track and provide some value again. It's not that hard, is it? Please? For all the green builds we've had together?

I sincerely hope we can sort out our differences soon.

Forever yours,
@ignored test

Make a habit of putting your code where others can (and will) use it.

Practice putting your code where people can use it.
Running code only on your local machine is like learning
to skydive without ever putting on a parachute.

Ian Buchanan, Developer Advocate | Class of “Hard Knocks”

Functional programmers vs. functional engineers

Sidney Shek
Developer
Class of 2000

I've felt resistance to functional programming and to functional programmers ever since functional programming became "a thing". While I'm disappointed by the arguments made against functional programming, I am equally (if not more) frustrated that the functional programming point of view has not evolved, leading it to be easily derided as developer religion.

It's time to move the discussion forward. Because no matter our views on functional programming vs. object-oriented programming vs. whatever else, we all need to be functional *engineers*.

Functional engineering is more than writing code. It's about solving problems in the context of the world we work and live in.

There are imperfections in the real world. We run into complexities that can't be modeled nicely yet still need to integrate with other (sometimes poorly-designed) systems. There are constraints in resources and schedules in order to hit business goals and keep the ship afloat. We have to make the compromises necessary to deliver solutions that meet all these requirements, negotiating where possible while standing firm on critical aspects (e.g., safety must be designed in; it cannot be tested in or retrofitted).

> "Functional engineering is more than writing code. It's about solving problems in the context of the world we work and live in.

Our compromises and firm stances need to be data-driven because engineers do not work on reactionary, baseless theories. We hypothesise and rely on quantitative evidence from our own experimentation and the experiences of our teammates. Then we must be able to communicate our findings and resulting reasoning to any audience—even a non-technical audience.

And speaking of non-technicals, we are always building up of set of "soft" skills and business acumen. We'll always work with non-technical people, so it's important to be able speak their language and apply engineering principles across other domains. This, of course, is in addition to continuous learning on the technical side. Our field moves fast. If you rest on what you already know for too long, you'll find that what you know isn't nearly as valuable as it used to be.

Build with urgency and quality

I also want to address the argument against time pressure—e.g., "your can't rush innovation". While time pressure probably is indeed a lousy way to foster innovation, we do live in a world of fast delivery of customer-facing value. Deadlines aren't just some arbitrary date from managers. They are driven by observing the market we operate in. Every company has competitors moving faster. Being a functional engineer means treating fast delivery and high quality as hard requirements. As if your team's survival depended on it (spoiler alert: it does).

Let's not dismiss this reality. Instead, let's look at it as an opportunity to prove our strength.

Those of us who've been lucky enough to choose the technology stacks for their projects also need to remember the responsibility that comes with that freedom. If we don't show improvements in delivery, it becomes harder for even the most supportive team leads to justify letting us make those choices the next time around (and I'm talking about the elephant in the room, here: mandated tech stacks).

So let's work together to support our team leads and prove we deserve the trust they've placed in us.

Deliver solutions, not components

Functional programmers (and software engineers in general) like to tinker with code. But functional *engineers* balance perfecting the code with ensuring the overall system is delivered and maintainable.

Considering the system as a whole is not simply a matter of considering all its smaller parts. There are operational considerations, documentation, integration of all the services, migration, then finally

the roll-out to customers. That's a lot to do beyond just the coding itself. And it all needs to fit into the time between now and the delivery date.

> “We can do ourselves a favor by applying functional programming principles not just to the code, but to the architecture of our systems.

We can do ourselves a favor by applying functional programming principles not just to the code, but to the architecture of our systems. For example, some of the work in perfecting a small microservice ends up going to waste because microservices are meant to be thrown out and rewritten if they no longer meet your needs. By contrast, applying functional concepts such as append-only stores and event-based architectures helps our efforts live much longer. This, in turn, let us spend more energy perfecting the system as a whole, instead of rabbit-holing on individual components.

Apply the principles of functional programming (versus “Functional Programming”)

The term “Functional Programming” means different things to different people. Some fear it, some love it, some believe they are doing it but don't really understand it. I feel the term is getting in the way of everyone understanding that we're all talking about the same principles.

Immutability wherever possible. This isn't just a Functional Programming thing. It's a tenant of building high-performing, highly-concurrent systems.

Less code means fewer bugs. Use available libraries over writing things ourselves not only means less code, it means more time for documentation, migration, and all the other things I mentioned above.

Don't repeat yourself. Find the right abstractions to reduce code duplication.

Think of software as data transformation. Almost all software we write is mainly just reading data from somewhere, allowing code or user transformations, and then saving it somewhere else.

We should seek to apply these fundamental principles in any codebase we work with — whether using a Functional Programming language or not. We will all be better for it. And for the true believers among us, getting our teams to make the next step toward Functional Programming (capital F, capital P) will become easier.

At the end of the day, everyone on your team wants to deliver systems quickly with valuable functionality and as few bugs as possible. As functional engineers, we need to seek the understand the environment we work in, and deliver the whole solution to our customers.

We have lots we can do and try. So let's get to it! ◎

Learn how to make trade-offs. Business value trumps correctness every time.

“Learn pragmatism and how to make trade-offs. This is hard if you come from a field of study where correctness is the only metric. But as a professional developer, delivered business value is the metric.”

Robbie Gates, Senior Architect | Class of 1992

When I grow up, I want to be... a team lead?

Agnes Ro
Senior Team Lead
Class of 2008

It's not quite what I had in mind when growing up, but right now my role title tells me I am a team lead. In the course of my time in this role, I've learned that being a team lead is something my fellow developers are interested in doing someday, so I want to share about my experience. This is what I've learned.

People management

I'll start with the obvious. A team lead is supposed to manage the people on their team. For me, these people are usually developers. Of course, everyone is different and situations arise in a variety of ways, some that I can prepare for or help with and some that I cannot.

Over the years I have learned a lot about how to manage people and how to deal with common problems, but the one common denominator is that I don't always know what to do in 100% of the situations. This is certainly a never-ending learning exercise. It's an interesting aspect of the role that managing no one person is ever the same. Getting a team of different people to work together, keeping them challenged and motivated, ensuring they are performing, improving, and growing, is all quite interesting and challenging. Seeing teams form to reach their potential and watching individuals progress in their career is what I find most rewarding about my job.

“ Seeing teams form to reach their potential and watching individuals progress in their career is what I find most rewarding about my job.

Project management

Generally as team lead, I'm also the one managing the projects and making sure they get delivered on time (<– why is that bit so hard?!).

Delivering the right features, a good user experience, and high quality software on time is damn hard. There are so many equally important projects going on that should have been delivered yesterday. There is never enough time, so you have to constantly prioritise, which often means just saying no.

The illustration below best describes the situation I often find myself in. Just picture me in the middle of this four way tug-of-war. In front of me is a product manager, who tells me all the lovely, but never-ending list of things we must do. To the left I have a designer showing me designs for the best user experience we could possibly build. To my right is the quality engineer who always seems to be scolding me about our test coverage and quality of code. And of course, behind me are the developers who tell me things are going to take longer than estimated, complain about the amount of code debt we have, and explain how we need to build things better.

Now, I may be slightly exaggerating here, but the truth is, everyone is rightfully doing their job and pulling me in the direction they are

supposed to. I just happen to be the person in the middle. The trick is finding the right balance and not falling over, which is quite hard to do. I sometimes worry if I'll even make anyone happy in this constant game of give and take.

When I'm not embroiled in a game of tug-of-war, some other things I do associated with project management include sprint planning, roadmap planning, people planning, risk mitigation planning, deployment planning and planning for planning (yes, it's a thing).

> “Letting go of coding is probably the first thing most devs-who-become-team-leads struggle with.

Having a bird's eye view

This overlaps with project management a bit, but I tend to think about it separately since it's on my mind a lot. As someone not on the ground coding everyday, I am able to take a step back and look at what my team is building more holistically. There is always a constant stream of questions to be asked and decisions to be made, ranging from technical concepts like implementation to other concepts like if the user experience makes sense. A final aspect of this perspective is considering the project dependencies, or how my team is depending on a different team, and how other teams are depending on us.

Wearing different hats

You don't necessarily have to be a team lead to wear different hats, but it's absolutely required of me when the product manager, designer, or QA engineer isn't available. I don't claim to do any of these roles particularly well, but I have to learn to think like these people. It is quite common for questions to arise once development has already started, and it's usually quicker for me to make these small decisions on the spot, which is where putting myself in someone else's shoes comes in handy.

Conversations

As a team lead, my calendar is full of meetings, which I prefer to call conversations, because that is what they are. Besides my 1-on-1s with the people I manage, my mentors, and my managers, my cal-

endar is usually full with conversations about the software we're going to build. There is planning to be done, design reviews to go over, technical discussions to be had, and more.

Coding

To be honest, I hardly code any more. I code 1-2 hours a week if I'm ~~lucky~~ organized. Most of the team leads I know don't code much either. Letting go of coding is probably the first thing most devs-who-become-team-leads struggle with. I learned that as team lead it is not my priority to be writing code. In fact, when I do have to code, I'm not as efficient as another developer because I haven't been following the issue entirely through the workflow. In an attempt to keep up, I try to do random code reviews, set up a dev environment, do a bug fix, or simply pair with whoever is available.

Well, this is my experience being a team lead. Even after a few years, I still find my job stressful, challenging and hard, but for the most part, it is highly rewarding and enjoyable. If one day you find yourself in my position, I hope hearing about my experience will have served you well in the challenges and opportunities you will face. ◎

Working as a team at work is way more fun than working on group assignments.

"Working as a team at work is way more fun than working on group assignments at school. And way more important."

Andre Serna, Development Manager | Class of 2001

On being a woman in tech

Denise Unterwurzacher,
Developer
Class of 2004

I've been working in technical roles for about 13 years—as a sys-admin on an IT team, support engineer, and currently as a developer. The tech sector has grown a ton in that time, which means loads of new people are coming in. Some are fresh out of school, some are seasoned HR or legal or finance professionals transitioning into tech from retail, banking, etc. Either way, they're often coming from environments with a rich ethnic mix and a near 50/50 split of men and women. So walking into a company that is predominantly male and overwhelmingly white can be a bit of a shock (even for white guys, I imagine).

We as an industry have finally started facing up to our lack of diversity in the past few years. It's not a women's issue or an LGBT* issue or an issue for any one group. It's *our* issue, collectively. And it's going to take all of us to reach a point where it's not an issue anymore.

> I'm lucky enough to be comfortable as one of the only women around, and consider it my responsibility to help make other women feel comfortable, too.

"One of the boys"

I ignored the lack of diversity for a long time, as do a lot of young women. I was pretty happy being "one of the boys". (Not all women in tech feel that way, of course.) There've been some awkward moments, nonetheless.

Like the first time I wore a dress to work, which was at least six months after I'd started. People noticed. I could see my teammate's faces change slightly as they realized "Oh right: Denise is a woman." Or the time someone came to standup with a cheerful "Hey, guys!" then saw me and sheepishly asked if it was ok to say "guys" when referring to our team. (I said it was fine with me.) Whilst neither epi-

sode materially changed the way my teammates felt about me, they did make me hyper-aware of being the lone female.

Even when women do feel comfortable as one of the boys, it's not like all problems are solved. Sometimes there's one woman on the team and she's so used to being the only woman that it's become part of her identity. If another woman joins the team, all of a sudden her identity is threatened, which can lead to a hostile environment for the new team member. I've experienced this kind of un-welcome myself, actually.

The sad fact that nobody wants to talk about is that women in tech can be our own worst enemy.

The way we present ourselves can either invite other women to feel welcome, or make them feel intimidated. I'm lucky enough to be comfortable as one of the only women around, and consider it my responsibility to help make other women feel comfortable, too. It's one way I can live two Atlassian values: "Be the change you seek" and "Build with heart and balance".

Confronting bias and privilege

Everyone carries unconscious biases and everyone enjoys privilege of some sort. The trick is being aware of them. (Even my comfort being one of few women in my department is a type of privilege, for example.) But few of us, myself included, have the self-awareness to understand all our biases and all our privileges. So what's a well-intentioned techie to do? We can listen.

When somebody makes you aware of something that's made them feel uncomfortable or marginalised, take that at face value. Even if you can't imagine how someone could possibly be upset by it. We have to remember that we all live different lives filled with different experiences, and that everyone is fighting some kind of personal battle.

Listening—*really* listening—is hard. We're trained by evolution to shut down or get defensive when we're feeling exposed and vulnerable. We have to make the effort to step outside ourselves and stop looking at the situation through the lens of our own personal experiences. That's what opens us up to understanding where the other person is coming from.

We have to overcome our own particular world view—to stop looking at the situation through the lens of our own personal experiences.

And, incidentally, making mis-steps doesn't mean you're a bad person. Embrace them as learning experiences and get back to the business of being awesome.

Everyone is invited

The benefits of diversity are totally obvious and totally impossible to quantify at the same time. We techies want to make software for the whole world, which we'll do a far better job of if the world is well represented on our teams. And each of us embodies a unique mixture of the human experience informed by our ethnicity, gender, age, religion, personal philosophies, hobbies… all of it. There are things you'll bring with you *as a human* that I won't, and vice versa.

It'd be great if we look back in ten or twenty years and wonder why on Earth we ever had women in tech or Latinos in tech events because it'll just seem absurd. But we're not there yet. The more that we can make tech an inviting place, the more diverse tech will become, and things will spiral upward from there.

I hope you'll join me in working to make tech more inclusive, whether you've been here for ages or are just arriving. ◎

Our industry is small. Behaving badly can come back to haunt you.

"Don't burn bridges. Our industry is surprisingly small and behaving badly in one job can easily come back to bite you at a different company."

Gillmore Davidson, Developer | Class of 2000

Maintaining a growth mindset

Steve Haffenden
Lead Developer
JIRA
Class of 1996

The concept of a growth mindset (as opposed to a fixed mindset) is a popular topic lately, and I sometimes say to myself "Self, you need to embrace the growth mindset. The growth mindset is cool." Why? Because the aptly-named growth mindset helps us grow in whatever we pursue—especially in our work. When we're growing, we're more productive. And when I'm productive, I just *feel* better. Know what I mean?

I believe my inability to stay in a growth mindset keeps me from heading home smiling more often. So I'm writing this to call myself out for moments of fixed-mindedness (maybe you'll recognize a bit of yourself in those moments as well), and share my techniques for working through them. If we can get better at catching ourselves *in the moment* as our brains revert to a fixed mindset, we can pull ourselves back into growth-iness and ultimately be more fulfilled.

For those new to the concept, here's the difference between a fixed and growth mindset:

FIXED MINDSET		**GROWTH** MINDSET
Avoids	CHALLENGES	Embraces
Loses focus	OBSTACLES	Persists in spite of
"It's fruitless."	EFFORT	"No pain, no gain."
Ignores	CRITICISM	Learns from
Threatened by	SUCCESS OF PEERS	Inspired by
Fails to reach potential		**Has higher goals, achieves more**

This whole concept struck a chord with me because every time I try, I get only so far before encountering something difficult and I stop. Behaving this way is all well and good when it comes to rock climbing, or judo, or kite surfing (among the many activities I've failed to master). But when it comes to my work, it's a problem.

Challenges

Here's an example of when I wasn't in the right mindset: learning Java. I studied a bit of Java at university, and have worked with it in almost every role I've had (including my current gig as a developer on JIRA Software), and yet I still find it hard to get my head around.

When I'm faced with a challenging coding problem, I have a tendency to work over one particular aspect of it again and again until I conclude that it's impossible or that I'm incapable of solving it. I sort of throw my hands up in surrender. In other words, I'm in a fixed mindset and avoid the problem by giving up.

So, lately, I try to recognize that avoidance and take a step back. I walk through the problem once again, bit by bit, until I see a way forward. I look for new ways to think about the problem. And for me, it always helps to talk these steps out—out loud, not in my head, and yes: my teammates have learned to ignore me in these moments or just put their headphones on. I jot down clues as they emerge. Soon enough, I begin to find the root causes of the problem.

Obstacles

My first impulse when I hit a roadblock is to immediately divert my attention to something else. Whether it's clicking a tempting bookmark in my browser, nipping out to grab coffee, or just about anything to delay staying at it and fighting through the discomfort. A coworker calls this "instant gratification monkey syndrome." It's the idea that we'll do just about anything other than the actual task at hand if it has suddenly become challenging, and especially when there are so many tempting diversions (articles, videos, email, social) mere clicks away.

In *Zen and the Art of Motorcycle Maintenance,* Robert Persig talks about the "gumption trap"—the phenomenon of knowing what needs to be done, but lacking the motivation to do it. His solution

is to stand up, put down whatever project is sucking the gumption out of you, walk away, and return when you're feeling more inspired. Now, granted, this isn't often a viable option for projects at work, but just being able recognize what's happening helps me make peace with it, and usually un-traps just enough of my gumption to do one more thing on the project. Then another one more thing. Then another. And so on. Eventually, my momentum builds up and I'm back in the groove.

Effort

I have similar troubles with cycling. When my alarm goes off at 5 A.M., the thought of going out on the bike is dreadful. The temptation to stay in bed is strong, strong, strong—damn you, instant gratification monkey!—but I also know this feeling won't last and I'll feel guilty later.

But on the mornings when I get up and ride, I *feel* better. It's hard work, yes, and the reward isn't instantaneous. But when the ride is over, my body feels better and my spirit is stronger. I'm proud of myself, and that goes a long, long way.

So when I find myself reaching for the snooze button, I think of this feeling. I make a conscious effort to stop and think about the implications. Is going back to sleep right now giving up too easily? Probably. When I take a shower this morning, would I rather be washing away pangs of regret, or the sweat of accomplishment? The question basically answers itself.

Criticism

I find it useful to review each day and assess how I've done. As you now know, I tend to judge my days based on my mood at the end. My worst days are those when I simply haven't managed to get things together, I've procrastinated and wasted time, and I find myself with very little to look at positively. And on those days, I'm a harsh self-critic.

So I've learned to ask myself two questions: *What was the best part of my day? What went wrong today and how can I make sure it doesn't happen again?*

These questions help me focus on the good *and* bad aspects of my day. Plus, they help me internalize strategies for creating more good days going forward.

Turns out, variations on those questions help me process criticism when it comes from people I work with. Assuming their critique is valid (and it usually is), I can hold a mini-retrospective with myself: *Which aspects of my work am I proud of? What can I do to improve the aspects I'm less proud of?*

Dealing with criticism is the dark side of growth, I guess.

Success of others

My team contains some of the smartest people I've ever had the pleasure of working with. But I didn't always think this way.

For the longest time I was fiercely jealous of those that were more successful or seemingly more intelligent than I. But our team's culture of openness and honesty helps me to understand how the work I'm doing relates to the work my peers are doing, and how it all comes together to create something awesome.

So where I once perceived those smarter than I as adversaries, I now see dedicated teammates with a passion for learning. Where I once

saw managers who were more successful, I now see peers with skills that complement mine, and together we're doing some of the best work of our lives.

This shift in mindset has come from open and frank interactions that I've had with fellow developers during peer reviews, in "one-on-ones," in hallway conversations, and through access to literally everyone in the company.

"A growth mindset opens me up to more creativity and possibility. I discover I'm able to set higher goals and actually reach them.

It might be impossible to retrofit a similar level of openness and commitment to authenticity into an existing team, but it's not hard to use these concepts as personal mantras that influence the way we interact with our peers. We can choose to be more transparent or more helpful than might be required. In other words, we can choose to go the extra mile, we can choose to be quick to praise and slow to criticize. We can assume the people we're working with are not idiots. We can seek first to understand.

Growing at work—and outside work, too

I've been guilty of having a fixed mindset far too often and I'm working to change that. Things like wisdom and achievement are the products of not avoiding hard work, and having the right mindset. We live in a society where the instant gratification monkey sits on everyone's shoulder. But the pleasures that result from determination and hard work far outweigh anything that stupid chimp has to offer.

Maybe all this seems obvious to you, in which case, you're lucky. For me, I've found that taking the time to assess what I'm doing is tremendously effective in keeping me on track at work. When I feel myself losing focus I try to remember that short-term frustration will likely result in long-term improvement of my abilities—if I stay with it. A growth mindset opens me up to more creativity and possibility. I discover I'm able to set higher goals and actually reach them. ◎

Move jobs or roles regularly to understand all the variations out there.

"Move companies or roles regularly when you are young and don't have commitments. It helps you understand all the variations out there in the 'real' world."

Mike Minns, Development Manager | Class of 1996

Just bloody do it

Gilmore Davidson
Developer
Class of 2000

Want to put up your hand for a new project, task or conference? Hesitating because you think you might be too new or unqualified? Stop hesitating and just bloody do it. Put the onus of choice on the decision makers. If it turns out that you are eligible, great! If not, what have you lost?

From my time as a developer, it's become clear that a lot of us thrive on this notion of a "do-ocracy": the idea of ownership and individual contribution and being the change you seek. When it comes to things you want to try, if you don't indicate your interest, how is anyone supposed to know? No one will think any worse of you for trying something new. In fact, stretching yourself and expanding your repertoire of skills should be encouraged and celebrated in any company.

You may have a fear of rejection, but in a friendly, supportive environment, your fears will subside. And if you do happen to get knocked back, don't take it too hard—if someone else is chosen over you, it's usually not personal, and for a good reason. Just by putting your name forward, you've already done the most important thing—you've indicated your interest.

Opportunities often come up again, and by showing you're keen the first time around, you greatly increase your chances of being picked the next time. Personally, I know that when I'm making a decision on who to choose for a task, it's a hell of a lot easier to pick someone who's already demonstrated their eagerness to do it.

So what are you waiting for? Stop dithering, quit hesitating. Just. Bloody. Do it. ◎

Working on side projects is a fun way to learn the full stack.

"Working on side projects is a fun way to learn and nothing beats experience. If you build an entire application from scratch, you're forced to understand the full stack to put it together."

Lori Lee, Developer | Class of 2011

RESOURCES & READING

Books

The Pragmatic Programmer
by David Thomas and Andrew Hunt

Why Programs Fail: A Guide to Systematic Debugging
by Andreas Zeller

The Clean Coder: A Code of Conduct for Professional Programmers
by Robert C. Martin

Don't Make Me Think, Revisted: A Common Sense Approach to Web Usability
by Steve Krug

Research paper

Promiscuous pairing, and the beginner's mind: Embrace inexperience
by Arlo Belshee

Websites

Coding Horror

The Daily WTF

Music for Programming