

SOFTWARE ENGINEERING FUNDAMENTALS: PART I

NATHANIEL SCHUTTA
@NTSCHUTTA
NTSCHUTTA.IO

Fundamentals matter.

What do professional athletes
work on over and over?

The things they were taught in
the beginning of their career.

Stance, grip, alignment for golfers.

Shooting form, dribbling,
layups for basketball players.

They don't spend much time on
the highlight reel stuff.



<https://twitter.com/DaliaShea/status/1367827097109078019>

There are any number of paths
to become a software engineer.

Comp Sci undergrad degrees.

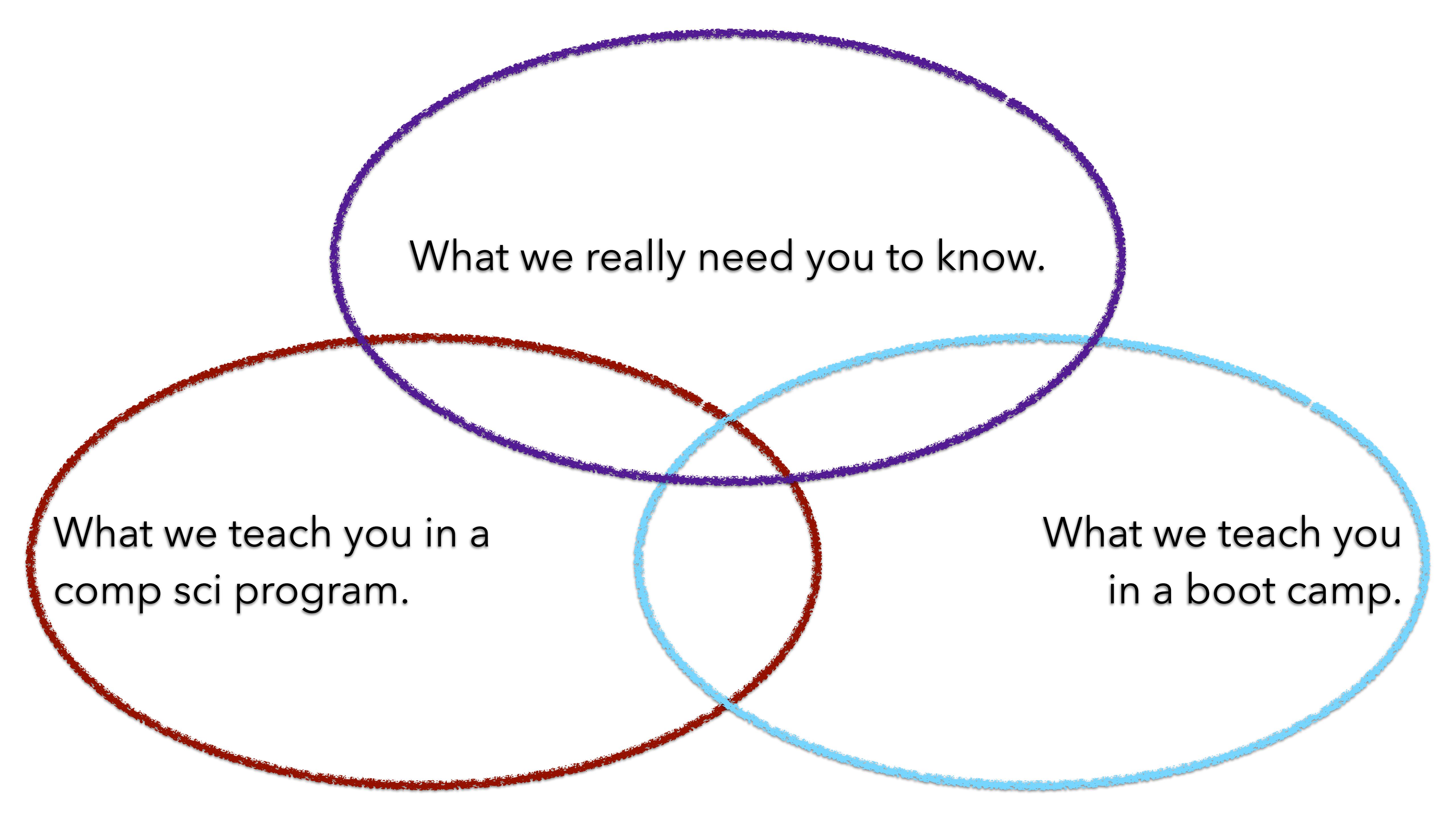
Boot camps.

Self taught.

Doesn't really matter.

But there is a huge gulf
between what we teach you...

And what you need to be successful.



What we really need you to know.

What we teach you in a
comp sci program.

What we teach you
in a boot camp.

Sorry about that!

Fundamental goals are different.

Undergrad programs prepare
you for...graduate programs.

Algorithms, language design,
compiler theory, OS.

Boot camps cram specifics into
a ***very*** short time frame.

Frameworks, language du jour.

More practical? Maybe.

More transitory? Maybe.

Some of these bootcamps
are run by Universities!



#probablyfine

Mostly, we teach you how to code.

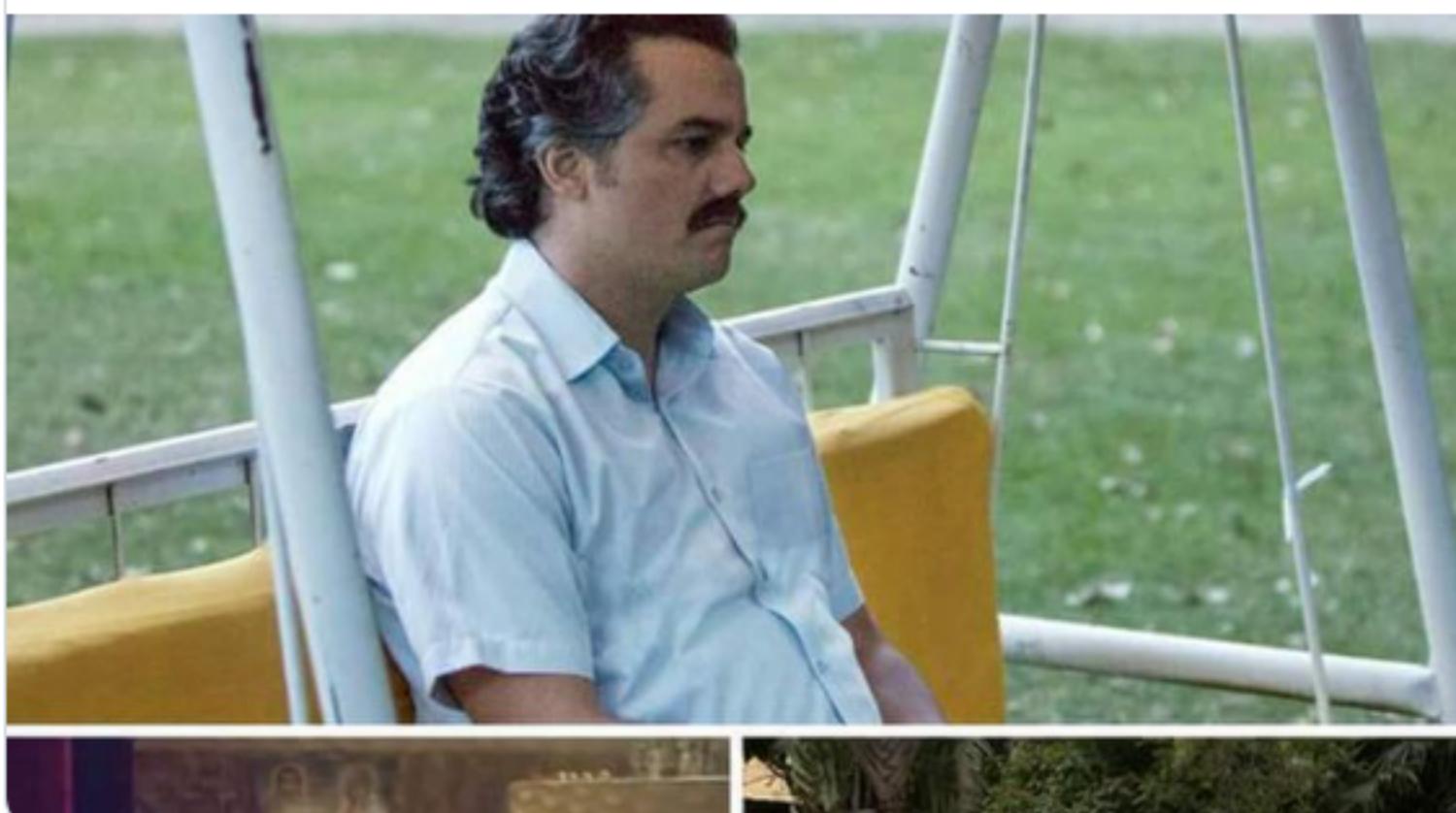
You learn a language or three.

Become familiar with our good
friends foo and bar.

 **Corneil du Plessis. he/him** 
@corneil

foo or bar? 

 **Vlad Mihalcea** @vlad_mihalcea · Aug 26
Software developer choosing a good name for a new method or variable.



12:58 AM · Aug 27, 2021 · Twitter for Android

1 Retweet 9 Likes

<https://twitter.com/corneil/status/1431133997124423680>

Learn a bit about debugging.

But a lot of important things are left out. For various reasons.

Time for one.

Undergrad degree *might* put you at
an advantage...for a year or two.

But things usually even out.

Don't forger, the "traditional"
undergrad comp sci program?

Didn't exist at most schools
until relatively recently.

And early on, they were
often math heavy...



<https://twitter.com/EmilyKager/status/1378009547734802433>

Never apologize for how you
became a developer.

Ultimately it is about problem
solving, tinkering, creativity.

If you have the mindset,
you have it. Period.

Never forget, you know more
(about something) than someone...

And someone knows more
(about something) than you...

Critical we learn from mistakes.

We must talk about
them. Share them.

Some companies are
better at this than others.

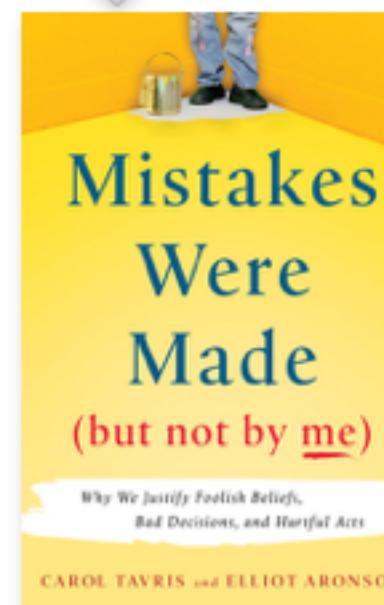
goodreads

Home My Books Browse ▾ Community ▾ Search books Sign In Join

Goodreads helps you keep track of books you want to read.

Start by marking "Mistakes Were Made (But Not by Me): Why We Justify Foolish Beliefs, Bad Decisions, and Hurtful Acts" as Want to Read:

Want to Read ▾



Mistakes Were Made (But Not by Me): Why We Justify Foolish Beliefs, Bad Decisions, and Hurtful Acts

by Carol Tavris, Elliot Aronson

★★★★★ 4.01 · Rating details · 24,216 ratings · 1,242 reviews

Why do people dodge responsibility when things fall apart? Why the parade of public figures unable to own up when they screw up? Why the endless marital quarrels over who is right? Why can we see hypocrisy in others but not in ourselves? Are we all liars? Or do we really believe the stories we tell?

Renowned social psychologists Carol Tavris and Elliot Aronson take a compe ...[more](#)

GET A COPY

Amazon Online Stores ▾ Libraries

Hardcover, 304 pages
Published May 7th 2007 by Houghton Mifflin Harcourt (first published 2007)
[More Details...](#) Edit Details

FRIEND REVIEWS

To see what your friends thought of this book, [please sign up](#).

READER Q&A

To ask other readers questions about *Mistakes Were Made (But Not by Me)*, [please sign up](#).

Popular Answered Questions

Is there a Spanish edition for this book?
Like · One Year Ago · [Add Your Answer](#)

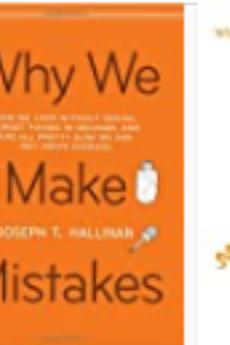
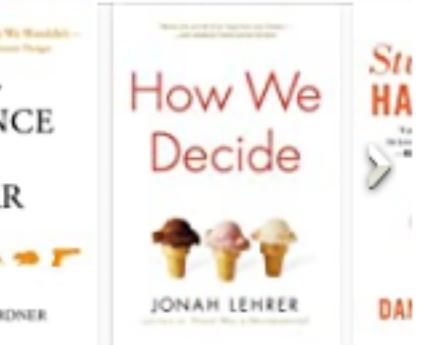
aml I don't think so....[more](#)

See 2 questions about *Mistakes Were Made (But Not by Me)*...

Share   

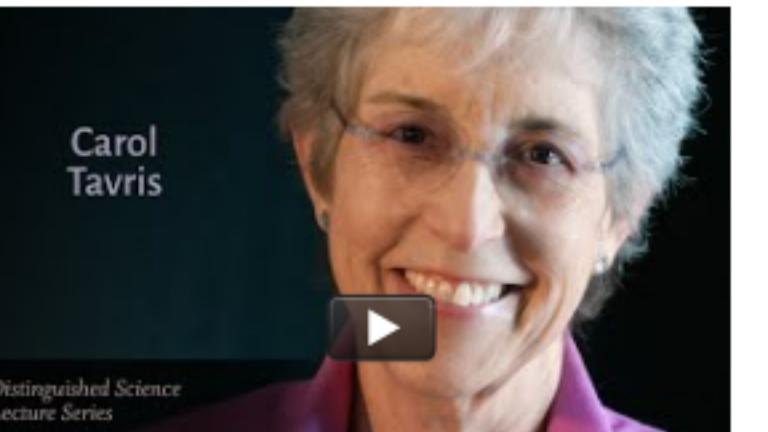
Recommend It | Stats | Recent Status Updates

READERS ALSO ENJOYED


[See similar books...](#)

VIDEOS ABOUT THIS BOOK



[Add a comment](#)
[More videos...](#)

GENRES

Psychology	1,035 users
Nonfiction	701 users
Science	205 users
Self Help	129 users
Sociology	80 users
Business	62 users
Self Help > Personal Development	55 users
Philosophy	47 users

 **Björn Fahller** 😊 ✈️ ✈️ @  [@bjorn_fahller](#) ...

If you're wondering why I sometimes (often?) tweet about stupid mistakes I've made, it follows from not very long ago having had a private pilot license, buzzing the sky for fun.

Mini thread (my 1st, I think.)
1/7

2:56 PM · Feb 4, 2021 · TweetDeck

16 Retweets 4 Quote Tweets 41 Likes

Reply

 **Tweet your reply**

 **Björn Fahller** 😊 ✈️ ✈️ @  [@bjorn_fahller](#) · Feb 4 ...
Replies to [@bjorn_fahller](#)
In the dark ages of aviation, it was dangerous. Part of it was, no doubt, because the aircraft of the time were flimsy and the physics of flight was poorly understood, but primarily it was a matter of attitude.
2/7

1 1 4 ...

 **Björn Fahller** 😊 ✈️ ✈️ @  [@bjorn_fahller](#) · Feb 4 ...
Attitude? Yes, attitude. There was this stiff upper lip attitude that the trade was dangerous and only people of the right stuff made it through. If you didn't make it, you weren't up to it. Simples.
3/7

1 1 4 ...

https://twitter.com/bjorn_fahller/status/135743283665156673

 **Angel/Samantha** @Aliofonzy43 · Aug 2

I got rejected from a 4 month long interview process and feedback was to not put one little git commit message saying "please work". And to not have dead code that I explained I wanted to show thought process because that's important for me to show.

336 383 2.3K 

 **Spencer Gibb** @spencerbgibb

Replying to @Aliofonzy43

You should see some of the commit messages on [@springcloud](#), [@MGrzejszczak](#), [@ryanjbaxter](#), [@olga_maciaszek](#), and I sometimes say some wacky things. I'll have to see if I can find an example.

8:01 PM · Aug 4, 2021 · TweetDeck

4 Likes

 Tweet your reply 

 **Spencer Gibb** @spencerbgibb · Aug 4

Replying to @spencerbgibb @Aliofonzy43 and 4 others

Here's a quick one

spring-cloud/spring-cloud-jenkins-jobs

remove jdk 15, third time's a charm

16 lines changed +2 -14



<https://twitter.com/spencerbgibb/status/1423086677375991809>

Needs to come from
senior people too!

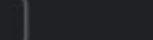
Create a safe environment.

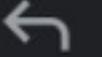
 emily freeman 
@editingemily

Whoever you are, we've got your back. Could have been any of us, honestly.

Breathe. It's gonna be OK.

 Julie Hubschman @juliehubs · Jun 17
Shout out to the HBO Max Engineer who is currently having a full blown panic attack
[Show this thread](#)

Integration Test Email #1  

 HBO Max 8:46 PM
to me ▾  

This template is used by integration tests only.

8:33 PM · Jun 17, 2021 · Twitter for iPhone

190 Retweets 12 Quote Tweets 2,197 Likes

<https://twitter.com/editingemily/status/1405700159967678464>

Not easy mind you.

So what **don't** we teach you?

How to work with others.

How to read code.

How to write code for
other humans.

Perfect is the enemy of the good.

How to test.

What to test.

Why to test!

How to solve problems.

How to figure out what our
customer **really** wants.

How to work with legacy code.

We teach (mostly) on green field.

Much like your physics teacher
saying “ignore air resistance”.

That doesn't work so
great in the real world.

We (mostly) work with existing,
some would say, heritage apps.

In school, projects are short, a few months, maybe less.

In the real world? Projects don't end.

Ok, like art, it is done
when it is abandoned!

Code spends most of its life in
the maintenance phase.

And our first exposure is almost
always live fire exercises.

So what does it mean to be a
software engineer?

It's far more than
banging out code.



EMBRACE THE LAZY
PROGRAMMER ETHOS.

Beware the brute force approach.

We often start there.
And hopefully iterate.

The performance problem.

Years ago I helped a client with
a performance issue.

Application processed widgets.

In an overnight batch run.

As they became more
successful, more customers...

More widgets. Couldn't process
overnight anymore.

Let's review the architecture!

I asked a simple question:

Does the processing of widget A
have anything to do with widget B?

No. Some aggregation at the end.

Could you process them in parallel?



Set of workers, grab the next
widget, process it...

Yes, yes we could.

I didn't have some special insight.

Just wasn't bound by...

“That’s how we’ve always done it”.

Doesn't mean it is the
right way to do things!

Could have been an accident.

Three monkeys in a cage!

There's probably a better way.

Pays to understand Big O notation.

Some algorithms are
better than others!

Bubble sort, for example, $O(n^2)$.

It's Not Great Dan.

Consider best case, worst
case and average case.

Choose wisely!

Odds are, you aren't the first
person to solve this problem.

Don't be afraid to search!

Ten minutes of searching could
save you days of work.

If it *feels* like there should be an easier way..there probably is!

The capitalization assignment.

Early in my career, I was
given a really simple task.

Capitalize the billing addresses.

Modify this routine and you
should call this function.

The capitalize function...was a mess.

It should take one argument -
the string to capitalize.

Except it didn't. It had
multiple arguments.

How long would it take to
decipher the home grown thing?

Hours?

Seemed like something the
language should solve.

I searched. Sure enough, built
in function. I used it.

Following week, team meeting,
described what I did.



Every head turned to look at me.

I was the least experienced
developer in that room.

No one else knew that feature
was baked into the language.

Who knows when it was added.

“That’s how we’ve always done it”.

Two lessons.

If you think there should be
an easier way, look for it.

Never underestimate the
value of fresh eyes.

Don't solution too quickly.

What is the real ask?
The real problem?

Give yourself some space to think.

We don't get many opportunities
to just think, uninterrupted.

Bugs happen.

If we find them in
production, costly to fix.

Cheaper to find them in QA.

Even cheaper if we find them
during development.

But what if we solved it in design?

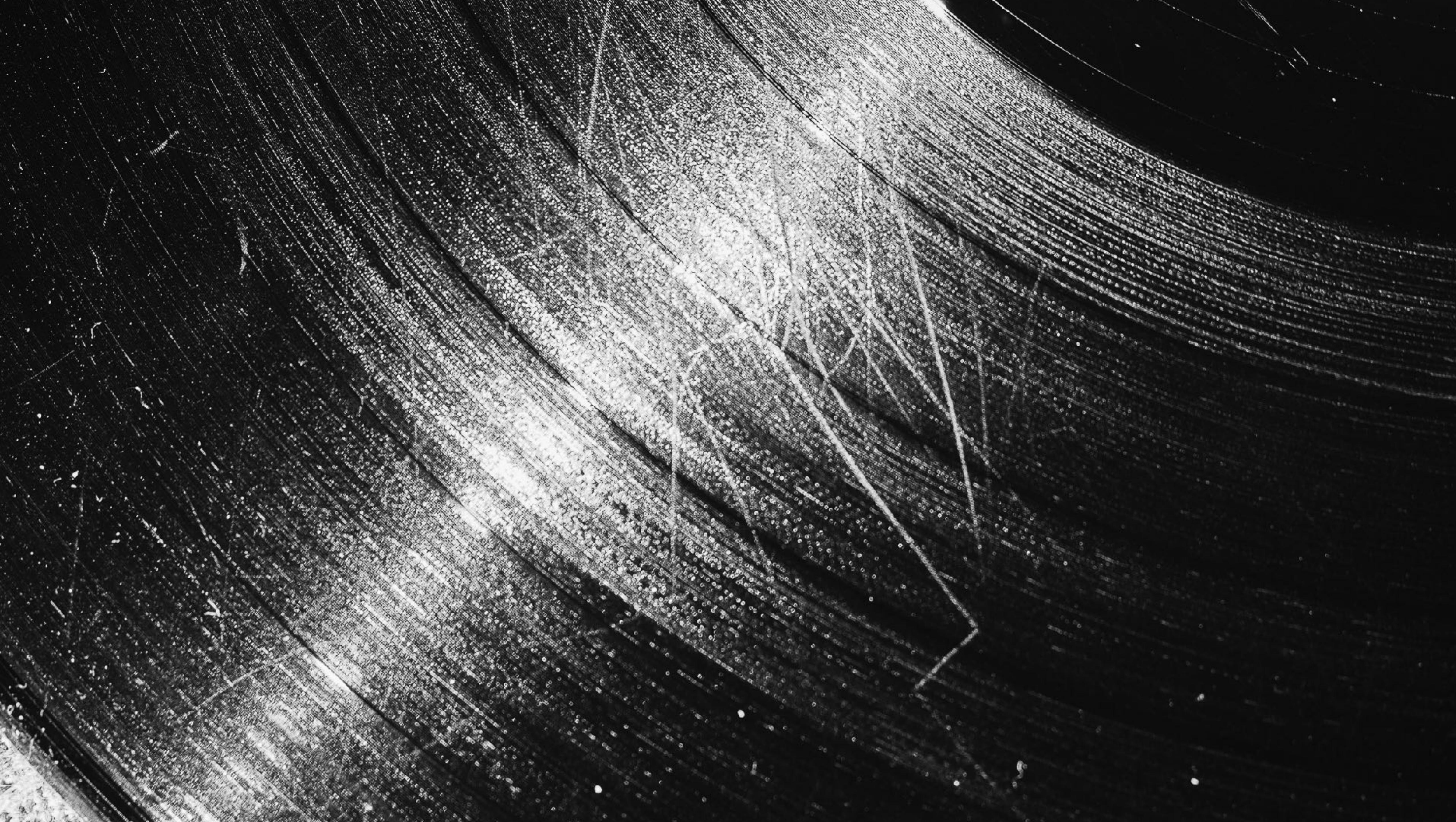
“...most of the big problems we have with software are problems of misconception. We do not have a good idea of what we are doing before we do it. And then, go, go, go, go and we do everything.”

-Rich Hickey

Problems of misconception.

Client vs. customer.

That'll work for a client...
but not a customer.



Client and customer are two
similar, but distinct things.

No amount of testing, no type
system, finds misconceptions.

Rather than work around a
problem? Avoid it in the first place.

Focus on solving problems.

Requires practice!

What **problem** does your
application solve?

Define the context
and the constraints.

What do we know?

What don't we know?

What are the related problems?

Noodle on solutions.

Think through the tradeoffs.

Waking mind vs
background mind.

L-mode vs. R-mode.

Get more sleep!



<https://twitter.com/kentbeck/status/9345238812>

Sleep matters.

We're not sure why we sleep.

Quite vulnerable...

Not about “rest.”

Brain is incredibly active.

Key to learning.



Aids memory formation.

Transfer of information.

With the inputs largely silenced,
brain can get to work.

Take out the garbage.

Keep the relevant bits.

Strengthens memory.

Makes connections.

Use it to solve problems.

Lack of sleep hurts performance.

Naps improve it.

26 minutes = 34% improvement.

Neglect sleep at your own peril.

Sleep on the solution.

How does it feel now?

Don't be surprised if you wake up
with a solution to something else!

Feature not a bug.

But how do we get time for that?

If I just hang out in a hammock
all day, I won't have a job!

Defend your schedule.
Block out time to think.

Manager vs. maker schedule.

Not Secure — paulgraham.com

PAUL GRAHAM

MAKER'S SCHEDULE, MANAGER'S SCHEDULE

"...the mere consciousness of an engagement will sometimes worry a whole day."

— Charles Dickens

July 2009

One reason programmers dislike meetings so much is that they're on a different type of schedule from other people. Meetings cost them more.

There are two types of schedule, which I'll call the manager's schedule and the maker's schedule. The manager's schedule is for bosses. It's embodied in the traditional appointment book, with each day cut into one hour intervals. You can block off several hours for a single task if you need to, but by default you change what you're doing every hour.

When you use time that way, it's merely a practical problem to meet with someone. Find an open slot in your schedule, book them, and you're done.

Most powerful people are on the manager's schedule. It's the schedule of command. But there's another way of using time that's common among people who make things, like programmers and writers. They generally prefer to use time in units of half a day at least. You can't write or program well in units of an hour. That's barely enough time to get started.

When you're operating on the maker's schedule, meetings are a disaster. A single meeting can blow a whole afternoon, by breaking it into two pieces each too small to do anything hard in. Plus you have to remember to go to the meeting. That's no problem for someone on the manager's schedule. There's always something coming on the next hour; the only question is what. But when someone on the maker's schedule has a meeting, they have to think about it.

For someone on the maker's schedule, having a meeting is like throwing an exception. It doesn't merely cause you to switch from one task to another; it changes the mode in which you work.

I find one meeting can sometimes affect a whole day. A meeting commonly blows at least half a day, by breaking up a morning or afternoon. But in addition there's sometimes a cascading effect. If I know the afternoon is going to be broken up, I'm slightly less likely to start something ambitious in the morning. I know this may sound oversensitive, but if you're a maker, think of your own case. Don't your spirits rise at the thought of having an entire day free to work, with no appointments at all? Well, that means your spirits are correspondingly depressed when you don't. And ambitious projects are by definition close to the limits of your capacity. A small decrease in morale is enough to kill them off.

Each type of schedule works fine by itself. Problems arise when they meet. Since most powerful people operate on the manager's schedule, they're in a position to make everyone resonate at their

For managers, schedule is just a series of one hour blocks.

One hour is the same
as any other hour.

Need to meet, find an open slot.

But that doesn't work for makers.

What unit of time do you think
in? Half a day? Full day?

A thirty minute meeting could
blow up the entire day.

Broken the day into pieces that
are too small for hard work.

No point in starting anything,
just have to context switch out.

Software is ultimately about
loading a problem into your brain.

 Kent Beck 
@KentBeck

The goal of software design is to create chunks or slices that fit into a human mind. The software keeps growing but the human mind maxes out, so we have to keep chunking and slicing differently if we want to keep making changes.

7:16 AM · Jan 27, 2021 · Twitter for iPhone

633 Retweets 44 Quote Tweets 2,191 Likes

 Tweet your reply 

 Kent Beck  @KentBeck · Jan 27
Replies to @KentBeck
This implies that software design is a human process with technical support, done by humans for humans. As shiny as the technical support is it took me a while to acknowledge that messy people stuff.

6 31 216

 Kent Beck  @KentBeck · Jan 27
I'll add that "fits into" and "maxes out" are metaphors and not precisely true, but they help make sense of a healthy, growing urge to re-design. Knowledge is not a quantity & the brain is not a space & we work together.

4 13 85

 Kent Beck  @KentBeck · Jan 27
I am continuing work on "Tidy First? An Exercise In Empirical Software Design" as a [@SubstackInc](#). No paid tier yet, but that's where book chapters will be appearing. Sign up here: kentbeck.substack.com.

 Software Design: Tidy First?
Software design is an exercise in human

<https://twitter.com/KentBeck/status/1354418068869398538>

Think about debugging.

You've created a
model in your brain.



And then someone interrupts you...

Some interruptions are more
equal than others.

In context vs. out.

Project rooms work.

Everything is in context.

Easy to tune out.

Allows for the adjacent possible.

Overhear someone saying "so
we'll drop the prod table..." etc.

And you can interject!

Turn off interruptions.

Leverage Do Not Disturb!

Email, Slack, etc.

Stay on target!

Don't be afraid to run
apps full page.

Set expectations.

Immediate response?

Really?

Email apnea.

Email bankruptcy.



No meeting Friday.

Quiet time/office hours.

How about scheduling
meetings for <1 hour?

Pomodoro technique.

Pick a task.

Set a 25 minute timer.

Work.

Take a break!

Rinse and repeat.



Take a longer break!

francescocirillo.com

ABOUT US FRANCESCO CALENDAR 2021 ARTICLES PRESS FAQ

Francesco Cirillo
Work Smarter, Not Harder

checkout
in EUR

POMODORO® POMODORO® APP TRAINING CERTIFICATIONS EVENTS SERVICES MEMBERSHIPS BOOKS

Improve your productivity skills in this COVID period.
Our courses are 50% off until the end of 2021.

The Pomodoro® Technique

Over 2 million people have already used the Pomodoro® Technique to transform their lives, making them more productive, more focused and even smarter.

Do more and have fun with time management

TERMS OF SERVICE

OBJECTIVES BENEFITS THE CORE PROCESS THE DAILY PROCESS THE WEEKLY PROCESS RESOURCES TESTIMONIALS FAQ

The Pomodoro® Technique is organised into six incremental objectives

The Pomodoro Core Process

Copy link

i

Treat yourself.

Best is subjective.

There are **many** ways
to solve a problem.

Never lose sight of the problem
you are trying to solve.

Don't be too quick to
jump to solutions!

What is the actual ask here?

What is the real weed?

Drive to the why, the itch.

Don't be afraid to ask questions!

What would you say you're
trying to do here?

Talk to your customers!

Talk to your analysts.

Customers ask for ~~X~~ but often
need ~~T~~. But they can't see it.

“That’s how we’ve always done it”.

“...most of the big problems we have with software are problems of misconception. We do not have a good idea of what we are doing before we do it. And then, go, go, go, go and we do everything.”

-Rich Hickey



Sorry, it isn't real. And besides,
you must be worthy to lift it...

Also applies to technologies.



Bring me a problem!

Vital we have a broad perspective.



Pick the right tool for the job!

Beware the over
engineered solution.



James Farmer
@JamesFarmer87

Well this has made my day.



10:05 AM · Sep 19, 2020 · Twitter for iPhone

<https://twitter.com/JamesFarmer87/status/1307335098757451776>

Be realistic!

We can't all be a third
of internet traffic!

YAGNI.

Yagni

26 May 2015



Martin Fowler

- ❖ PROCESS THEORY
- ❖ PROJECT PLANNING
- ❖ EVOLUTIONARY DESIGN
- ❖ PROGRAMMING STYLE

Yagni originally is an acronym that stands for "You Aren't Gonna Need It". It is a mantra from [ExtremeProgramming](#) that's often used generally in agile software teams. It's a statement that some capability we presume our software needs in the future should not be built now because "you aren't gonna need it".

Yagni is a way to refer to the XP practice of Simple Design (from the first edition of [The White Book](#), the second edition refers to the related notion of "incremental design"). [1] Like many elements of XP, it's a sharp contrast to elements of the widely held principles of software engineering in the late 90s. At that time there was a big push for careful up-front planning of software development.

Let's imagine I'm working with a startup in Minas Tirith selling insurance for the shipping business. Their software system is broken into two main components: one for pricing, and one for sales. The dependencies are such that they can't usefully build sales software until the relevant pricing software is completed.

At the moment, the team is working on updating the pricing component to add support for risks from storms. They know that in six months time, they will need to also support pricing for piracy risks. Since they are currently working on the pricing engine they consider building the presumptive feature [2] for piracy pricing now, since that way the pricing service will be complete before they start working on the sales software.

Yagni argues against this, it says that since you won't need piracy pricing for six months you shouldn't build it until it's necessary. So if you think it will take two months to build this software, then you shouldn't start for another four months (neglecting any buffer time for schedule risk and updating the sales component).

You Ain't Gonna Need It.

Avoid speculation.

“Prediction is very difficult,
especially if it's about the future.”

-Niels Bohr (attributed)

But we have to walk a fine line.

Don't want to paint
ourselves into a corner!

But we don't want to build a
bunch of stuff we throw away.

Or worse...keep around, unused.

[KonMari Philosophy, 6 Rules of Tidying](#)

Rule 6: Ask Yourself If It Sparks Joy

Only you can know what kind of environment makes you happy – this is the underlying principle of the KonMari Method™. It's also what sets it apart from other tidying techniques that rely on strict guidelines instead of personal criteria. You've probably heard “Discard anything you haven’t used in two years,” or “Every time you buy something new, get rid of something old.” But Marie’s method is different.

In the KonMari Method™, your *feelings* are the standard for decision making – specifically, knowing what sparks joy. To determine this when tidying, the key is to pick up each object one at a time, and ask yourself quietly, “Does this spark joy?” Pay attention to how your body responds. Joy is personal, so everyone will experience it differently; Marie describes it as “...a little thrill, as if the cells in your body are slowly rising.”

Through the process of selecting only those things that inspire joy, you can identify precisely what you love – and what you need.

Need a refresher? Review the [6 Rules of Tidying](#).

Photo by M.K. Sadler for KonMari Media, Inc.; video by Yayo Ahumada for KonMari Media, Inc.



Share



Does this code spark joy?

No? Thank it...and let it go.

READING CODE.



Despite the way we teach...

You'll spend far more time
reading code than writing it.

Yet we jump right into writing.

Not how you'd learn
French or Italian is it?

Reading code is practically a developer's nightmare.



()

—

Why do we dislike
other's code so much?

When we read someone's code,
we have two problems.

We need to understand the domain, the problem at hand.

And we have to see that
problem through another's eyes.

It is the later that often
frustrates us the most.

We've all read some code and
wondered "what idiot wrote this?"

Only for you to realize...***you***
were the idiot the wrote this!



<https://twitter.com/pawpoise/status/38010102002888704>

We're also dealing with
patches on top of patches.

Often made with inadequate
time or understanding.

Ship it!

There's also the IKEA effect.

We place a higher value on
things we've created.

One study found people would pay
63% more if they assembled it.



The mere-exposure effect.

Familiarity may breed contempt...

But we tend to prefer
things we are familiar with.

Part of the dogmatism around
programming languages.

Developers tend to think time began
with the first language learned.

“Why does Java need these
new fangled Lambdas?”



Blub paradox.

Not Secure — paulgraham.com

Home Essays H&P Books YC Arc Bel Lisp Spam Responses FAQs RAQs Quotes RSS Bio Twitter

PAUL GRAHAM

BEATING THE AVERAGES

Want to start a startup? Get funded by Y Combinator.

April 2001, rev. April 2003

(This article is derived from a talk given at the 2001 Franz Developer Symposium.)

In the summer of 1995, my friend Robert Morris and I started a startup called [Viaweb](#). Our plan was to write software that would let end users build online stores. What was novel about this software, at the time, was that it ran on our server, using ordinary Web pages as the interface.

A lot of people could have been having this idea at the same time, of course, but as far as I know, Viaweb was the first Web-based application. It seemed such a novel idea to us that we named the company after it: Viaweb, because our software worked via the Web, instead of running on your desktop computer.

Another unusual thing about this software was that it was written primarily in a programming language called Lisp. It was one of the first big end-user applications to be written in Lisp, which up till then had been used mostly in universities and research labs.
[1]

The Secret Weapon

Eric Raymond has written an essay called "How to Become a Hacker," and in it, among other things, he tells would-be hackers what languages they should learn. He suggests starting with Python and Java, because they are easy to learn. The serious hacker will also want to learn C, in order to hack Unix, and Perl for system administration and cgi scripts. Finally, the truly serious hacker should consider learning Lisp:

Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot.

This is the same argument you tend to hear for learning Latin. It won't get you a job, except perhaps as a classics professor, but it will improve your mind, and make you a better writer in languages you do want to use, like English.

But wait a minute. This metaphor doesn't stretch that far. The reason Latin won't get you a job is that no one speaks it. If you write in Latin, no one can understand you. But Lisp is a computer language, and computers speak whatever language you, the programmer, tell them to.

So if Lisp makes you a better programmer, like he says, why wouldn't you want to use it? If a painter were offered a brush that would make him a better painter, it seems to me that he would want to use it in all his paintings, wouldn't he? I'm not trying to make fun of Eric Raymond here. On the whole, his advice is good. What he says about Lisp is pretty much the conventional wisdom.

Languages exist along a
power continuum.

Looking down the axis, you see
languages missing key features!

How could anyone be
productive without that?!?

Looking up the continuum, you
see weird languages.

And features you don't use
(since they don't exist in Blub!)

We tend to get attached to a language and stick with it.

Be careful here.

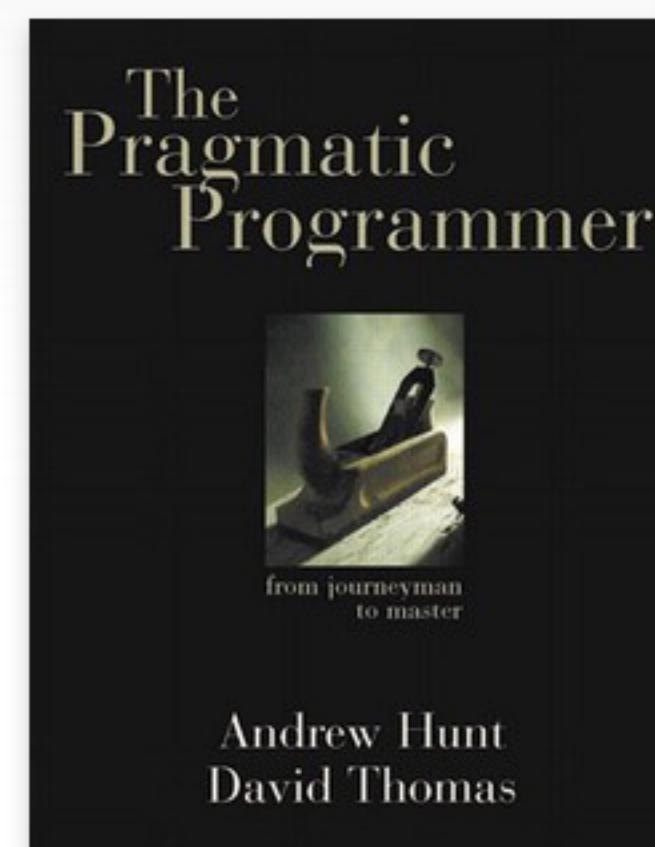


Important to look at
other languages.

Consider learning a new
language every year or two.

[SIGN IN](#)[TRY NOW >](#)

See everything available through O'Reilly online learning and start a free trial. Explore now.

[Search](#)

The Pragmatic Programmer: From Journeyman to Master

by

Released October 1999

Publisher(s): Addison-Wesley Professional

ISBN: 020161622X

Explore a preview version of *The Pragmatic Programmer: From Journeyman to Master* right now.

O'Reilly members get unlimited access to live online training experiences, plus books, videos, and digital content from 200+ publishers.

[START YOUR FREE TRIAL >](#)

Book description

What others in the trenches say about *The Pragmatic Programmer*...

"The cool thing about this book is that it's great for keeping the programming process fresh. The book helps you to continue to grow and clearly comes from people who have been there."

Even if you don't use them in
your day to day work.

Learning Ruby/Python/Haskell/Lisp/
Rust/Go will change how you code.

The more languages you know,
the easier it is to learn another.

You have more things to
compare it to.

Oh, that's like this in Ruby
and that in JavaScript.

Diversity makes us stronger.

How do you approach
a new code base?

Start with the team.

What would you say...
you do here?

Are there any docs?

Are there any diagrams?

Are there any tests?

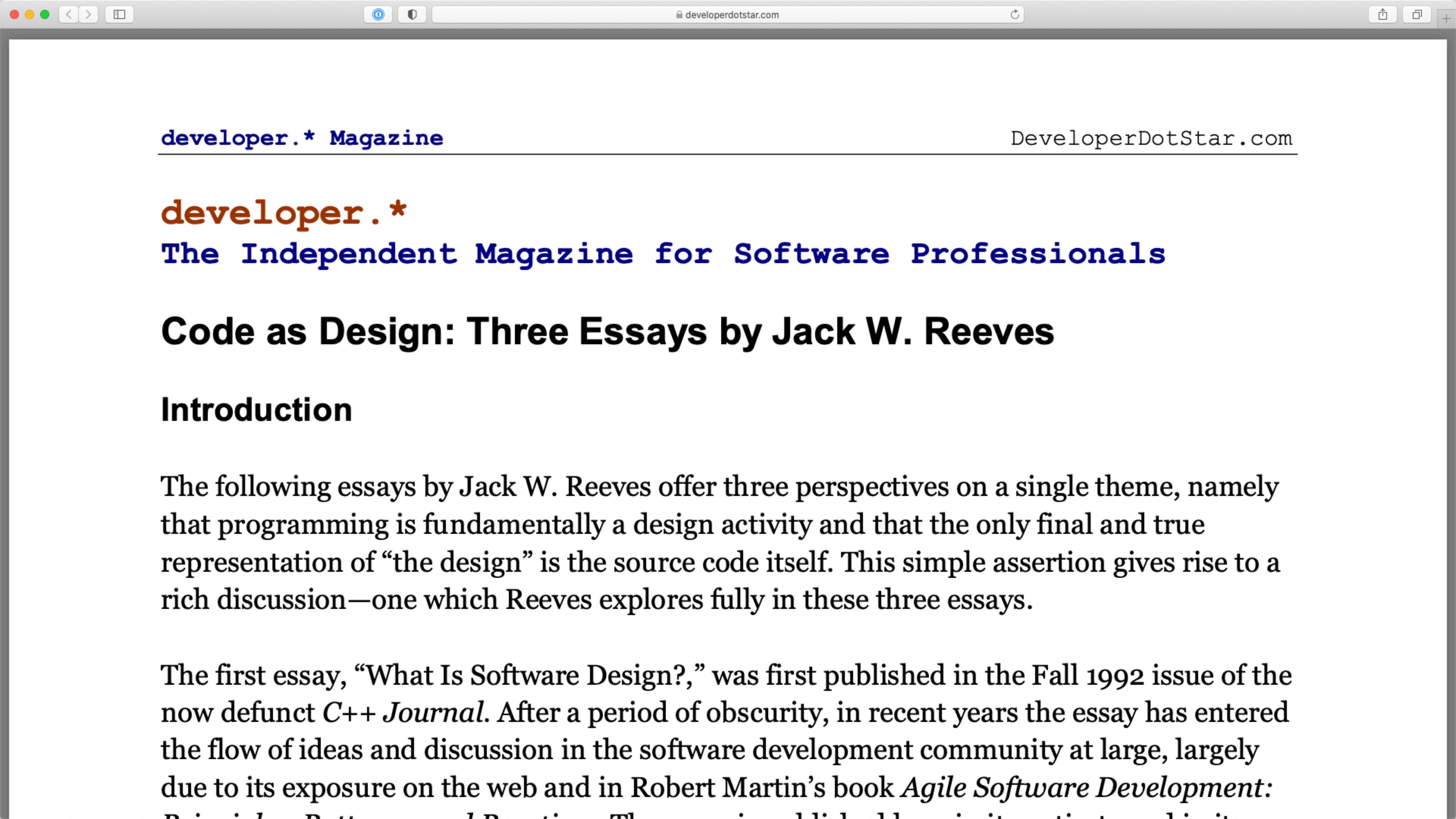
Documentation and diagrams.

Documentation often misleads...

“No matter what the documentation says,
the source code is the ultimate truth, the
best and most definitive and up-to-date
documentation you're likely to find.”

-Jeff Atwood

<https://blog.codinghorror.com/learn-to-read-the-source-luke/>



developer.*

The Independent Magazine for Software Professionals

Code as Design: Three Essays by Jack W. Reeves

Introduction

The following essays by Jack W. Reeves offer three perspectives on a single theme, namely that programming is fundamentally a design activity and that the only final and true representation of “the design” is the source code itself. This simple assertion gives rise to a rich discussion—one which Reeves explores fully in these three essays.

The first essay, “What Is Software Design?,” was first published in the Fall 1992 issue of the now defunct *C++ Journal*. After a period of obscurity, in recent years the essay has entered the flow of ideas and discussion in the software development community at large, largely due to its exposure on the web and in Robert Martin’s book *Agile Software Development: Principles, Patterns, and Practices*. The essay is available online at <http://www.developerdotstar.com/magazine/essays/design/>.

No documentation? Consider
creating some as you learn.

What should we document?

What does your service do?

How does it work?

What does it depend on?

Ever say something like “the documentation is useless”?



Nate Schutta
@ntschutta

"That's a write once read never artifact." @neal4d
I think Neal may have just described nearly every architectural document. #architecture #thisisfine

1:37 PM · Oct 9, 2020 · Tweetbot for iOS

View Tweet activity

3 Retweets 27 Likes

1 Reply 3 Retweets 27 Likes

<https://twitter.com/ntschutta/status/1314636196270739457>

It doesn't have to be.

Golden rule!

Do it for those that come after you.

Don't forget, sometimes *you* are
the person that comes after you!

How long does it take for a new team member to be productive? Weeks?

Months?

Solid onboarding guide.

Make sure it is updated.

Documentation should
be easy to find.

Probably a website/wiki.

Documentation is often out of date. It doesn't have to be.

Updating the wiki should be
part of the developer workflow.

Consider a simple (low ceremony) template.

Description - what does your service do? Don't skimp here.

An architectural diagram or three.

Contact information as well as
the on call rotation.

Links to helpful things like the repo,
dashboard link, on call book.

FAQ.

Onboarding/development guide.

Coding standards.

Development pipeline.

Whatever helps the
team understand.

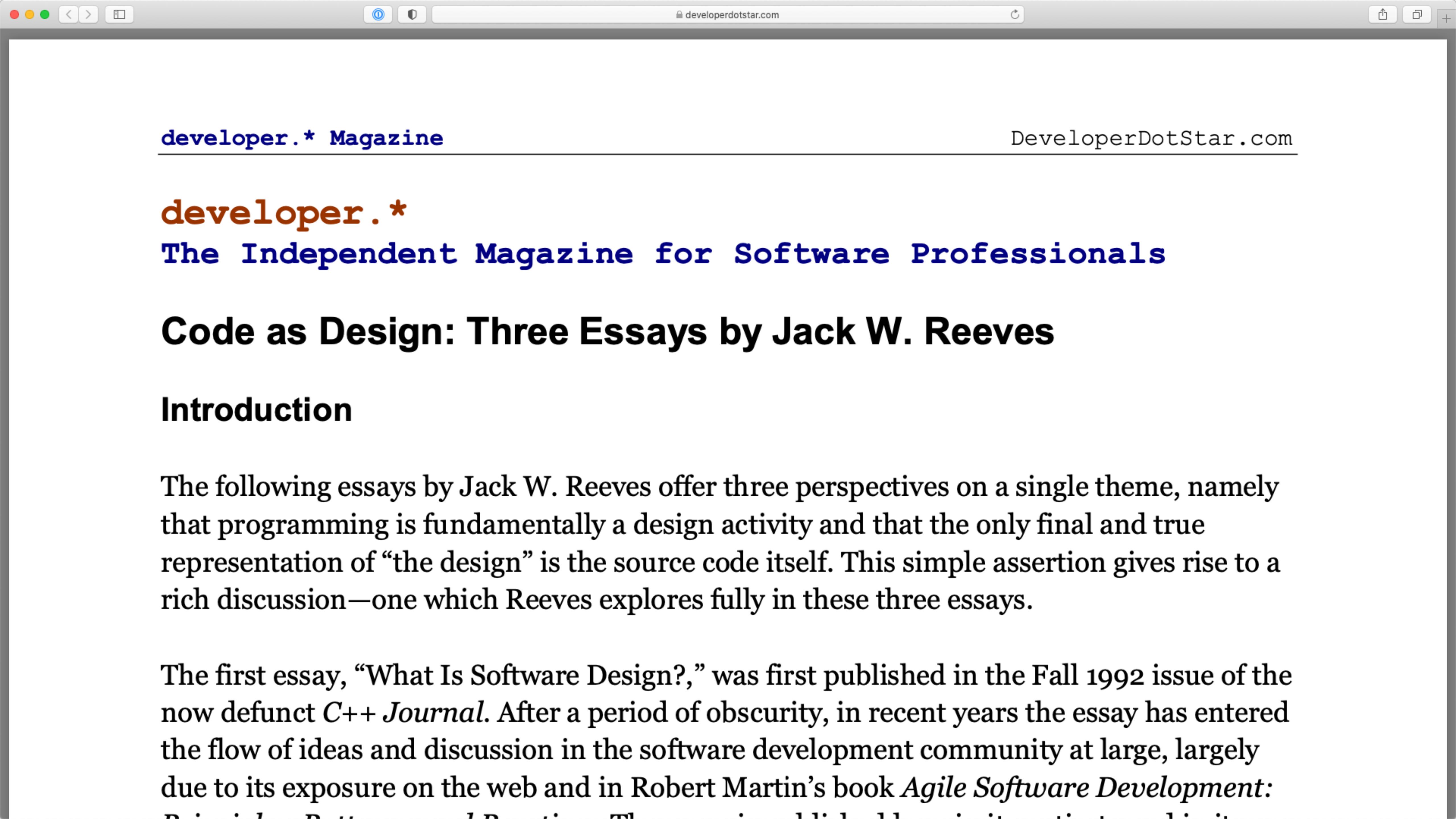
Everyone should “get it” and be able
to describe it. So have them do it.

Shouldn't be a static thing!

Documentation should be reviewed
along with the architecture.

Speaking of diagrams...

Diagrams don't compile.



developer.*

The Independent Magazine for Software Professionals

Code as Design: Three Essays by Jack W. Reeves

Introduction

The following essays by Jack W. Reeves offer three perspectives on a single theme, namely that programming is fundamentally a design activity and that the only final and true representation of “the design” is the source code itself. This simple assertion gives rise to a rich discussion—one which Reeves explores fully in these three essays.

The first essay, “What Is Software Design?,” was first published in the Fall 1992 issue of the now defunct *C++ Journal*. After a period of obscurity, in recent years the essay has entered the flow of ideas and discussion in the software development community at large, largely due to its exposure on the web and in Robert Martin’s book *Agile Software Development: Principles, Patterns, and Practices*. The essay is available online at <http://www.developerdotstar.com/magazine/essays/design/>.

Provides context.

Understand and
manage complexity.

Decompose the problem.

Predict quality attributes.

AKA theilities or non
functional requirements.

Design for certain quality attributes.

Some organizations require it!

Can be a blessing and a curse...

Can help during software
archaeology expeditions.

Assuming it's accurate...

Useful for education.

Assuming it's accurate...

Knowledge transfer.

Assuming it's accurate...

Should architectural diagrams
be kept up to date?

How permanent are they?

Diagrams generally have an
expiration date...

It's OK to throw a diagram away.

Whiteboards make for
excellent modeling tools.

Find yourself drawing it
again and again?

Might be time to formalize it.

Or just take a picture.

Speaking of tools...

The message matters.

Not the tool.

If you really like a tool
and it helps you, use it.

But don't let it hold you back.

Paper and pencil work.

Whiteboards work.

When in doubt, keep it simple.

What's better?

No diagram?

Or one that lies?

Communication!

Helps us tell the story.

If the team doesn't understand it...

They can't build it.

Don't forget a key!

Simon Brown @simonbrown

tl;dr, and my challenge to you for 2019: create a key/legend for the next software architecture you draw. If you find this activity hard ... that's feedback.

Simon Brown @simonbrown

At the end of the day, I don't really care which visual language you use to describe software architecture. Please make sure it's comprehensible, and helps rather than hinders your team. Happy new year! 🎉

Show this thread

3:05 AM · Jan 2, 2019 · Twitter Web Client

8 Retweets 20 Likes

Reply Retweet Like Share

<https://mobile.twitter.com/simonbrown/status/1080389551842095105>

 **Michael Nygard**
@mtnygard

Following

When diagramming, make sure your readers know a) what the elements are; b) what the relationships mean; and c) what the constraints are. That's the meta-model necessary to understand the model.

2:18 PM - 30 Dec 2018

46 Retweets 122 Likes



3 46 122

 **Michael Nygard** @mtnygard · 30 Dec 2018

Many diagrams I see don't label the arrows. Does an arrow mean data flow? Invocation? Dependency?

Are we looking at dynamic behavior or static structure?

2 2 17

 **Michael Nygard** @mtnygard · 30 Dec 2018

A legend on every diagram is the absolute minimum courtesy you should provide for your readers.

4 4 11

 **Michael Nygard** @mtnygard · 30 Dec 2018

We use diagrams to communicate. That requires both the transmitter and the receiver to understand the signal. You may have brilliant ideas but if you leave your reader lost without signposts, you've failed to communicate.

4 3 24

<https://twitter.com/mtnygard/status/1079471792291491840>

projects.spring.io

spring

DOCS GUIDES PROJECTS BLOG QUESTIONS

PROJECTS

Spring Petclinic

PetClinic demonstrates the use of a Spring Boot with Spring MVC and Spring Data. The PetClinic has an old and varied history dating right back to the beginning of the Spring Framework. It started life as a demonstration of nearly all the common things that you could do with Spring, back when it was possible to conceive of such a demonstration. These days it is a very small slice of what you could achieve, but the community has a soft spot for it, so it's nice to see it still going after all this time, so we hope you enjoy it too.

QUICK START

Fork me on GitHub

Quick Start

You need Java 1.8 and git:

```
$ git clone https://github.com/spring-projects/spring-petclinic.git  
$ cd spring-petclinic  
$ ./mvnw spring-boot:run
```

You can then access petclinic here: <http://localhost:8080/>

spring

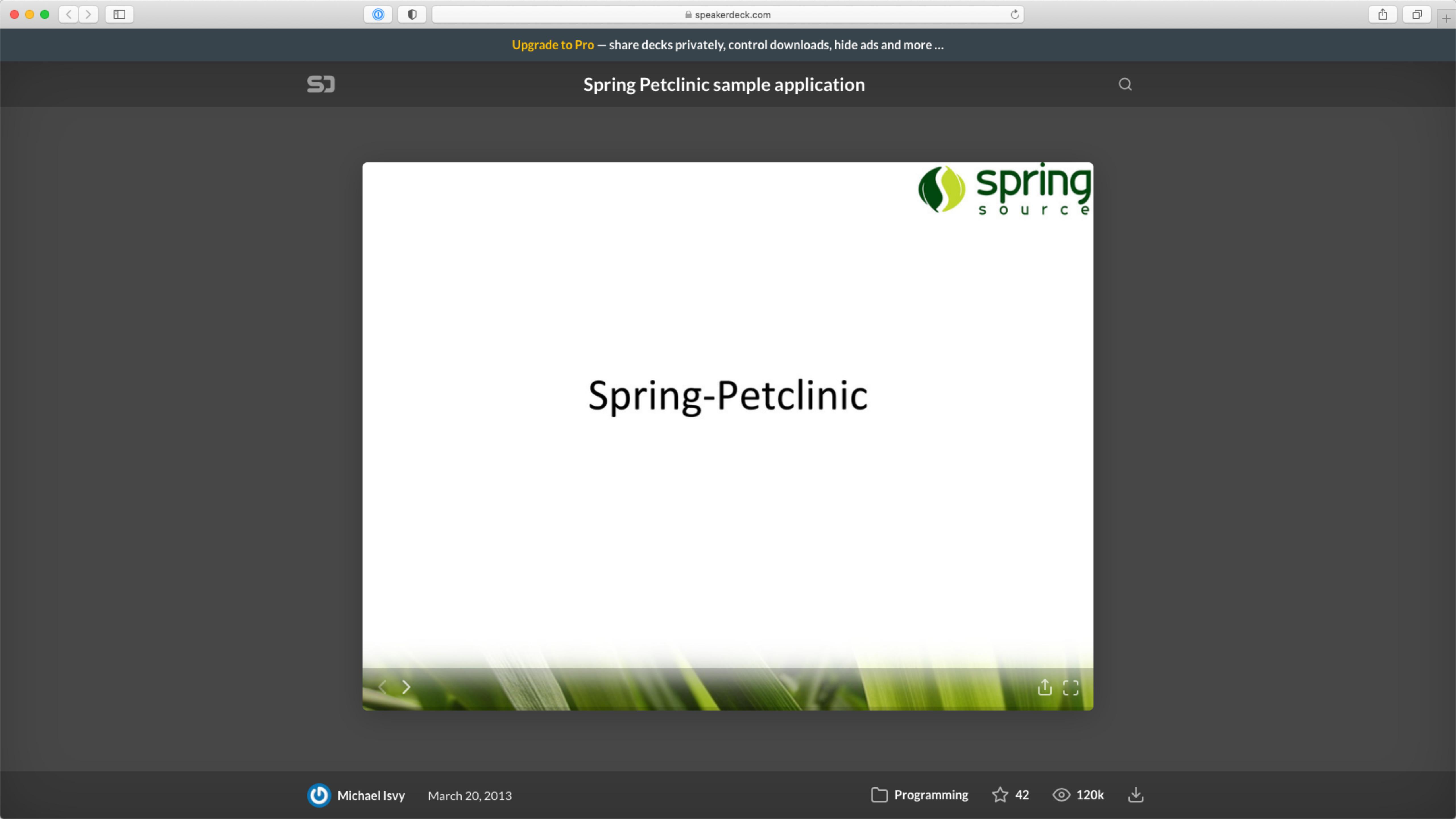
HOME FIND OWNERS VETERINARIANS ERROR

Owners

Name	Address	City	Telephone	Pets
Jeff Black	1450 Oak Blvd.	Monona	6085555387	Lucky
Jean Coleman	105 N. Lake St.	Monona	6085552654	Max Samantha
Betty Davis	638 Cardinal Ave.	Sun Prairie	6085551749	Basil

Spring Projects

Spring Boot
Spring Framework
Spring Data



Upgrade to Pro – share decks privately, control downloads, hide ads and more ...

SD

Spring Petclinic sample application

Q

spring source

Spring-Petclinic

< >

Programming 42 120k

Michael Isvy March 20, 2013

Read the code. Sorry.

Software archeology time!

Roll up your sleeves and root
around your codebase.

Look for (apologies Isaac Newton)
smoother pebbles and prettier shells.

Code structure.

How is the code organized?
It is organized...right?



A screenshot of a GitHub repository page for `spring-projects / spring-petclinic`. The page shows a list of files and their commit history under the `main` branch.

The repository navigation bar includes links for `Pull requests`, `Issues`, `Marketplace`, and `Explore`. The top right corner shows options to `Watch` (375), `Star` (5k), `Fork` (13k), and a user profile icon.

The main content area displays the file structure and commit history for the `spring-petclinic` directory:

- Commit:** `dsyer Apply spring-format plugin` (2 years ago)
- Files modified:** `model`, `owner`, `system`, `vet`, `visit`, `PetClinicApplication.java`
- Message:** Apply spring-format plugin
- Contributors:** 2 (dsyer, 4e1f874)
- Date:** Jan 3, 2020
- History:** (link)

The footer contains links to GitHub's Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About pages.

File	Message	Time Ago
<code>model</code>	Apply spring-format plugin	2 years ago
<code>owner</code>	Apply spring-format plugin	2 years ago
<code>system</code>	Apply spring-format plugin	2 years ago
<code>vet</code>	Apply spring-format plugin	2 years ago
<code>visit</code>	Apply spring-format plugin	2 years ago
<code>PetClinicApplication.java</code>	Apply spring-format plugin	2 years ago

Poke around, get a sense for
the code, how it fits together.

Read the tests!

```
41  @WebMvcTest(value = PetController.class,
42      includeFilters = @ComponentScan.Filter(value = PetTypeFormatter.class, type = FilterType.ASSIGNABLE_TYPE))
43  class PetControllerTests {
44
45      private static final int TEST_OWNER_ID = 1;
46
47      private static final int TEST_PET_ID = 1;
48
49      @Autowired
50      private MockMvc mockMvc;
51
52      @MockBean
53      private PetRepository pets;
54
55      @MockBean
56      private OwnerRepository owners;
57
58      @BeforeEach
59      void setup() {
60          PetType cat = new PetType();
61          cat.setId(3);
62          cat.setName("hamster");
63          given(this.pets.findPetTypes()).willReturn(Lists.newArrayList(cat));
64          given(this.owners.findById(TEST_OWNER_ID)).willReturn(new Owner());
65          given(this.pets.findById(TEST_PET_ID)).willReturn(new Pet());
66
67      }
68
69      @Test
70      void testInitCreationForm() throws Exception {
71          mockMvc.perform(get("/owners/{ownerId}/pets/new", TEST_OWNER_ID)).andExpect(status().isOk())
72              .andExpect(view().name("pets/createOrUpdatePetForm")).andExpect(model().attributeExists("pet"));
73      }
74
75      @Test
76      void testProcessCreationFormSuccess() throws Exception {
77          mockMvc.perform(post("/owners/{ownerId}/pets/new", TEST_OWNER_ID).param("name", "Betty")
78              .param("type", "hamster").param("birthDate", "2015-02-12")).andExpect(status().is3xxRedirection())
79              .andExpect(view().name("redirect:/owners/{ownerId}"));
80      }
81
82      @Test
83      void testProcessCreationFormHasErrors() throws Exception {
84          mockMvc.perform(post("/owners/{ownerId}/pets/new", TEST_OWNER_ID).param("name", "Betty").param("birthDate",
85              "2015-02-12")).andExpect(model().attributeHasNoErrors("owner"))
86              .andExpect(model().attributeHasErrors("pet")).andExpect(model().attributeHasFieldErrors("pet", "type"))
87              .andExpect(model().attributeHasFieldErrorCode("pet", "type", "required")).andExpect(status().isOk())
88              .andExpect(view().name("pets/createOrUpdatePetForm"));
89      }
90
91      @Test
92      void testInitUpdateForm() throws Exception {
```

Run the application.

What does it do?

Can you map that
back to the code?

Find a landmark. “I know the
code does X, let’s find that.”

“Mental models are built piece by piece.
Code should be read the same way.”

-Sunny Beattieay

<https://medium.com/launch-school/how-to-read-source-code-without-ripping-your-hair-out-e066472bbe8d>

 Kent Beck 
@KentBeck

The goal of software design is to create chunks or slices that fit into a human mind. The software keeps growing but the human mind maxes out, so we have to keep chunking and slicing differently if we want to keep making changes.

7:16 AM · Jan 27, 2021 · Twitter for iPhone

633 Retweets 44 Quote Tweets 2,191 Likes

 Tweet your reply 

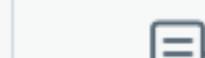
 Kent Beck  @KentBeck · Jan 27
Replies to @KentBeck
This implies that software design is a human process with technical support, done by humans for humans. As shiny as the technical support is it took me a while to acknowledge that messy people stuff.

 6  31  216 

 Kent Beck  @KentBeck · Jan 27
I'll add that "fits into" and "maxes out" are metaphors and not precisely true, but they help make sense of a healthy, growing urge to re-design. Knowledge is not a quantity & the brain is not a space & we work together.

 4  13  85 

 Kent Beck  @KentBeck · Jan 27
I am continuing work on "Tidy First? An Exercise In Empirical Software Design" as a [@SubstackInc](#). No paid tier yet, but that's where book chapters will be appearing. Sign up here: kentbeck.substack.com.

 Software Design: Tidy First?
Software design is an exercise in human

<https://twitter.com/KentBeck/status/1354418068869398538>

Work your way backwards.

Debuggers to the rescue.

Breakpoints for fun and profit!

Speaking of which...

Don't ~~assume~~ the
code is doing X.

Confirm it.

Take the time it takes
so it takes less time.

“We don't rise to the level of
our expectations, we fall to the
level of our training.”

- Archilochus

Exceptions can mislead.

Catching Exception and
assuming it had to be...

Trust...but verify.

Not all code is organized...

Or uses meaningful variable names.

Use your editor.

Use your SCM.

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Advanced search Search

Advanced options

From these owners

In these repositories

Created on the dates >YYYY-MM-DD, YYYY-MM-DD

Written in this language

Repositories options

With this many stars

With this many forks

Of this size

Pushed to <YYYY-MM-DD

With this license

Return repositories including forks.

Code options

With this extension

Of this file size

In this path

With this file name

Rinse and repeat!

That's cool but we do
microservices.

We use Kotlin.

We use Angular or React.

Cool, cool, cool.

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Spring Petclinic community
Open Source community hosting Spring Petclinic forks
🔗 https://spring-petclinic.github.io

Repositories 15 Packages People 1 Projects

Pinned repositories

- [spring-petclinic-microservices](#)
Distributed version of Spring Petclinic built with Spring Cloud
Java 929 1.1k
- [spring-framework-petclinic](#)
A Spring Framework application based on JSP, Spring MVC, Spring Data JPA, Hibernate and JDBC
Java 274 1.3k
- [spring-petclinic-reactjs](#)
ReactJS (with TypeScript) and Spring Boot version of the Spring Petclinic sample application
Less 181 187

- [spring-petclinic-angular](#)
Angular 8 version of the Spring Petclinic sample application (frontend)
TypeScript 153 346
- [spring-petclinic-kotlin](#)
Kotlin version of Spring Petclinic
Kotlin 239 100
- [spring-petclinic-data-jdbc](#)
Spring Data JDBC version of the Spring Boot Petclinic
Java 59 65

Find a repository... Type Language Sort

spring-petclinic-cloud
Fork of the Spring Cloud Microservices project packaged to be deployed on several Cloud platforms: Kubernetes and Cloud Foundry

docker kubernetes spring-cloud cloud-foundry wavefront tanzu

Java Apache-2.0 99 63 3 1 Updated 11 days ago

spring-petclinic-graphql
PetClinic Example based on GraphQL

react graphql spring spring-boot

Top languages

- Java
- JavaScript
- Kotlin
- TypeScript
- CSS

Most used topics

- spring-boot
- spring
- sample
- spring-cloud
- spring-data-jpa

People 1 >

How do you improve your
code reading skills?

High quantity of high
quality examples.

Seek out and read well written code.

OSS strikes again!

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Watch 695 Star 9k Fork 1.3k

Code Pull requests Actions Projects Security Insights

master 22 branches 159 tags Go to file Add file Code

 puredanger refresh contributing and add pull request template a29f9b9 on Jun 24 3,348 commits

File/Folder	Description	Time Ago
.github	refresh contributing and add pull request template	last month
.idea/libraries	fix for leak caused by ddc65a9	9 years ago
doc/clojure/pprint	fix typos in docstrings and docs	8 years ago
src	CLJ-2603: Added support for trailing, conj-able element in map-dest...	5 months ago
test	CLJ-2603: Added support for trailing, conj-able element in map-dest...	5 months ago
.gitignore	gitignore IntelliJ stuff	7 years ago
CONTRIBUTING.md	refresh contributing and add pull request template	last month
antsetup.sh	add test.generative as a test dependency	9 years ago
build.xml	first cut of datafy	3 years ago
changes.md	changelog for 1.10.3	6 months ago
clojure.iml	elide unused constants	6 years ago
epl-v10.html	Moved to Eclipse Public License - see epl-v10.html or	13 years ago
pom.xml	[maven-release-plugin] prepare for next development iteration	5 months ago
readme.txt	update asm license	3 years ago

readme.txt

```
* Clojure
* Copyright (c) Rich Hickey. All rights reserved.
* The use and distribution terms for this software are covered by the
* Eclipse Public License 1.0 (http://opensource.org/licenses/eclipse-1.0.php)
* which can be found in the file epl-v10.html at the root of this distribution.
* By using this software in any fashion, you are agreeing to be bound by
* the terms of this license.
```

About

The Clojure programming language

[clojure.org](#)

Readme

Releases

159 tags

Packages

No packages published

Used by 3.5k

+ 3,445

Contributors 152

+ 141 contributors

Environments 1

github-pages Active

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Watch 3.5k Star 43.8k Fork 30.7k

Code Issues 1k Pull requests 256 Actions Projects Wiki Security Insights

main 15 branches 213 tags Go to file Add file Code

sbrannen Polish contribution ... ce94f69 3 days ago 22,716 commits

- .github Update link for reporting security issues 6 months ago
- buildSrc Introduce Gradle Toolchain support in build 5 months ago
- ci Make script executable 3 months ago
- framework-bom Remove BOM workaround 15 months ago
- gradle Reintroduce left-hand side navigation in ref docs 17 days ago
- integration-tests Drop explicit zeroing at instantiation of Atomic* objects 10 months ago
- spring-aop Polish contribution 3 days ago
- spring-aspects Polishing 9 months ago
- spring-beans Support ResolvableType in BeanDefinitionBuilder 22 days ago
- spring-context-indexer Polish Javadoc in spring-context-indexer 2 months ago
- spring-context-support Use StringBuilder.append(char) where possible last month
- spring-context Improve @Cacheable documentation regarding java.util.Optional 7 days ago
- spring-core Update copyright date 5 days ago
- spring-expression Update copyright year of changed file last month
- spring-instrument Delete obsolete log4j config 17 months ago
- spring-jcl Polishing 2 years ago
- spring-jdbc DataClassRowMapper suppresses setter method calls for constructor... 25 days ago
- spring-jms ObjectMapper.configure(MapperFeature, boolean) is deprecated as o... 11 days ago
- spring-messaging ObjectMapper.configure(MapperFeature, boolean) is deprecated as o... 11 days ago
- spring-orm Polish tests 4 months ago

About Spring Framework spring.io/projects/spring-framework framework spring spring-framework

Readme Apache-2.0 License

Releases 213 v5.2.16.RELEASE Latest 20 days ago + 212 releases

Packages No packages published

Contributors 548 + 537 contributors

Environments 1 github-pages Active

github.com

facebook / react

Code Issues Pull requests Actions Projects Wiki Security Insights

main 109 branches 138 tags Go to file Add file Code

mrkev Switch on enableProfilerChangedHookIndices flag for OSS devtool... 3a1dc3e 3 hours ago 14,351 commits

.circleci [DRAFT] Import scheduling profiler into DevTools Profiler (#21897) 12 days ago

.codesandbox Clean up Scheduler forks (#20915) 3 months ago

.github Updated scripts and config to replace "master" with "main" branch (#...) last month

fixtures Some remaining instances of master to main (#21982) 4 days ago

packages Switch on enableProfilerChangedHookIndices flag for OSS devtools b... 3 hours ago

scripts Some remaining instances of master to main (#21982) 4 days ago

.editorconfig https link to editorconfig.org (#18421) 16 months ago

.eslintignore DevTools: Update named hooks match to use column number also (#...) 26 days ago

.eslintrc.js remove obsolete SharedArrayBuffer ESLint config (#21259) 4 months ago

.gitattributes .gitattributes to ensure LF line endings when we should 8 years ago

.gitignore Build stable and experimental with same command (#20573) 7 months ago

.mailmap updates mailmap entries (#19824) 11 months ago

.nvmrc fix: version in nvmrc (#21430) 3 months ago

.prettierignore DevTools: Update named hooks match to use column number also (#...) 26 days ago

.prettierrc.js Bump Prettier (#17811) 2 years ago

.watchmanconfig .watchmanconfig must be valid json (#16118) 2 years ago

AUTHORS Remove my deadname from AUTHORS (#21152) 4 months ago

CHANGELOG.md Updated scripts and config to replace "master" with "main" branch (#...) last month

CODE_OF_CONDUCT.md Update code of conduct (#21251) 4 months ago

CONTRIBUTING.md [Gatsby] "https://facebook.github.io/react/" -> "https://reactjs.org/" (4 years ago

About

A declarative, efficient, and flexible JavaScript library for building user interfaces.

reactjs.org

react javascript library ui
frontend declarative

Readme

MIT License

Releases 138

17.0.2 (March 22, 2021) Latest on Mar 22 + 137 releases

Packages

No packages published

Used by 7.1m

+ 7,093,147

Contributors 1,493

Star 172k Fork 34.6k

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Angular

https://angular.io Verified

Repositories 198 Packages People 49 Projects

Pinned repositories

- angular**
The modern web developer's platform
TypeScript 75.4k 19.7k
- components**
Component infrastructure and Material Design components for Angular
TypeScript 21.6k 5.8k
- angular-cli**
CLI tool for Angular
TypeScript 24.7k 11.3k
- angular.js**
AngularJS - HTML enhanced for web apps!
JavaScript 59.6k 28.5k
- material**
Material design for AngularJS
JavaScript 16.7k 3.6k
- protractor**
E2E test framework for Angular apps
JavaScript 8.8k 2.4k

Find a repository... Type Language Sort

ngcc-validation
Angular Ivy library compatibility validation project
angular ivy ngcc
TypeScript 41 107 1 0 Updated 3 minutes ago

angular
The modern web developer's platform
javascript angular typescript web-performance web pwa
web-framework
TypeScript MIT 19,691 75,436 1,746 (30 issues need help) 175 Updated 38 minutes ago

Top languages

- JavaScript
- TypeScript
- HTML
- Dart
- Java

Most used topics

- angular javascript
- angularjs-material material-design
- typescript

People 49 >

github.com

Search or jump to... Pull requests Issues Marketplace Explore

LLVM

This is the LLVM organization on GitHub for the LLVM Project: a collection of modular and reusable compiler and toolchain technologies.

https://llvm.org board@llvm.org Verified

Repositories 22 Packages People 178 Projects

Pinned repositories

llvm-project

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Note: the repository does not accept github pull requests at this moment. Please submit your patches at...

9.8k 3.8k

Find a repository... Type Language Sort

circt

Circuit IR Compilers and Tools

llvm mlir circt

C++ 75 635 124 (7 issues need help) 21 Updated 2 minutes ago

llvm-iwg

The LLVM Infrastructure Working Group

5 3 28 (1 issue needs help) 5 Updated 7 minutes ago

llvm-project

The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Note: the repository does not accept github pull requests at this moment. Please submit your patches at <http://reviews.llvm.org>

Top languages

C++ HTML SCSS JavaScript Python

People 178 >

github.com

Search or jump to... Pull requests Issues Marketplace Explore

Explore Topics Trending Collections Events GitHub Sponsors Get email updates

Trending

See what the GitHub community is most excited about today.

	Repositories	Developers	Spoken Language: Any	Language: Any	Date range: Today
1	mitmproxy / mitmproxy	An interactive TLS-capable intercepting HTTP proxy for penetration testers and software developers.	Python	★ 23,729 ⚡ 2,972 Built by	☆ 239 stars today
2	Hiroshima / voicevox	TypeScript	★ 323 ⚡ 23 Built by	☆ 75 stars today	
3	sysprog21 / lkmpg	The Linux Kernel Module Programming Guide (updated for 5.x kernels)	TeX	★ 718 ⚡ 60 Built by	☆ 538 stars today
4	dataease / dataease	人人可用的开源数据可视化分析工具。	Java	★ 1,615 ⚡ 210 Built by	☆ 144 stars today
5	facebookresearch / droidlet	A modular embodied agent architecture and platform for building embodied agents	Jupyter Notebook	★ 427 ⚡ 26 Built by	☆ 45 stars today
6	commaai / openpilot	openpilot is an open source driver assistance system. openpilot performs the functions of Automated Lane Centering and Adaptive Cruise Control for over 100 supported car makes and models.	C++	★ 26,404 ⚡ 5,295 Built by	☆ 199 stars today

What do you like about the
code? What don't you like?

What would you change to
make it more readable?

What would you have
done differently? Why?

The more code you've read, the
easier it is...to read code.

No different than reading prose.

ALL CAPS IS 10% SLOWER!

Why?

We scan for shapes.

ALL CAPS?

Just a bunch of rectangles...

Writing readable content (and why All Caps is so hard to read)

By Marty Friedel

Published September 9th, 2015

Content

[View all Blog articles](#)

There's nothing worse than browsing the web and being hit with a huge slab of text in All Caps – that is, all in CAPITAL LETTERS.

the water out of
the bucket it's

Gaining expertise takes time.

There is a giant bucket of things
we don't know we know.

Hidden, essential knowledge.

Experts build that up over years.

How do they pass that
knowledge along?

Especially if they can't verbalize
what they do or how they do it?

They just know?!?

Apprentice model.

Apprenticeships date
back to the Middle Ages.

Still common today in certain fields.

There aren't any shortcuts.

Sorry.

It does get easier over time!

And you will get faster.

Virtuous cycle!

You don't have to take
my word for it.

READING CODE IS HARDER THAN WRITING IT

We should invest more time in the skill of reading code.

Abstract

It's funny that computer languages are the only languages where one learns to write before learning to read. It's actually not uncommon for people to never really learn to read code. This seems a little unbalanced given that we actually read code much more frequently than we write it.

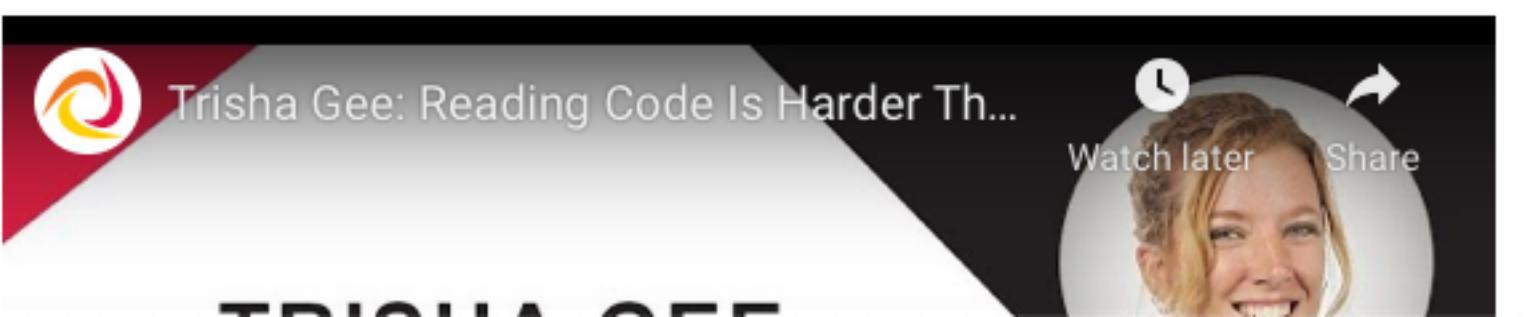
Even those who promote software as a craft sometimes fall into the trap of often talking about writing clean code that people can read, yet not placing much emphasis on the skill of reading the code.

The ability to read code must also be a skill, and as such it must be something that can be learnt and practiced. In this presentation, we're going to look at:

- *The different reasons we might have to read code, and how that should impact our reading*
- *The problems we face when reading code (even our own!)*
- *Tips to bear in mind when we're reading code*
- *Tools we can use to help our understanding*
- *How and where to practice these skills*

At the end of the talk we will at least have considered whether we need to level up our "Reading Code" skill.

Videos



We use cookies on our website to give you the most relevant experience by remembering your preferences and repeat visits. By clicking "Accept", you consent to the use of ALL the cookies.

[Cookie settings](#)

[ACCEPT](#)

youtube.com

YouTube

Search

All Computer programming Courses

Scott Hanselman - Beyond Mentorship - Storytelling and... NDC Conferences 92 views • 31 minutes ago New

lofi hip hop radio - beats to relax/study to Lofi Girl 32K watching LIVE NOW

Mix - NDC Conferences YouTube

Refactoring to Immutability - Kevlin Henney NDC Conferences 59K views • 3 years ago

The Simplest Math Problem No One Can Solve Veritasium 5.8M views • 4 days ago New

Why You Will Marry the Wrong Person @THE SCHOOL OF LIFE 3.8M views • 3 years ago

How Do I Learn to Read Code? Why Should I Learn to Read... IAmTimCorey 9.4K views • 11 months ago

CppCon 2018: Simon Brand "How C++ Debuggers Work" CppCon 16K views • 2 years ago

Inside the mind of a master procrastinator | Tim Urban TED 41M views • 5 years ago

NDC { Sydney }

17-21 September 2018

Inspiring Software Developers since 2008

ndcsydney.com

Reading other peoples code - Patricia Aas

2,650 views • Oct 19, 2018

Like 57 Dislike 4 Share Save

NDC Conferences 114K subscribers

Subscribe

@PATI_GALLARDO



- So... You Got Someone Else's Code?
- Before You Start
- 10 Techniques
- Excellence

1:37 / 55:43

CC

SHARE

SAVE

Software Breakpoints

How C++ Debuggers Work

Simon Brand

1:01:00

TED

14:04

1:06:14

1:03:22

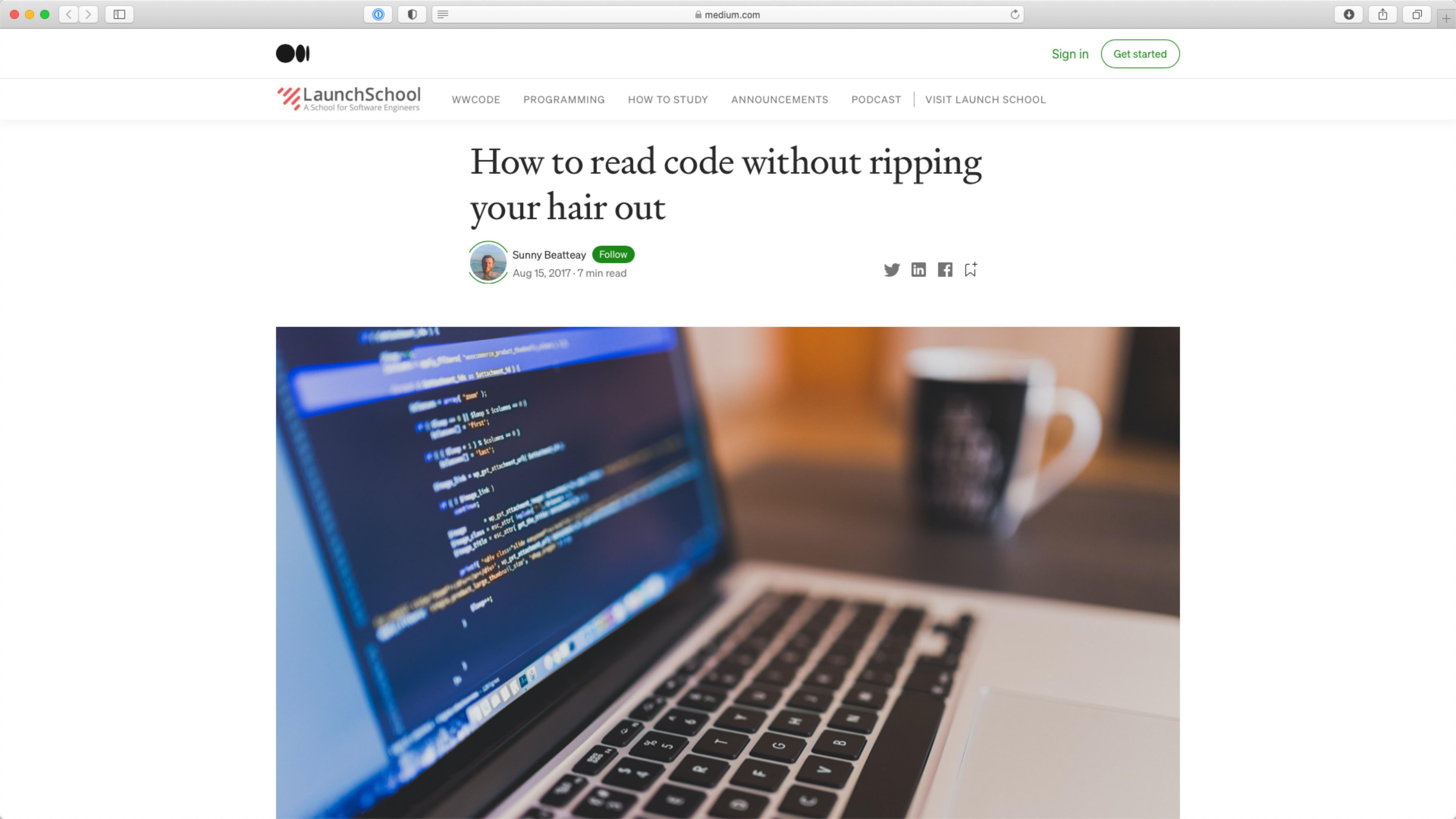
22:09

22:20

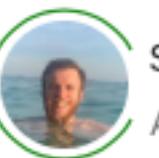
17:40

1:01:00

14:04

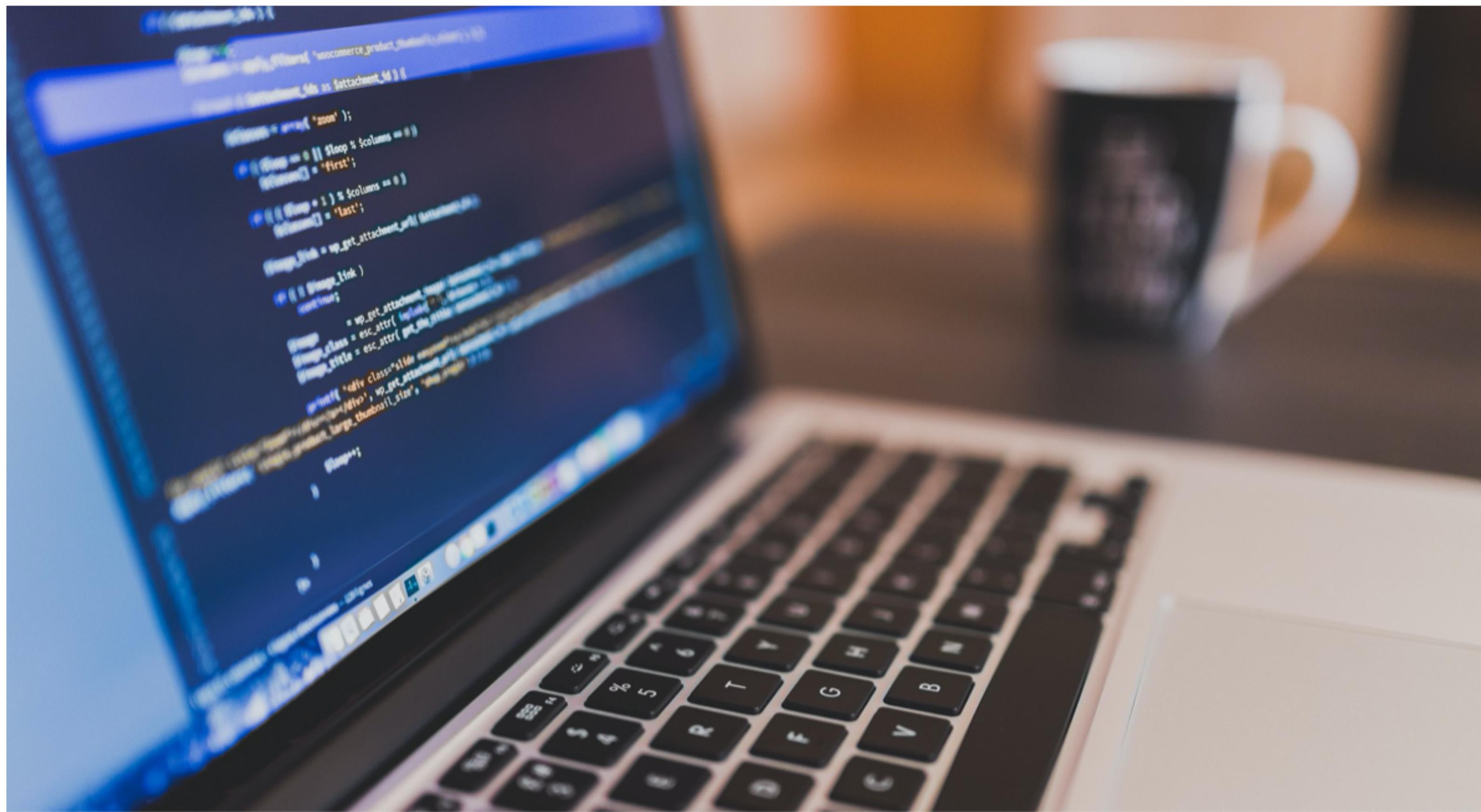


How to read code without ripping your hair out



Sunny Beatteay [Follow](#)
Aug 15, 2017 · 7 min read

[Twitter](#) [LinkedIn](#) [Facebook](#) [Link](#)



blog.codinghorror.com

CODING HORROR

programming and human factors

ENHANCED BY Google

16 Apr 2012

Learn to Read the Source, Luke

In the calculus of communication, writing coherent paragraphs that your fellow human beings can comprehend and understand is *far more difficult* than tapping out a few lines of software code that the interpreter or compiler won't barf on.

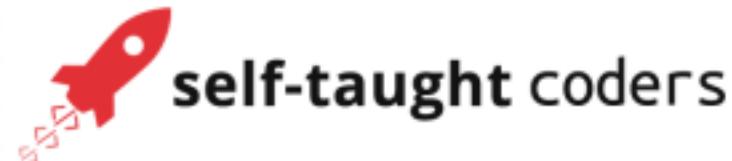
That's why, when it comes to code, *all the documentation probably sucks*. And because writing for people is way harder than writing for machines, the documentation will *continue to suck* for the foreseeable future. There's very little you can do about it.

Except for one thing.



You can **learn to read the source, Luke**.

The [transformative power of "source always included" in JavaScript](#) is a major reason why I coined – and continue to believe in – [Atwood's Law](#). Even if "view source" isn't built in (but it totally should be), you should demand access to the underlying source code for



How to quickly and effectively read other people's code

by Alex Coleman | Learning, Web Development

Just the other day, a fellow STCer (Self-Taught Coder) asked me the following question:

"How do you go about understanding someone else's code? I am starting to feel more comfortable with my own code, but whenever I try to look at something someone else wrote I feel totally lost. I know some of this is unavoidable, especially if the code is poorly (or not at all) documented, but right now I have no strategy at all. Any tips would be greatly appreciated!"

I love this question for a few reasons:

1. The method I'll recommend for reading and understanding someone else's code will also help you: 1) better understand *your own* code; and 2) help you increase the speed and ease with which you understand *all* new pieces of code you approach.
2. It sheds light on one of the most important aspects of learning a new skill, like programming: exposure to high quantity, high quality examples of expertise.

There are a lot of wins here. Let's start at the beginning.

What's the best way to read and understand someone else's code?

The best way I've ever discovered to read and understand someone else's code is to:

1. Find one thing you know the code does, and trace those actions





WRITING CODE.

Obviously we write code.

Maybe too much!

Has someone else
solved this already?

Is there a library you can leverage?

Could it be a language feature?

If you think there should be
an easier way, look for it.

What is good code?



I know it when I see it.

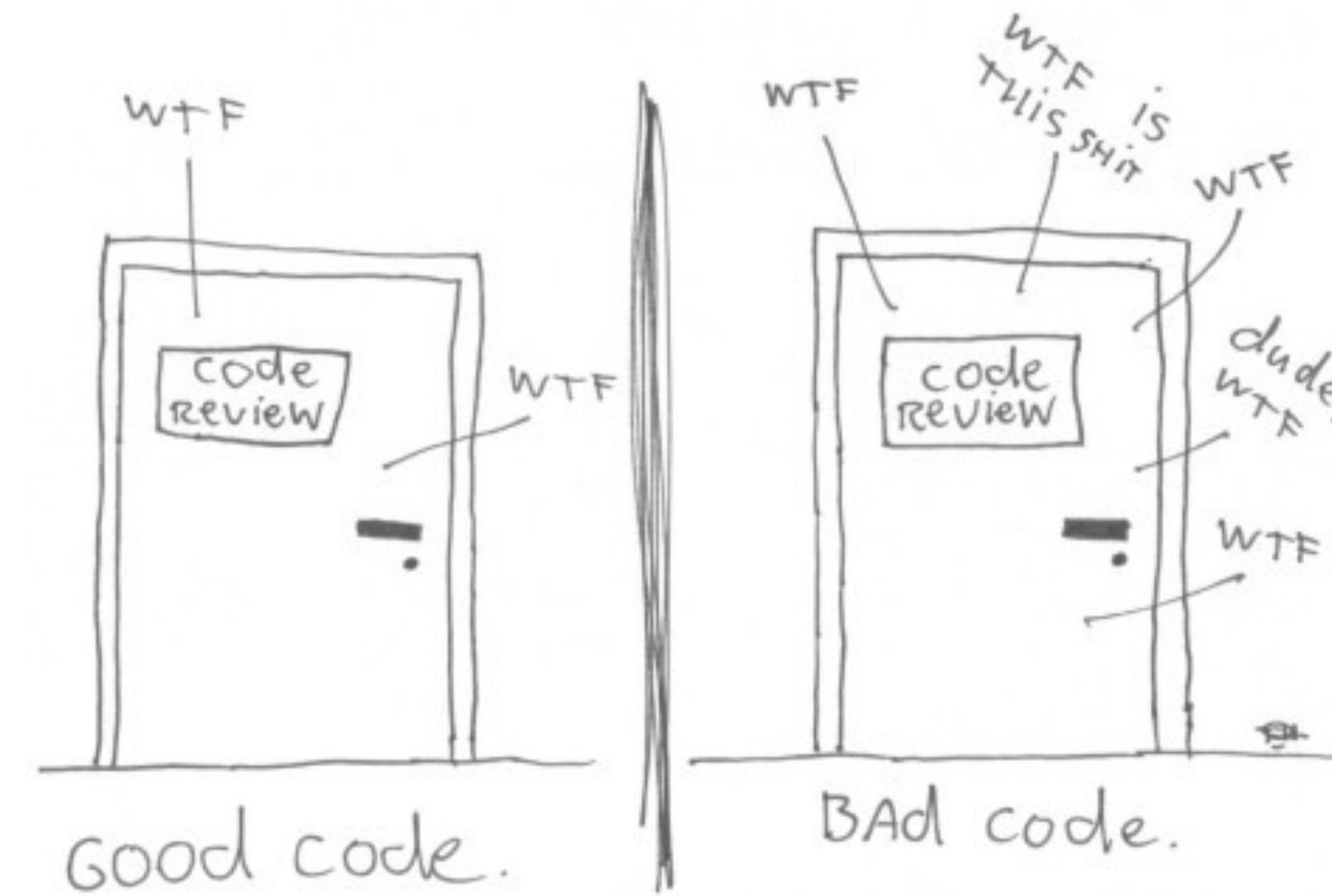
Very subjective thing!

WTFs/m

👤 Thom Holwerda 📅 2008-02-04 🏷️ Comics 💬 25 Comments

Comic: "WTFs/m".

The ONLY VALID MEASUREMENT
OF CODE QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

<https://www.osnews.com/story/19266/wtfsm/>

We all know bad code right?

Sniffable.

CodeSmell

9 February 2006



Martin Fowler

- ◆ TECHNICAL DEBT
- ◆ PROGRAMMING STYLE
- ◆ REFACTORING

A code smell is a surface indication that usually corresponds to a deeper problem in the system. The term was first coined by Kent Beck while helping me with my [Refactoring book](#).

The quick definition above contains a couple of subtle points. Firstly a smell is by definition something that's quick to spot - or *sniffable* as I've recently put it. A long method is a good example of this - just looking at the code and my nose twitches if I see more than a dozen lines of java.

The second is that smells don't always indicate a problem. Some long methods are just fine. You have to look deeper to see if there is an underlying problem there - smells aren't inherently bad on their own - they are often an indicator of a problem rather than the problem themselves.

The best smells are something that's easy to spot and most of time lead you to really interesting problems. Data classes (classes with all data and no behavior) are good examples of this. You look at them and ask yourself what behavior should be in this class. Then you start refactoring to move that behavior in there. Often simple questions and initial refactorings can be the vital step in turning anemic objects into something that really has class.

One of the nice things about smells is that it's easy for inexperienced people to spot them, even if they don't know enough to evaluate if there's a real problem or to correct them. I've heard of lead developers who will pick a "smell of the week" and ask people to look for the smell and bring it up with the senior members of the team. Doing it one smell at a time is a good way of gradually teaching people on the team to be better programmers.

Aren't always problematic.

Requires investigation.

Smell of the week?

We don't write bad code...

Always that other person...

“Hello, my name is Nate
and I write s#@!&y code.”

Metrics can help.

Cyclomatic complexity.

Source code analyzers.

SonarQube, PMD, JDepend,
CodeScene, CodeRush, Checkstyle.

The list goes on and on!

Change Risk Analysis and Predictions.

Cyclomatic complexity
vs. code coverage.

Simple code? Lower coverage
is...less problematic.

Complicated code? Better have
high code coverage!

With a reasonable ceiling on
complexity mind you.

And it isn't 83.

Hawthorne effect.

People modify their behavior
because they're observed.

That can be used to our advantage.

Want more code coverage?
Prominently display coverage stats.

Be careful with metrics.

An Appropriate Use of Metrics

Management love their metrics. The thinking goes something like this, "We need a number to measure how we're doing. Numbers focus people and help us measure success." Whilst well intentioned, management by numbers unintuitively leads to problematic behavior and ultimately detracts from broader project and organizational goals. Metrics inherently aren't a bad thing; just often, inappropriately used. This essay demonstrates many of the issues caused by management's traditional use of metrics and offers an alternative to address these dysfunctions.

19 February 2013



Patrick Kua

Patrick Kua leads development teams drawing upon agile methods to deliver valuable software for customers. He is author of [The Retrospective Handbook: A guide for agile teams](#)

CONTENTS

- [What's wrong with how we use metrics?](#)
- [Be careful what you measure](#)
- [Guidelines for a more appropriate use of metrics](#)
 - [Explicitly link metrics to goals](#)
 - [Favor tracking trends over absolute numbers](#)
 - [Use shorter tracking periods](#)
 - [Change metrics when they stop driving change](#)
- [Conclusion](#)

SIDEBARS

- [Metrics as a ratchet](#)

 METRICS

 PRODUCTIVITY

 PROJECT PLANNING

 TECHNICAL LEADERSHIP

Link metrics to goals.

Focus on trends.

Favor short time horizons.

Adapt and adjust!

Less is more.

Small code bases are your friend.

Microservices and functions!

Monoliths often have dictionary
sized getting started guides.

Build times measured in
phases of the moon.

It can take months for a new developer to get up to speed.

What was your longest stretch to
get to productive team member?

Smaller scope === less to get
your head wrapped around.

The 0th Law of Computer Science:

High cohesion, low coupling...

Boilerplate is evil.

Even if it is generated.

Short classes - few pages.

What do **you** mean by short?

Definitely language dependent.

Some languages are more
verbose than other.

Do you have to scroll?
Might be too long.

Short methods.

Think single digit lines of code.

Do one thing, do it well.

Linux like - pipe simple things
together to get complex results.

Say what you do. Don't be clever.

Naming is hard.

“Object Calisthenics”

One level of indent.

No else statements.



<https://twitter.com/mfeathers/status/292509590757376>

One dot per line.

No abbreviations.

Constraints shall set you free?

Descriptive names.

Simplify. Then simplify some more.



Download MP3

01:01:26

Summary

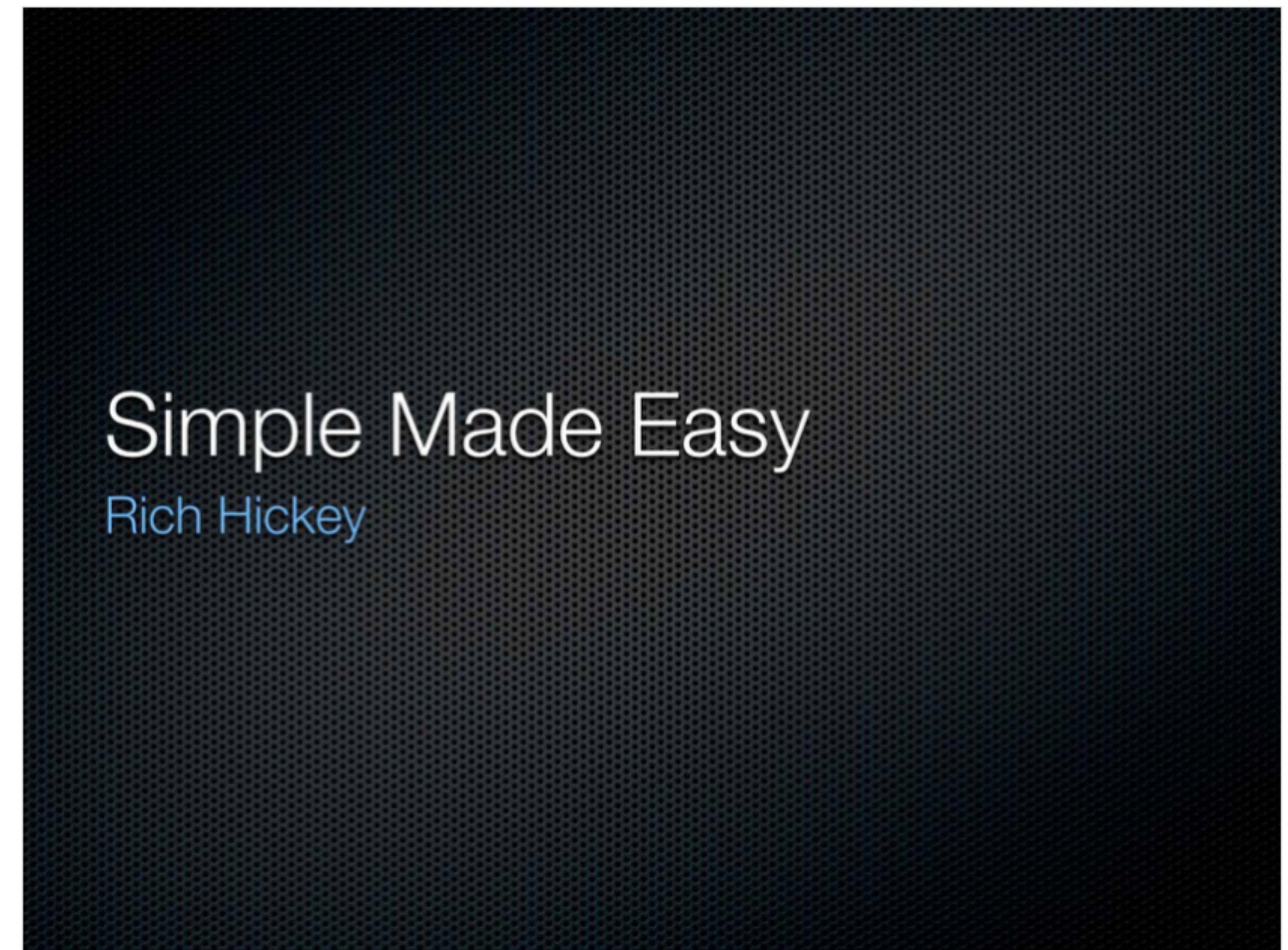
Rich Hickey emphasizes simplicity's virtues over easiness', showing that while many choose easiness they may end up with complexity, and the better way is to choose easiness along the simplicity path.

About the conference

Strange Loop is a multi-disciplinary conference that aims to bring together the developers and thinkers building tomorrow's technology in fields such as emerging languages, alternative databases, concurrency, distributed systems, mobile development, and the web.

Bio

Rich Hickey, the author of Clojure, is an independent software designer, consultant and application architect with over 20 years of experience in all facets of software



^ Key Takeaways

- We should aim for simplicity because simplicity is a prerequisite for reliability.
- Simple is often erroneously mistaken for easy. "Easy" means "to be at hand", "to be approachable". "Simple" is the opposite of "complex" which means "being intertwined", "being tied together". Simple != easy.
- What matters in software is: does the software do what is supposed to do? Is it of high quality? Can we rely on it? Can problems be fixed along the way? Can requirements change over time? The answers to these questions is what matters in writing software not the look and feel of the experience writing the code or the cultural implications of it.
- The benefits of simplicity: ease of understanding, ease of change, ease of

Recorded at:



OCT 20, 2011

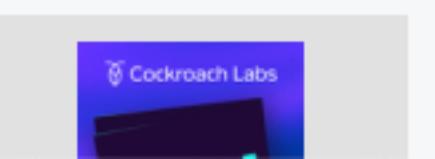
by



Rich Hickey

FOLLOW

RELATED SPONSORED CONTENT



Remove duplication,
even small amounts.

But if it is terse...
won't it be confusing?



Sunday, February 10, 2008

Portrait of a Noob

The older I grow, the less important the comma becomes. Let the reader catch his own breath.

— Elizabeth Clarkson Zwart

This is how I used to comment my code, twenty years ago (Note: dramatization):

```
/**  
 * By the time we get to this point in the function,  
 * our structure is set up properly and we've created  
 * a buffer large enough to handle the input plus some  
 * overflow space. I'm not sure if the overflow space  
 * is strictly necessary, but it can't hurt. Next we  
 * have to update the counter to account for the fact  
 * that the caller has read a value without consuming  
 * it. I considered putting the counter-increment on  
 * the shoulders of the caller, but since it meant every  
 * caller had to do it, I figured it made more sense to  
 * just move it here. We can revisit the decision down  
 * the road if we find some callers that need the option  
 * of incrementing it themselves.  
 */  
counter++; // increment the consumed-value counter
```

```
/**  
 * Now we've got to start traversing the buffer, but we  
 * need an extra index to do it; otherwise we'll wind up  
 * at the end of the function without having any idea  
 * what the initial value was. I considered calling this  
 * variable 'ref', since in some sense we're going to be  
 * treating it as a reference, but eventually I decided  
 * it makes more sense to use 'pos'; I'm definitely open  
 * to discussion on it, though.  
 */  
char* pos = buffer; // start our traversal
```

About Me



STEVE YEGGE
KIRKLAND,
WASHINGTON, UNITED
STATES

[VIEW MY COMPLETE PROFILE](#)

Previous Posts

- [Emergency Elisp](#)
- [Blogging Theory 201: Size Does Matter](#)
- [Code's Worst Enemy](#)
- [Boring Stevey Status Update](#)
- [Ten Tips for a \(Slightly\) Less Awful Resume](#)
- [Stevey's Tech News, Issue #1](#)
- [How To Make a Funny Talk Title Without Using The W...](#)
- [Rhino on Rails](#)
- [Rich Programmer Food](#)
- [That Old Marshmallow Maze Spell](#)



Favor composition
over inheritance.

 **Jez Humble** 
@jezhumble

I teach a graduate level class on OO and TDD and every year I get upset because I have undergrads in my class who have been taught that encapsulation means adding getters and setters to your classes 

 **Jason Gorman (only, more indoors than usual)** @jasongorman · Aug 4
I still get comments 10 years later on my Tell, Don't Ask video objecting to the idea that the object with the data should do the work

12:01 PM · Aug 4, 2021 · Twitter for iPhone

61 Retweets 13 Quote Tweets 266 Likes

 Tweet your reply 

 **Jez Humble** 
@jezhumble · Aug 4
Replies to [@jezhumble](#)

Like, setters are obviously bad because you want your objects to be immutable unless there is a really good reason for them not to be (there almost never is). Use constructors.

But it takes the whole semester for me to maybe help people realize what to do instead of getters.

 6  6  37 

 **Jez Humble** 
@jezhumble · Aug 4
The other thing that gets me mad is the overuse of inheritance. People use inheritance as a code reuse mechanism all the time and produces terrible, horrible, no-good code that is completely unmaintainable and full of defects and hard to test. I am begging you to stop it.

 9  22  84 

<https://twitter.com/jezhumble/status/1422965825858772993>

Reuse is a byproduct,
not a rationale!

<https://news.ycombinator.com/item?id=1620244>

What is the ideal
length of a function?

FunctionLength

30 November 2016



Martin Fowler

METRICS

PROGRAMMING STYLE

During my career, I've heard many arguments about how long a function should be. This is a proxy for the more important question - when should we enclose code in its own function? Some of these guidelines were based on length, such as functions should be no larger than fit on a screen [1]. Some were based on reuse - any code used more than once should be put in its own function, but code only used once should be left inline. The argument that makes most sense to me, however, is the **separation between intention and implementation**. If you have to spend effort into looking at a fragment of code to figure out what it's doing, then you should extract it into a function and name the function after that "what". That way when you read it again, the purpose of the function leaps right out at you, and most of the time you won't need to care about how the function fulfills its purpose - which is the body of the function.

Once I accepted this principle, I developed a habit of writing very small functions - typically only a few lines long [2]. Any function more than half-a-dozen lines of code starts to smell to me, and it's not unusual for me to have functions that are a single line of code [3]. The fact that size isn't important was brought home to me by an example that Kent Beck showed me from the original Smalltalk system. Smalltalk in those days ran on black-and-white systems. If you wanted to highlight some text or graphics, you would reverse the video. Smalltalk's graphics class had a method for this called 'highlight', whose implementation was just a call to the method 'reverse' [4]. The name of the method was longer than its implementation - but that didn't matter because there was a big distance between the intention of the code and its implementation.

Some people are concerned about short functions because they are worried about

Function names indicate the what.

Shorten the distance from
intent to implementation.

Perfectly fine to have
a one line function!

Code can read like a
newspaper article.

Inverted pyramid.

The lead, key facts, background.

Reader decides how in depth
they want to go.

Reader can stay high level or
dive into the details.

Code comments.

Should we comment our code?

Are they a code smell?

The screenshot shows a web browser window with the following details:

- Address Bar:** Not Secure — nealford.com
- Page Title:** Neal Ford | Author, Thoughtworker, & Meme Wrangler ...
- Header:** A large logo "NF" and "nealford.com" is on the left. To its right are navigation links: "Abstracts/Slides", "Biography", "Books", "Video", and "Writing".
- Section Header:** "Comments == Code Smell" is displayed in bold black text.
- Text Content:** The page discusses the author's opinions on code comments, mentioning JavaDoc-style comments, in-line comments, and the DRY principle. It also touches on the evolution of programming languages and the concept of composed methods.
- Footers:** Navigation links at the bottom include "← Previous", "Archive", "Next →", and a "Search" icon.

Comments == Code Smell

I am sometimes asked about my position on code comments, and, like most things, I have strong opinions about it. Two kinds of comments exist:

- JavaDoc-style comments (which encompasses JavaDoc, XMLDoc, RDoc, etc), which are designed to produce developer documentation at a high level (class and method names and what they do)
- In-line comments, generally scattered around the code to indicate a note from developer to developer

Both kinds of comments represent different smells, each with different odors depending on the target.

What makes comments so smelly in general? In some ways, they represent a violation of the DRY (Don't Repeat Yourself) principle, espoused by the [Pragmatic Programmers](#). You've written the code, now you have to write about the code. In a perfect world, you'd never have to write comments for this purpose: the code will be expressive enough that someone who reads it will understand it. Two things help achieve this result: expressive, readable languages and the composed method pattern.

The language makes a big difference as to the readability of the code. If you write in assembly language, you're pretty much forced to write comments; no one on earth can read the code directly. As languages have matured, you can get much closer to the ideal of self-documenting code (which was explicitly attempted in [Literate Programming](#)). Especially in the modern wave of non-ceremonious languages (like Ruby, Groovy, Scala, etc), you can craft extremely readable code. To this end, ThoughtWorks projects generally avoid the more magical features of languages (like the implicit global variables in Ruby, for example) because it hurts readability. One of the side effects of dynamic typing is outstanding method names. The method name is the only vector of information about what the method does (you can't crutch on return or parameter types), so methods tend to be named much better on the dynamic language projects upon which I've worked.

The second key to readable code is not to have too much of it, especially at the method level. In [Smalltalk Best Practice Patterns](#), Kent Beck defines composed method (which I write about extensively in [The Productive Programmer](#)). Composed method encapsulates the idea that all your methods should do one and only one thing, making them as small as possible. The side effect of this discipline leads to public methods that mainly consist of calls to a large number of very small private methods, each of which do only one thing. Composed method has lots of beneficial side effects on your code: small methods are easier to test, you end up with really low cyclomatic complexity for your methods, you discover and harvest reusable code chunks more easily, and your code is readable. This last one brings us back to the topic of comments. If you use composed method, you'll find much less need to have comments to delineate sections of code within methods (actual method calls do that), and you'll find that you use better method names.

Now let's talk about how this applies to the two types of comments. First, where are comments indeed useful (and less smelly)? If you are writing an API, you need some level of generated documentation so that people can use your API. JavaDoc style comments do this job well because they are generated from code and have a fighting chance of staying in sync with the actual code. However, tests make much better documentation than comments. Comments always lie (maybe not now, but on a long enough timeline, all comments will become outdated). Tests can't lie or they fail. When I'm looking at work in progress on projects, I always go to the tests first. The comments may or may not be there, but the tests define what's now done and working.

We were on a project where the client insisted on Javadoc comments for every public class and method. We started the project adding those comments, but eventually stopped. When doing agile development, you don't want anything that hampers refactoring. Having comments in place caused a dilemma: do I refactor the method and change the comment (the most work), refactor the method and leave the old comment (with the theory being that I'll change it again later, and would rather just have to update the comment once), or not refactor? No good options here. Having pervasive comments discourages refactoring because it adds significant extra friction. On this project, we abandoned commenting as we went along. The last week of the project we literally did nothing but go back and add comments to the code, which worked well because the code base had settled down by that point. But notice what's lurking in wait: the client wanted all the comments there to make it easier to maintain in the future. But who's to say that whoever maintains that code will keep the comments up to date? You have the same DRY violation as before. If they maintain the tests, they have to change as code changes because they are executable.

The new wave of Behavior Driven Development tools (like [JBehave](#), [RSpec](#), and [easyb](#) make this "executable specification" style of comment + test feasible. I expect to see the usage of these tools skyrocket because they give you documentation that has a fighting chance of staying up to date.

Inline comments are almost always a smell. The only legitimate use of inline comments is when you have some very complex algorithm that you need to have some thoughts about beside the code. Otherwise, the presence of inline comments indicates that you've written code that needs explanation, meaning that it cries out for refactoring. I frequently troll code bases upon which I'm working to look for inline comments so that I can refactor the code to eliminate the need for them.

Comments are a great example of something that seems like a Good Thing, but turn out to cause more harm than good. Fortunately, we've figured out how to achieve the same benefits that comments allegedly provide with tests, particularly BDD-style tests.

Violates the DRY Principle.

Don't Repeat Yourself.

You wrote the code...then
wrote about the code.

Code should be readable.

Self documenting code?

Expressive languages help!

Smaller surface area FTW.

May want to avoid “magic”
aspects of languages...

What happens when we
change the code?

Do we update the comments?





<https://mobile.twitter.com/themarcba/status/1367366516044427269>

We're all familiar with the second
law of thermodynamics...

Otherwise known as a
teenagers bedroom.

The universe really
wants to be disordered.

Default to the lower energy state.

The worst comments?
Code change blocks...

Date, work order, author,
change description.

If only we had source code
management tools...

Oh wait.

Some comments are...
less than helpful.

```
//print out integers 0-99
for (int i=0; i<100; i++)
{
    System.out.print(i);
    System.out.print(' ');
}
```

Explaining something
that isn't obvious.

Thought try and simplify the
code if at all possible!

Thoughts or commentary on a
very complex algorithm.

In general, if the code needs to
be explained, rewrite it.

Reminder to our future selves...

We tend to be very expansive in
what we mean by future.



<https://twitter.com/mfeathers/status/9290086320>



<https://twitter.com/gurlcode/status/1367867319138086929>



<https://twitter.com/github/status/1367885997527171073>

Sometimes a hack works,
we don't (yet) know why...

//Magic. Do not touch.

Warnings to future developers.

```
// Dear maintainer:  
//  
// Once you are done trying to  
// 'optimize' this routine,  
// and have realized what a terrible  
// mistake that was,  
// please increment the following  
// counter as a warning  
// to the next person:  
//  
// total_hours_wasted_here = 42
```

#truth

//When I wrote this, only
//God and I understood
//what I was doing. Now,
//God only knows.

-Karl Weierstrass

Need a laugh?

stackoverflow.com

stackoverflow

About Products For Teams Search... Log in Sign up

Home PUBLIC Questions Tags Users COLLECTIVES Explore Collectives FIND A JOB Jobs Companies TEAMS Stack Overflow for Teams – Collaborate and share knowledge with a private group. Create a free Team What is Teams?

What is the best comment in source code you have ever encountered? [closed]

Asked 12 years, 10 months ago Active 4 years, 4 months ago Viewed 3.0m times

360 votes 6274 Closed 10 years ago.

Q As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Locked. This question and its answers are [locked](#) because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

What is the best comment in source code you have ever encountered?

comments Share edited Sep 18 '11 at 1:54 community wiki 14 revs, 11 users 61% Robert Harvey

Comments disabled on deleted / locked posts / reviews

518 Answers Active Oldest Votes

1 2 3 4 5 ... 18 Next

1462 votes I am particularly guilty of this, embedding non-constructive comments, code poetry and little jokes into most of my projects (although I usually have enough sense to remove anything directly offensive before releasing the code). Here's one I'm particularly fond of, placed far, far down a poorly-designed 'God Object':

```
/**  
 * For the brave souls who get this far: You are the chosen ones,  
 * the valiant knights of programming who toil away, without rest,  
 * fixing our most awful code. To you, true saviors, kings of men,  
 * I say this: never gonna give you up, never gonna let you down,  
 * never gonna run around and desert you. Never gonna make you cry,  
 * never gonna say goodbye. Never gonna tell a lie and hurt you.
```

The Overflow Blog
Communities are a catalyst for technology development
Podcast 363: Highlights from our 2021 Developer Survey

Featured on Meta
Join me in Welcoming Valued Associates: #945 - Slate - and #948 - Vanny

Check out these companies
Sandia National Laboratories Follow Exceptional service in the national interest mpi openmp cuda 5 open jobs · 1 follower
Transporeon GmbH Follow Bringing transportation in sync with the world maven spring hibernate 2 open jobs · 16 followers
Wunderkind Follow Wunderkind is a performance marketing channel that powers one-to-one messages at unprecedented scale. golang java python 6 followers
cie Cie Follow Cie is an innovation accelerator for large

Join Stack Overflow to learn, share knowledge, and build your career. Sign up with email Sign up with Google Sign up with GitHub Sign up with Facebook

If comments are considered
harmful, what should we do?

Write tests!

Especially more fluent styles.

Behavior Driven Development.

DAN NORTH & ASSOCIATES LTD

Blog Talks Books Case Studies Courses Articles Categories

Introducing BDD

I had a problem. While using and teaching agile practices like test-driven development (TDD) on projects in different environments, I kept coming across the same confusion and misunderstandings. Programmers wanted to know where to start, what to test and what not to test, how much to test in one go, what to call their tests, and how to understand why a test fails.

The deeper I got into TDD, the more I felt that my own journey had been less of a wax-on, wax-off process of gradual mastery than a series of blind alleys. I remember thinking "If only someone had told me that!" far more often than I thought "Wow, a door has opened." I decided it must be possible to present TDD in a way that gets straight to the good stuff and avoids all the pitfalls.

My response is behaviour-driven development (BDD). It has evolved out of established agile practices and is designed to make them more accessible and effective for teams new to agile software delivery. Over time, BDD has grown to encompass the wider picture of agile analysis and automated acceptance testing.

Test method names should be sentences

My first "Aha!" moment occurred as I was being shown a deceptively simple utility called [agiledox](#), written by my colleague, Chris Stevenson. It takes a JUnit test class and prints out the method names as plain sentences, so a test case that looks like this:

```
public class CustomerLookupTest extends TestCase {  
    testFindsCustomerById() {  
        ...  
    }  
  
    testFailsForDuplicateCustomers() {  
        ...  
    }  
}
```

SPEAKING

RECENT POSTS

[We need to talk about testing](#)

[CUPID - the back story](#)

[The mystery of the missing date](#)

[Monte Python Simulation: misunderstanding](#)

[Monte Carlo](#)

[In praise of SWARMing](#)

SOCIAL

 Twitter

 LinkedIn

Tests act as the documentation.

Executable specifications!

Stays up to date with
the code changes.

Comments...not so much.

But what about those
consuming my services?

They need the generated
comment docs!

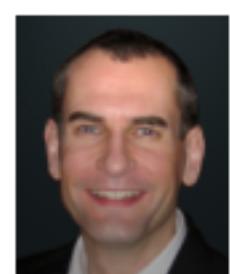
Definitely less smelly than most
comments, but...

Tests make for better
documentation.

Consumer-Driven Contracts: A Service Evolution Pattern

This article discusses some of the challenges in evolving a community of service providers and consumers. It describes some of the coupling issues that arise when service providers change parts of their contract, particularly document schemas, and identifies two well-understood strategies - adding schema extension points and performing "just enough" validation of received messages - for mitigating such issues. Both strategies help protect consumers from changes to a provider contract, but neither of them gives the provider any insight into the ways it is being used and the obligations it must maintain as it evolves. Drawing on the assertion-based language of one of these mitigation strategies - the "just enough" validation strategy - the article then describes the "Consumer-Driven Contract" pattern, which imbues providers with insight into their consumer obligations, and focuses service evolution around the delivery of the key business functionality demanded by consumers.

12 June 2006



CONTENTS

[Evolving a Service: An Example](#)

[Interlude: Burdened With Services](#)

[Schema Versioning](#)

[Extension Points](#)

[Breaking Changes](#)

[Schematron](#)

[Consumer-Driven Contracts](#)

[Provider Contracts](#)

[Consumer Contracts](#)

[Consumer-Driven Contracts](#)

[Summary of Contract Characteristics](#)

[Implementation](#)

[Benefits](#)

[Liabilities](#)

Ian Robinson

Ian Robinson is a Principal Consultant with Thoughtworks. He specialises in helping clients create sustainable service-oriented development capabilities that align business and IT from inception through to operation. He has written guidance for Microsoft on implementing service-oriented systems with Microsoft technologies, and has published articles on business-oriented development methodologies and distributed systems design - most recently in The

spring.io

Why Spring ▾ Learn ▾ Projects ▾ Training Support Community ▾

Spring Cloud Contract 3.0.0

OVERVIEW LEARN SAMPLES

Spring Cloud Contract is an umbrella project holding solutions that help users in successfully implementing the Consumer Driven Contracts approach. Currently Spring Cloud Contract consists of the Spring Cloud Contract Verifier project.

Spring Cloud Contract Verifier is a tool that enables Consumer Driven Contract (CDC) development of JVM-based applications. It is shipped with Contract Definition Language (DSL) written in Groovy or YAML. Contract definitions are used to produce following resources:

- by default JSON stub definitions to be used by WireMock (HTTP Server Stub) when doing integration testing on the client code (client tests). Test code must still be written by hand, test data is produced by Spring Cloud Contract Verifier.
- Messaging routes if you're using one. We're integrating with Spring Integration, Spring Cloud Stream and Apache Camel. You can however set your own integrations if you want to.
- Acceptance tests (by default in JUnit or Spock) used to verify if server-side implementation of the API is compliant with the contract (server tests). Full test is generated by Spring Cloud Contract Verifier.

Spring Cloud Contract Verifier moves TDD to the level of software architecture.

To see how Spring Cloud Contract supports other languages just check out [this blog post](#).

Features

When trying to test an application that communicates with other services then we could do one of two things:

- deploy all microservices and perform end to end tests
- mock other microservices in unit / integration tests

Both have their advantages but also a lot of disadvantages. Let's focus on the latter. Deploy all

Spring Boot

Spring Framework

Spring Data >

Spring Cloud ▾

- Spring Cloud Azure
- Spring Cloud Alibaba
- Spring Cloud for Amazon Web Services
- Spring Cloud Bus
- Spring Cloud CLI
- Spring Cloud for Cloud Foundry
- Spring Cloud - Cloud Foundry Service Broker
- Spring Cloud Cluster
- Spring Cloud Commons
- Spring Cloud Config
- Spring Cloud Connectors
- Spring Cloud Consul
- Spring Cloud Contract
- Spring Cloud Function
- Spring Cloud Gateway
- Spring Cloud GCP
- Spring Cloud Netflix
- Spring Cloud Open Service Broker
- Spring Cloud Pipelines
- Spring Cloud Schema Registry
- Spring Cloud Security
- Spring Cloud Skipper

Again, stays up to date
with code changes.

And assures us we didn't
break a consumer.

Accidental complexity is the
bane of our existence!

Avoid clever code.

//When I wrote this, only
//God and I understood
//what I was doing. Now,
//God only knows.

-Karl Weierstrass

You will forget.

Don't take shortcuts.

```
if (condition)
    doFoo();
    doBar();
```

Cost me two weeks.

Oops.

Technically, the brackets are
optional. Technically.

One of many error prone forms.

New rule: always, always,
always put brackets around ifs.

Always.

Every language has those types
of things. Avoid them.

What are your coding standards?

Expect some...discussion.
And that is fair.

People want and need to be heard.

May have to practice
some influence skills.

How do we get the
decision makers to buy in?

What techniques can we
use to influence them?

Outline the benefits.

Find common ground.

Avoid aggression.

Listen.

Have a conversation!

Can be hard to convince people!

Two approaches...





Find the influencers.

Influence the influencers.

“A reform often advances
most rapidly by indirection.”

– Ms. Frances Willard

Approach as equals.

Rely on the strength of your
ideas and your reputation.

Your reputation speaks for you
when you aren't there.

Not sure what your rep is?

Ask.

May not like the answer...

But you can work to change it.

Find common ground.

Reciprocity rules...

Be helpful.

Be respectful.

Research your ideas.

Use trusted sources.

Recruit credible allies.

Nothing wrong with bringing help!

We have to sell our decisions.



**ALWAYS
BE
CLOSING**

What hill do you want to die on?

Tabs vs. spaces?

Be ruthlessly pragmatic.

Apply linters and other scans to
ensure they don't sneak in...

All code is legacy code.

And code inevitably lives
longer than we expect it to.



<https://mobile.twitter.com/themarcba/status/1367366516044427269>



 Grady Booch ✅
@Grady_Booch

Old software never dies; you have to kill it.



Google will kill off very old versions of Android next month
Google's tighter login security means Android 2.3.7 and lower will lose functionality.
arstechnica.com

10:14 PM · Aug 3, 2021 · Twitter Web App

23 Retweets 3 Quote Tweets 116 Likes

Reply Share Like Retweet

https://twitter.com/Grady_Booch/status/142275825806176257

More or less as soon
as it is committed!

“Let the past die. Kill
it, if you have to.”

- Ben Solo

It's always tempting
to nuke and pave.

Don't just dismiss existing code.

Have an app that's delivered
business value for ten years?

Congratulations!

Pat yourselves on the back.

Legacy tends to be derogatory.

Heritage might be better.

Can be an amazing
learning opportunity!



<https://twitter.com/daliashea/status/1374343674876854273>

Code reviews.

One of the best ways to learn.

And socialize experience.

We learn from each other.

More eyes on the code is a win!

Consider pair programming!
Code review while coding...

Can be formal or informal.

Could be as simple as asking a
colleague to look over something.

Could be on a regular cadence.

Give people time to review,
block out a couple of hours.

Don't be snarky!

It can be hard to criticize.

It can be harder to be criticized.

Remember - it isn't personal.

Well, it shouldn't be.

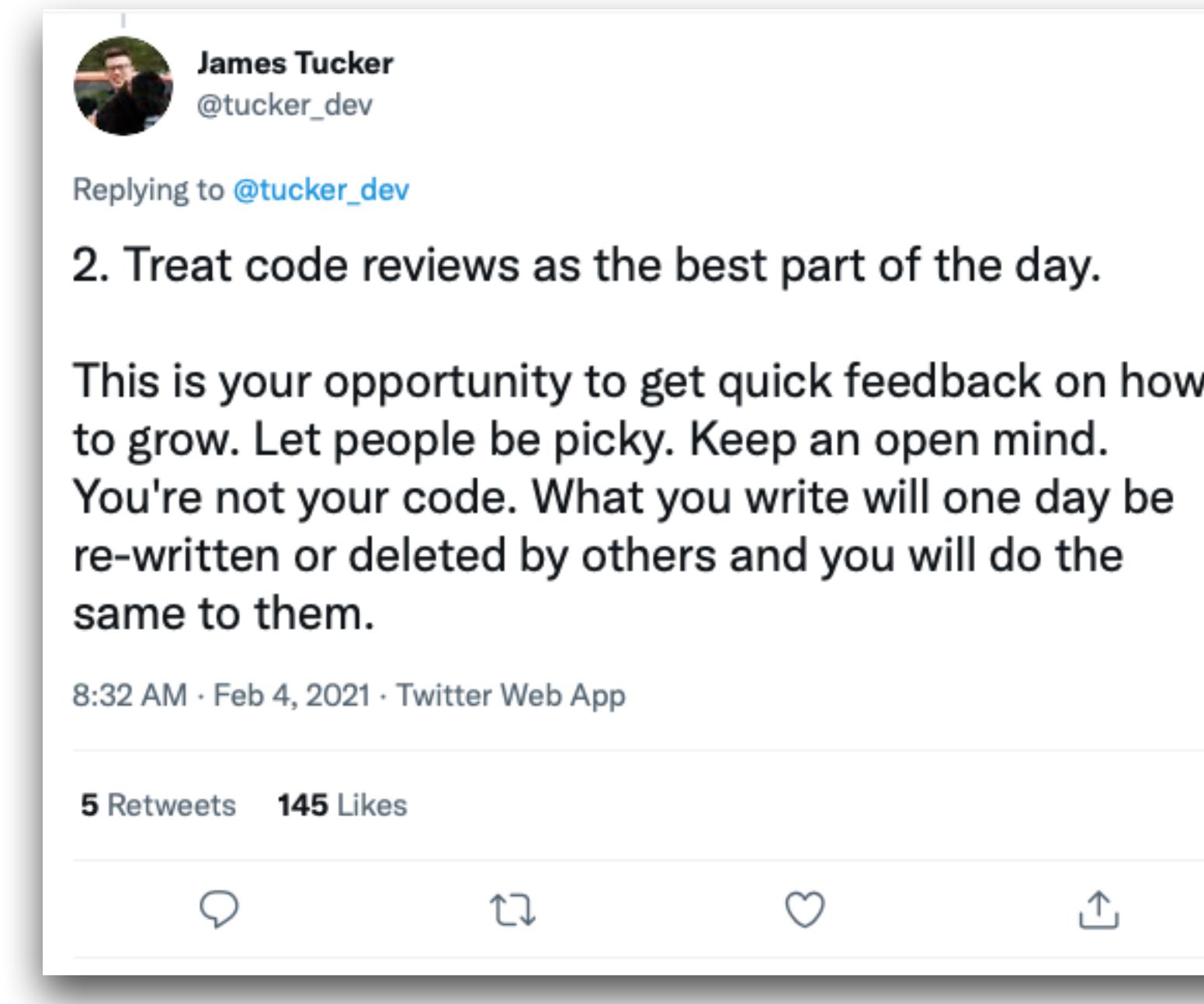
None of us is perfect.

It's all about building
better applications.

“...the only way to make something great is to recognize that it might not be great yet. Your goal is to find the best solution, not to measure your personal self-worth by it.”

-Jonas Downey

<https://signalvnoise.com/posts/3838-strategies-for-getting-feedback-and-not-hating-it>



https://twitter.com/tucker_dev/status/1357336190446301187

We're all on the same team!

It's about the code, not the coder.

No snark.

No sarcasm.

Be humble.

Ask helpful questions.

Did you consider X? How will
the code handle Y?

Is X a concern for this solution?

Share your experiences.

A former project ran into...

When you get to X, ping me
and I'll help with...

Etc.

Don't make proclamations!

This won't scale.

Really? Based on what?

Does that apply here?

Try to sandwich criticism
with compliments.

Really concerned?

Talk to the developer.

Don't ambush them!

No one wins there...

Remember, reviews are a
chance to educate.

“Every single one of us is doing
the absolute best we can given
our state of consciousness.”

- Deepak Chopra

Before you get on
your high horse...

Do you have all the context?

Are there mitigating
circumstances?

Some background you don't have?

There may be...constraints.

Keep it to the code.

Stay grounded.

Back it up.

In some instances, “code review” is a checkbox.

All code must be reviewed
before it is committed.

How does that usually work?

“Hey, Jen, can you
review this for me?”

Click.

Done!



Defeats the purpose.

What about PRs?

 **Justin Searls** 
@searls

I think it's time to discuss the efficacy of Pull Request Reviews as a way to ensure code quality and shared understanding on software teams. Here's a little thread on some of the experiences I've had in my career, why I think this matters, and what we can do about it.

1:23 PM · Mar 12, 2021 · Twitter Web App

275 Retweets 110 Quote Tweets 808 Likes

 Tweet your reply 

 **Justin Searls**  @searls · Mar 12
Replies to @searls
First though, a bit on my background.

In the early days of my career, most teams used cvs or svn. A "code review" was literally a scheduled meeting in a physical room with a 1024x768 projector. Everyone nitpicked your code, it was terrifying, and it took at most two hours.

 2  3  70 

 **Justin Searls**  @searls · Mar 12
Now we use git. git's easy merges enabled GitHub to build a sophisticated Pull Request Review tool & workflow. People love it. Most teams now require an approved review for every PR. Everyone nitpicks your code, it's terrifying, and it takes anywhere from 4 hours to 3 years.

 4  6  128 

<https://twitter.com/searls/status/1370455315246952449>

How can you foster trust?

Collective ownership?

Consider a “bug of the week”.

If you run into something
noteworthy, share it!

Treat everyone with respect.

Do what works.

Do what's right.

Be kind.

Remember the Golden Rule?

Probably learned it
in kindergarten.

Treat others as you
want to be treated.

Write for the developer
that comes after you.

And again...that developer
might in fact be ***you!***

What would future you hope to
find in the code base?

Code is a communication
mechanism.

To other developers.

Write code that is meant to be read!

Don't optimize for a compiler.

Be consistent.

Naming things is hard.

But worth the effort.

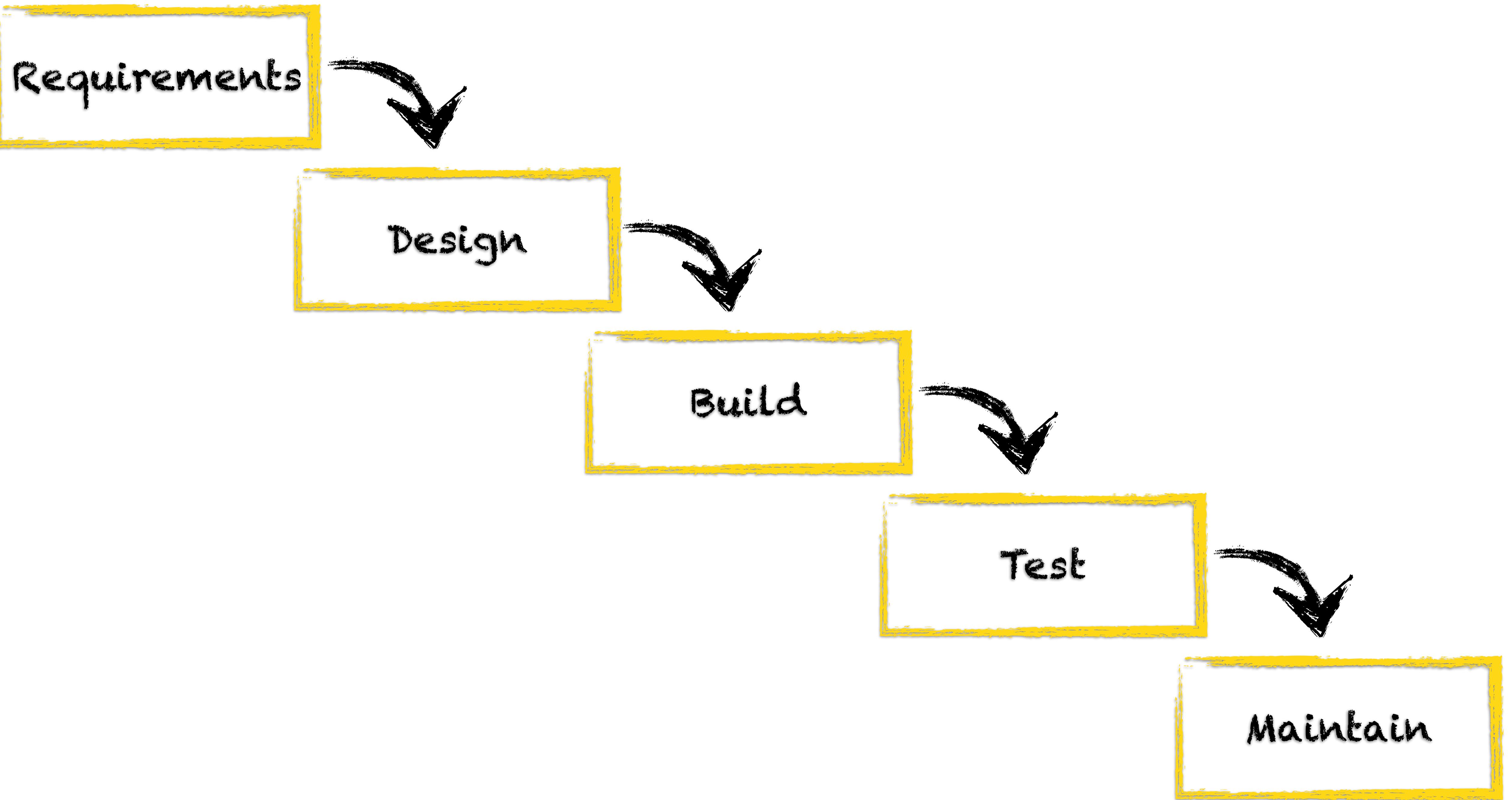
ALL OF US IS SMARTER
THAN ONE OF US.



Software development was
often very siloed.

Remember waterfall?

One phase flowed into the next.



That worked. For some
definition of worked.

Teams were separated.

Business
Analysts

Developers

Testers

UI

Operators

Infrastructure

We didn't always...
work together well.

Sometimes there was animosity
between groups.

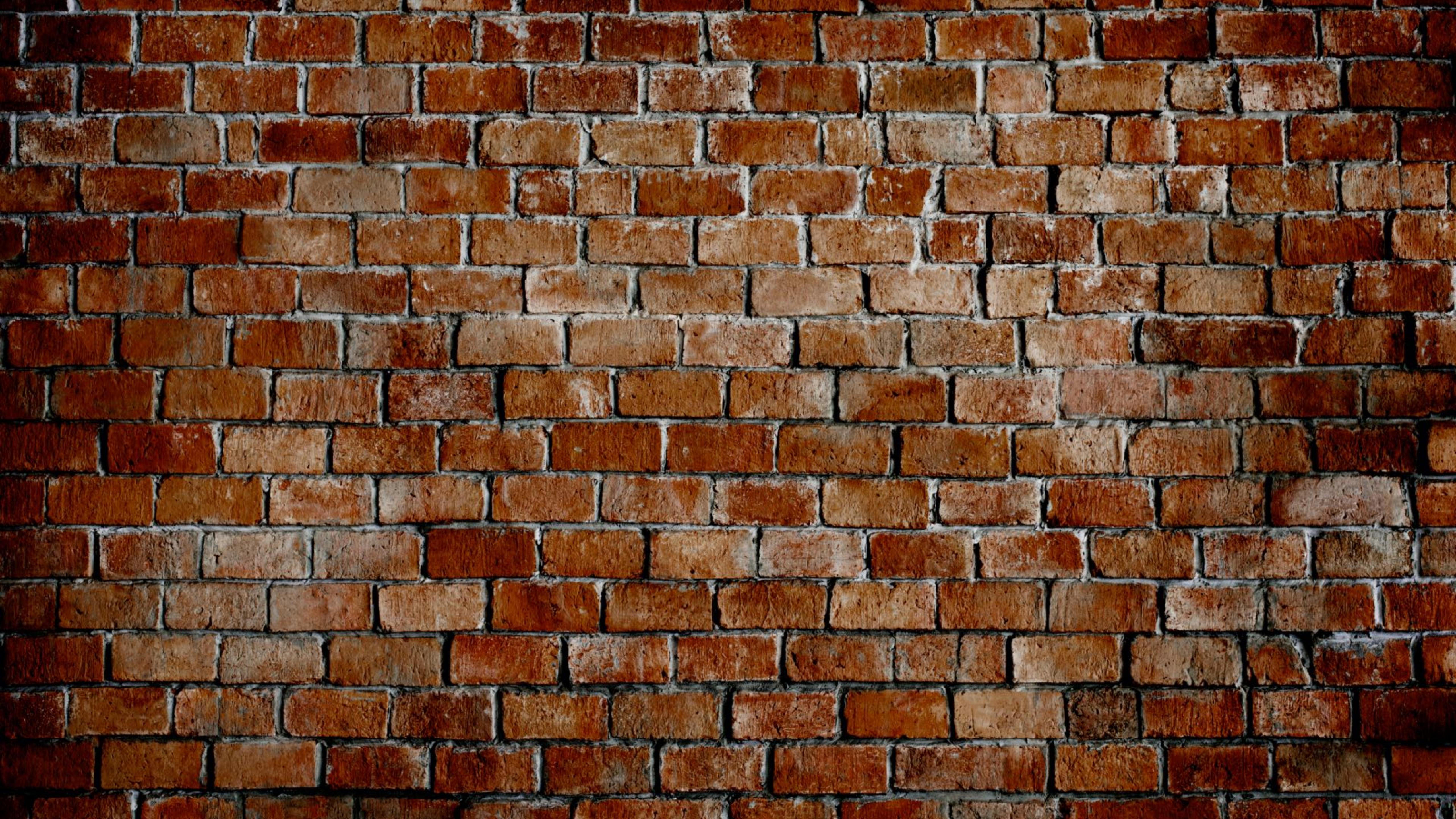




Handoffs were
one way traffic.

Usually scheduled and limited
to a few instances a year.

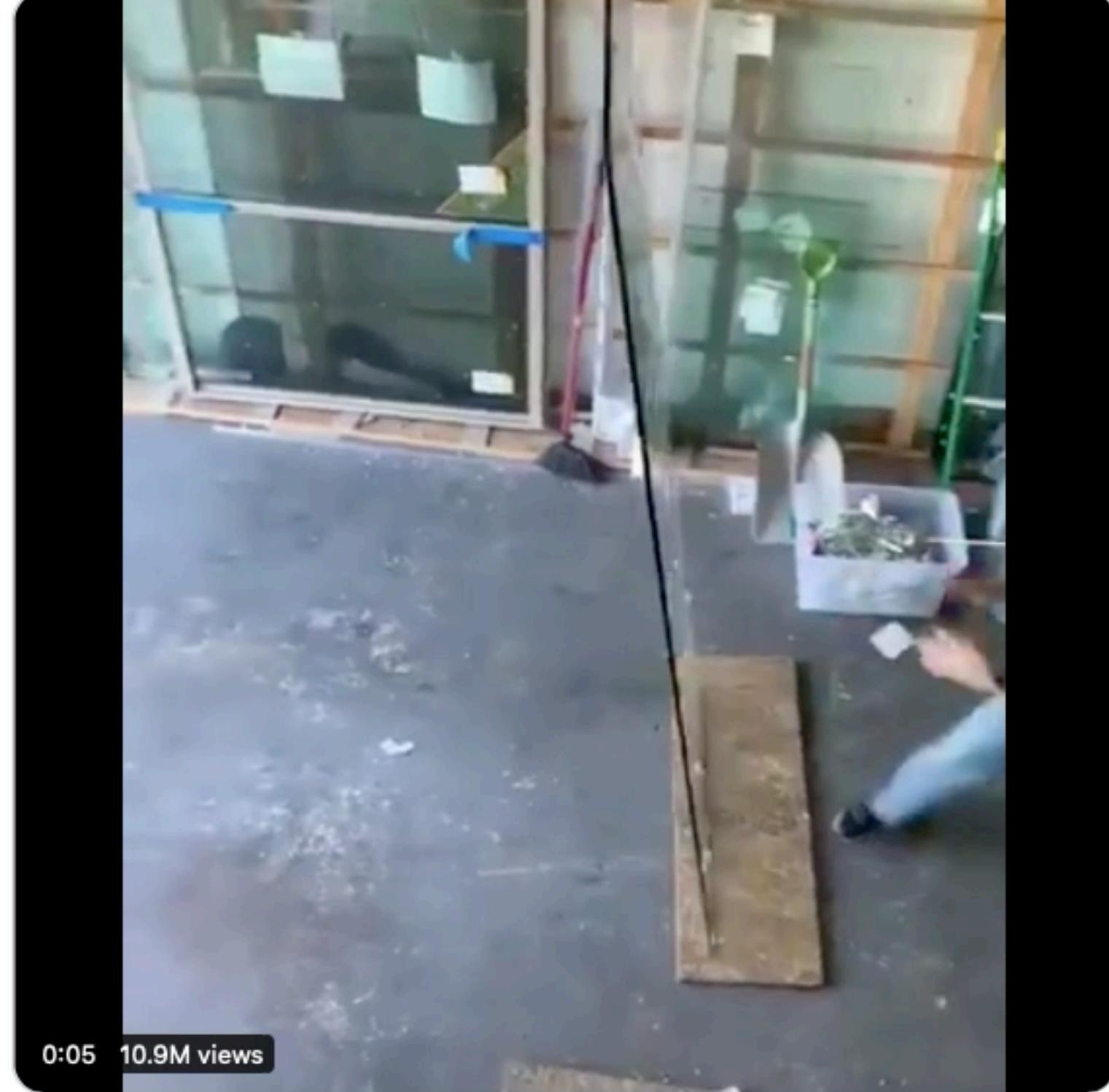
Or to put it another way...



Throw it over the wall to QA.

 Angie Jones
@techgirl1908

This is what happens when you throw your feature over the wall to QA instead of collaborating the whole time



0:05 10.9M views

From Best Engineering videos

7:23 PM · Jan 25, 2021 · Twitter for Android

123 Retweets 21 Quote Tweets 614 Likes

Reply Retweet Like Share

<https://twitter.com/techgirl1908/status/135386158257926146>

Fair amount of finger pointing...

“Works as designed.”

“Unable to reproduce.”

“You brought down prod again.”

Needless to say, not an
ideal way to work.

Cycle times were measured in
astronomical units.



The world doesn't stand still.

Business cycles sped up.
We had to adapt.

Always be changing.

Deliver in days not months.

 **Nate Schutta**
@ntschutta

Yes, even your company in your industry can move away from four deploys a year to, well thousands a month. [#springone](#)



The image shows a man in a dark suit standing on a stage, speaking into a microphone. Behind him is a large white screen displaying the text "After 10 Months, with Our PAAS PCF Platform". Below this, three statistics are listed in red: "29 teams", "4 countries", and "3000+ deploys/month". The Scotiabank logo is visible at the bottom right of the screen. The stage has a red and white geometric pattern on the floor.

<https://mobile.twitter.com/ntschutta/status/938109379995353088>

Speed matters.

Disruption impacts **every** business.

Your industry is not immune.

Amazon Prime customers can
order from Whole Foods.

Some insurance companies
view Google as a competitor.

We're all technology
companies today.

What do we do about it?



Ask a software engineer how to
improve something...

And they'll make it look more
like software engineering.

youtube.com

YouTube

Search

Buy

Saturday Night Live

Get season 46 on YouTube

Will Ferrell Ruined Christopher Walken's Life with SNL's More...

The Tonight Show Starring Ji... ✓

3.9M views • 1 year ago

All Will Ferrell Jimmy Fallon Comedy

Old Movie Stars Dance to Uptown Funk

Nerd Fest UK

57M views • 5 years ago

Matt Foley: Van Down By The River - SNL

Saturday Night Live ✓

18M views • 7 years ago

Will Ferrell Hilarious Acceptance Speech At The...

cocksandballs123

15M views • 9 years ago

Wayne's World: Aerosmith - SNL

Saturday Night Live ✓

2.6M views • 7 years ago

Funniest Game Show Answers of All Time

theSeanGshow

50M views • 7 years ago

The Barry Gibb Talk Show: Bee

Christopher Walken in the SNL Cowbell sketch.

#SNL

More Cowbell - SNL

15,731,005 views • Feb 22, 2019

192K 3.2K SHARE SAVE ...

I gotta have more cowbell...

Testing approach isn't working?

Let's make **testing** more like
software engineering!

Developers are responsible for
writing automated tests.

Frees the testing professionals
to push the application.



Brenan Keller
@brenankeller

A QA engineer walks into a bar. Orders a beer. Orders 0 beers. Orders 999999999999 beers. Orders a lizard. Orders -1 beers. Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

3:21 PM · Nov 30, 2018 · Twitter for iPhone

25.4K Retweets 1.6K Quote Tweets 64.1K Likes

Reply Share Like Retweet

<https://twitter.com/brenankeller/status/1068615953989087232>

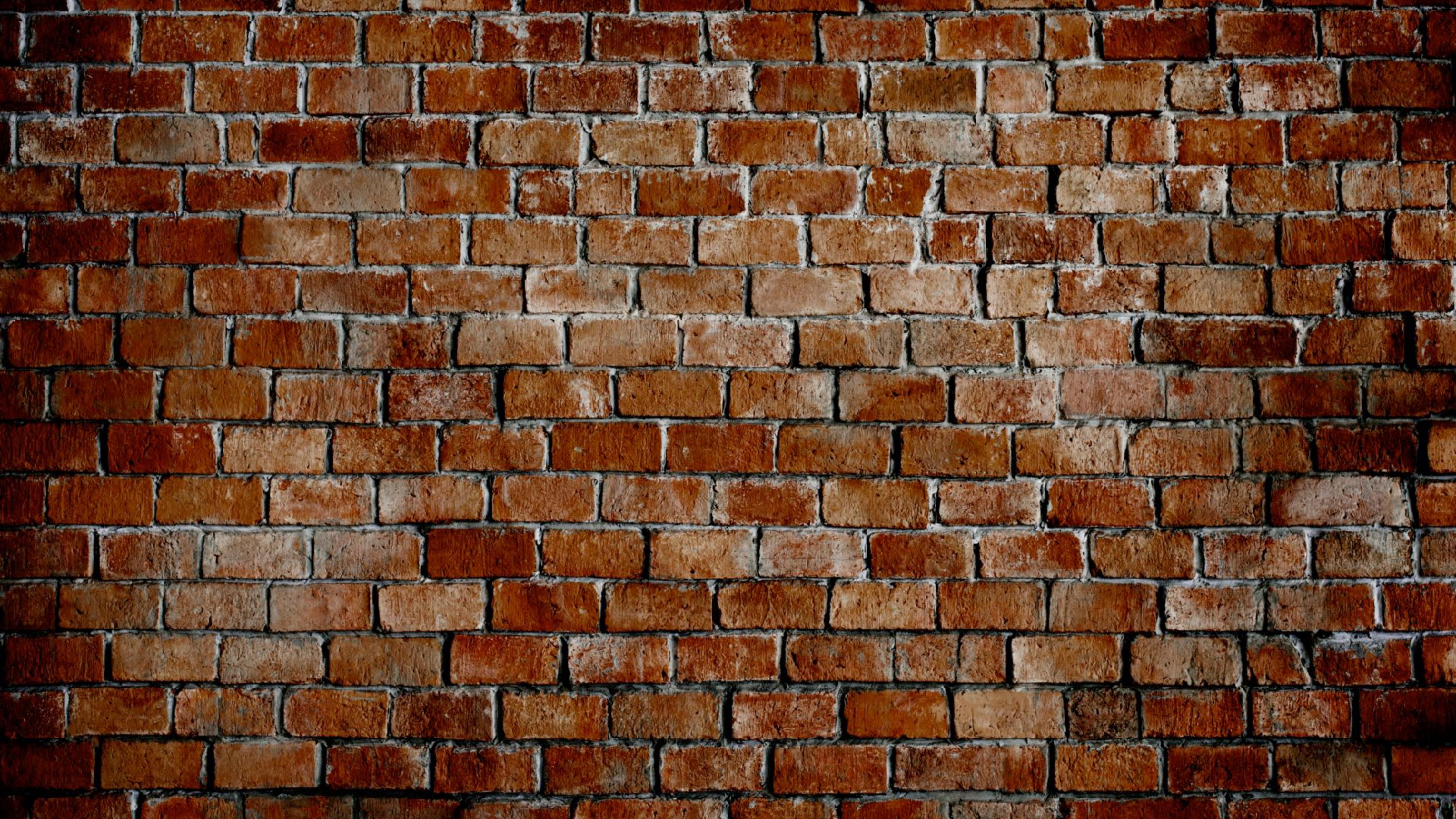
What about production?



Not my problem, besides...

Works on my box.

What did we do?



Think if parent's did this...

The kid is five, they're the
world's problem.

Worked in my house?

Ops wants to keep production
up and running.

And developers just write bugs
and bring prod down.

If we don't release, up
time will be 100%!

Ok, not quite - just make it hard
to release software...

So what did we do?



Let's make ops more like
software engineering!

Developers become the first
line of production support.

Getting paged at 3 am tends to
result in more reliable software.

Obviously oversimplified!

But we were able to
release more often.

Nested feedback loops.

Demonstrate progress,
make adjustments.

But that isn't the only feedback loop.

Developers work with operators
to reduce friction.

Operators work with developers to
bake in fault tolerance.

Smaller batch sizes, less risk.

Run experiments. A/B testing.

Respond to business changes.

Increased development velocity!

Production quality increased.

Better software.

More business value.

Happier customers!

Ultimately software has become
a more collaborative game.

Aligned accountability.

Leverage error budget.

Establish your target availability.

Say 99%.

Your error budget is 1.0%.

Spend it however you want!
Just don't go over that limit.

Goal is not zero outages. Goal is to
stay within the error budget.

Go ahead with those experiments.

Try different things.

Once we use up the
budget though...

Have to slow our roll.

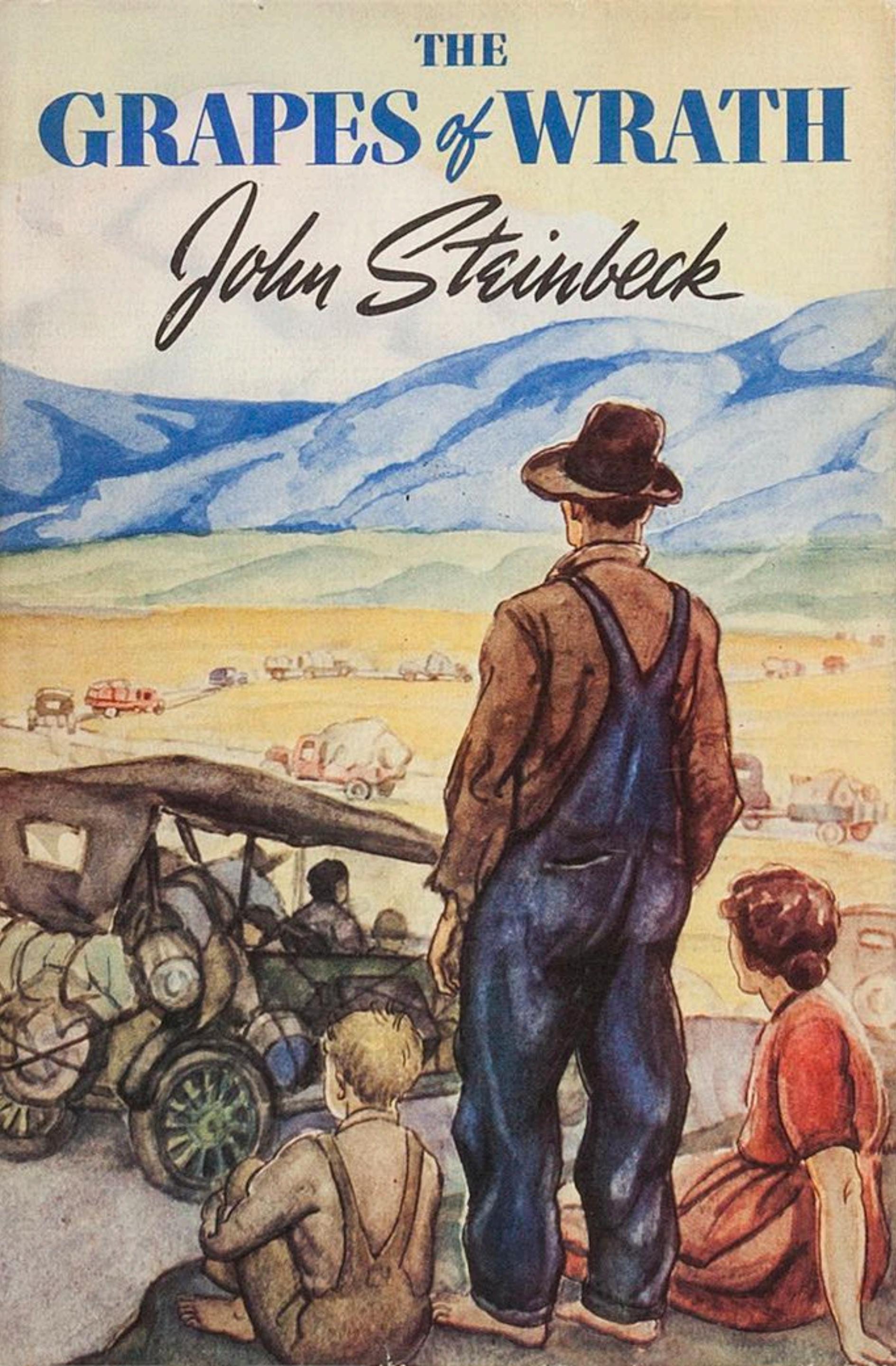
Aligns incentives.

Helps everyone understand
the cost/benefit tradeoff.

Instead of measuring ops on
uptime and devs on velocity...

We are responsible for
delivering business value.

We own this.



Focus on more features to
production more frequently.

Working with others.

The loan wolf developer is a myth.

You will work with people.

We don't just work with other
software engineers.

Designers, Product Owners,
BAs, Testers, DevOps, SRE...

Be respectful.

We can get myopic.

Set in our ways.

“Why do we close the tag on
the following line?”

“That’s how we’ve always done it”.

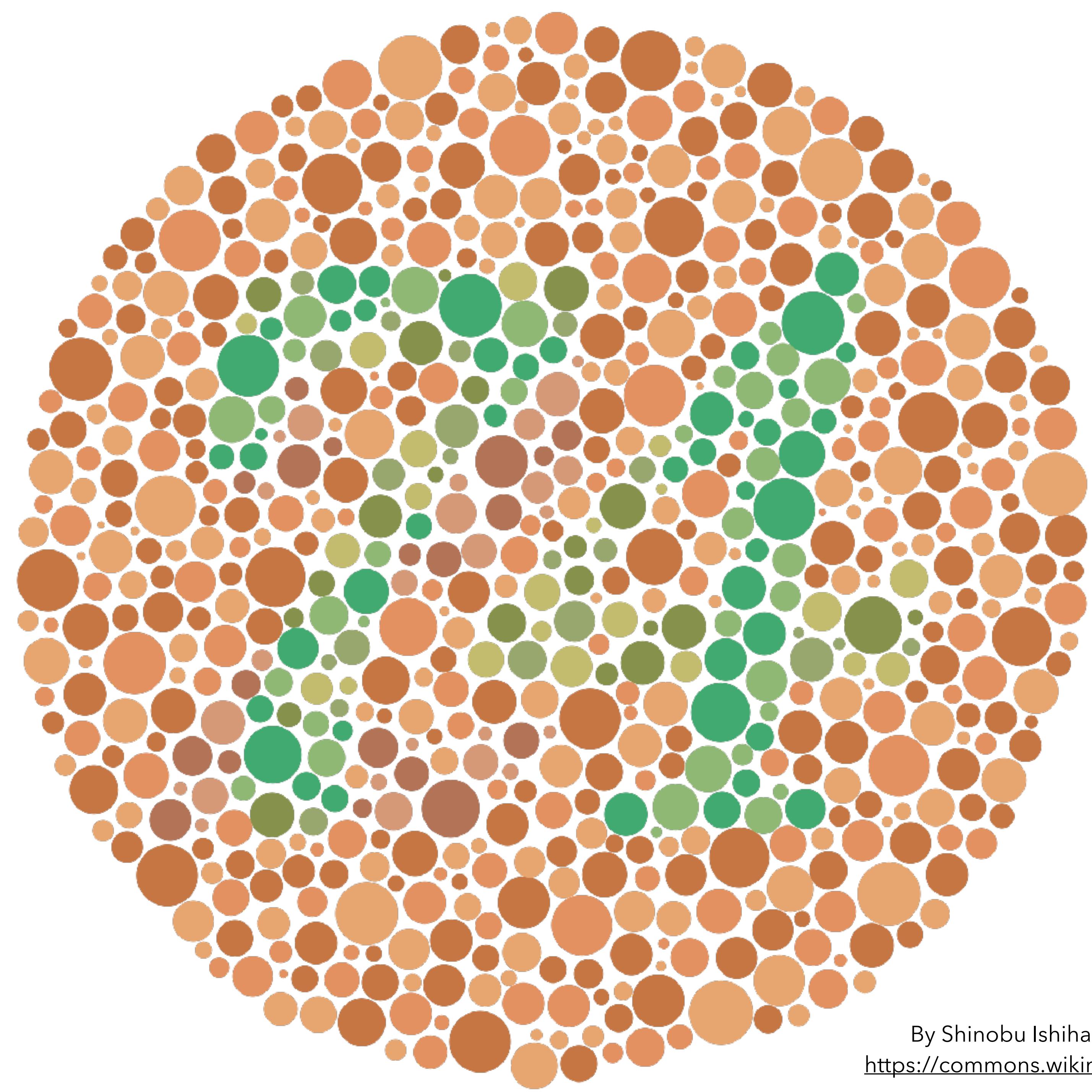
Never dismiss the power of
a fresh perspective.

New person on the team? She'll
find something you missed.

Person right out of school?
No preconceived notions!

Diversity is a strength.

Literally see things we don't.



By Shinobu Ishihara - File:Ishihara 9.png, Public Domain
<https://commons.wikimedia.org/w/index.php?curid=104034287>

Leads to more innovation.

hbr.org

Harvard Business Review

Diversity | Why Diverse Teams Are Smarter

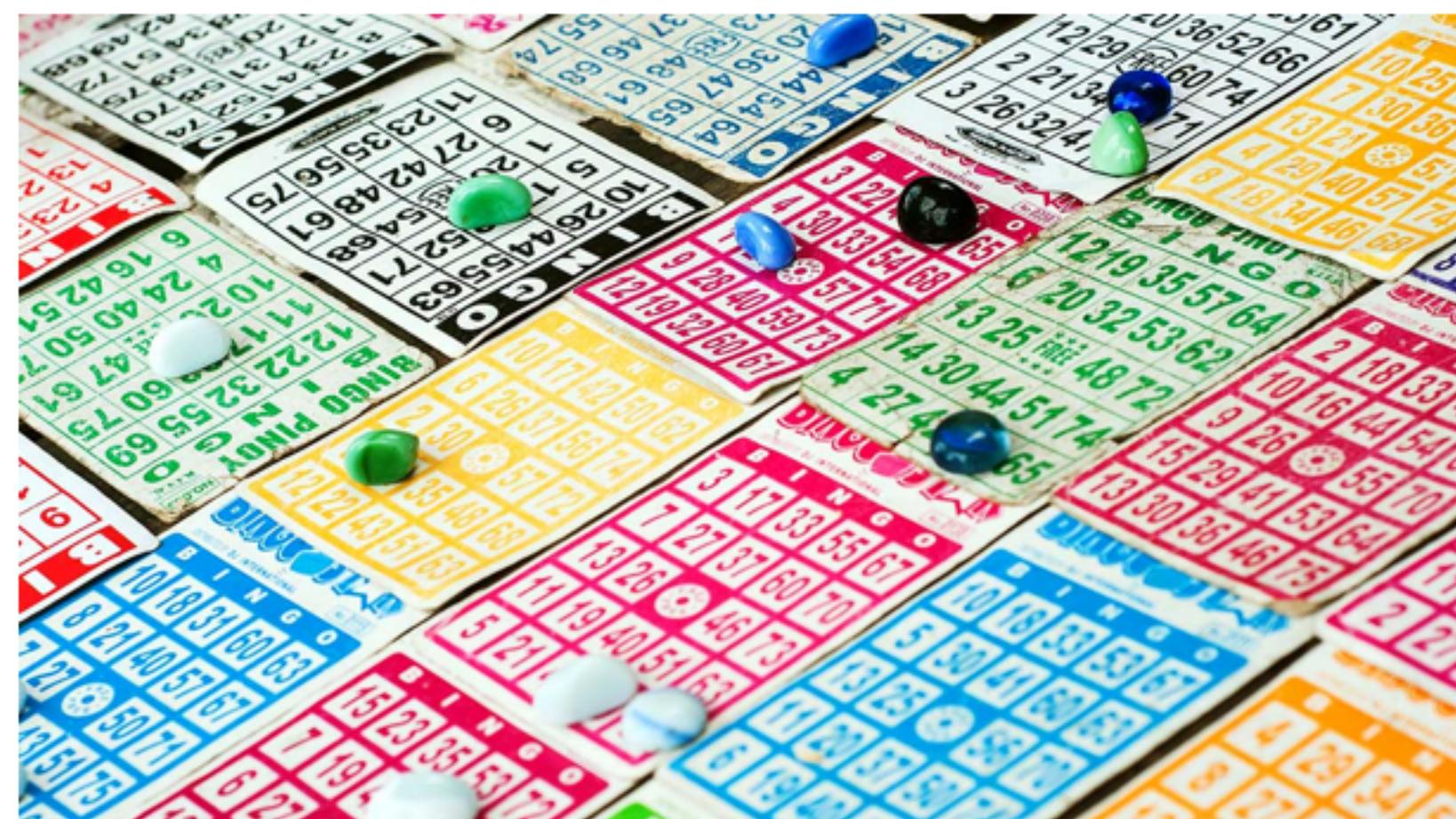
Subscribe Sign In

Diversity

Why Diverse Teams Are Smarter

by David Rock and Heidi Grant

November 04, 2016



Striving to increase workplace diversity is not an empty slogan — it is a good business decision. A 2015 McKinsey report on 366 public companies found that those in the top quartile for ethnic and racial diversity in management were 35% more likely to have financial returns above their industry mean, and those in the top quartile for gender diversity were 15% more likely to have returns above the industry mean.

In a global analysis of 2,400 companies conducted by Credit Suisse, organizations with at least one female board member yielded higher returns on equity and higher net income growth than those that did not.

[Tweet](#)

[Post](#)

[Share](#)

[Save](#)

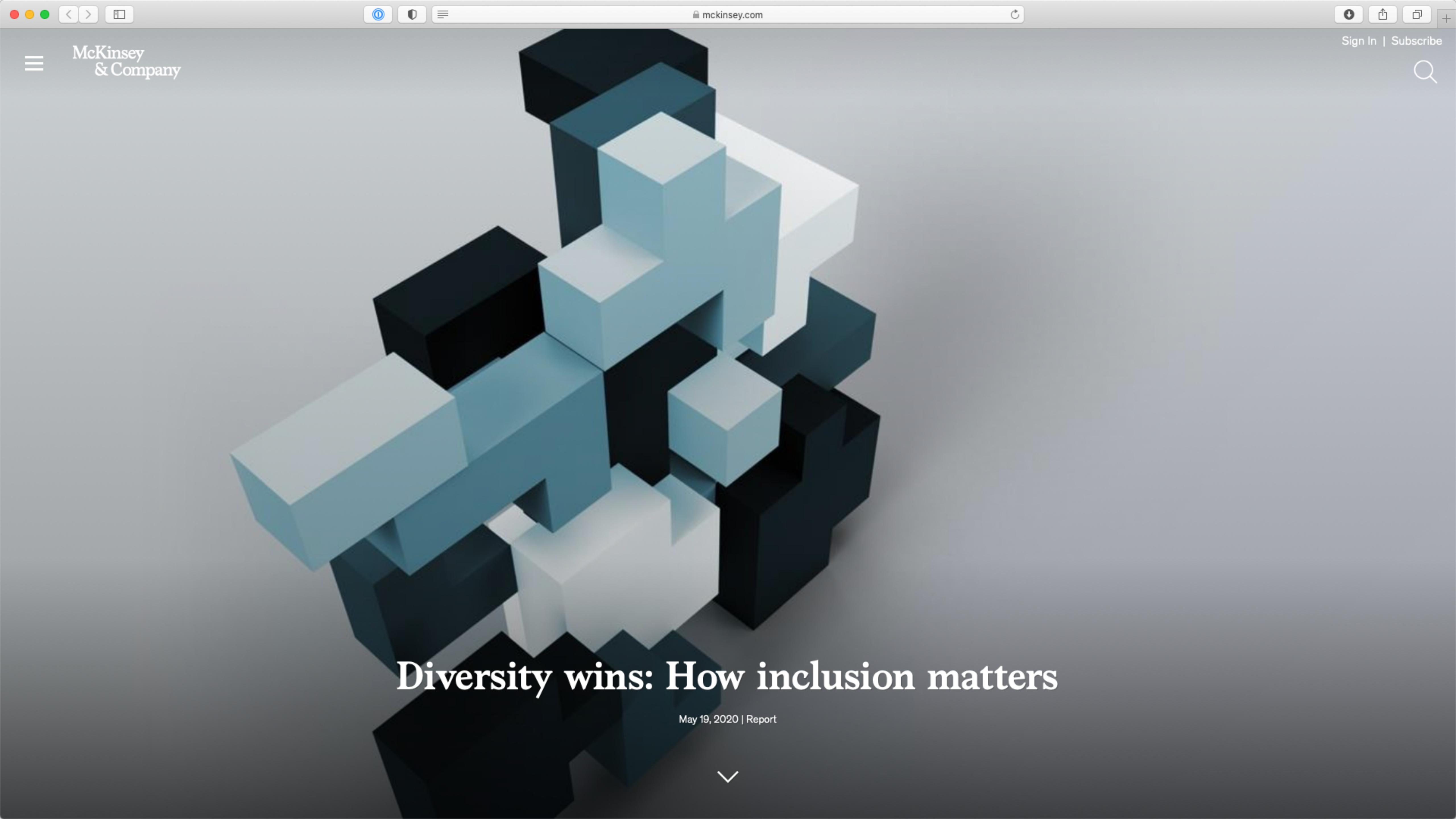
SPECIAL REPORT

Cybersecurity in the Era of Intelligence and an Expanding 'Attack Surface'

[Free download](#)

Sponsored by:

DXC TECHNOLOGY



Diversity wins: How inclusion matters

May 19, 2020 | Report

Don't be dismissive just because
someone has a different role.

Avoid arrogance and
condescension.

No one wins there.

Reciprocity rules...

Be helpful.

Be respectful.

Try to see things from
their perspective.

There might be a rational
reason for current state.

Maybe.

What I told you was true... from
a certain point of view

Learn to speak their language.

Enhanced credibility.

At some point, you won't get
along with someone.

Inevitable.

Could be a personality clash.

They went to a rival school.

Going to have to deal with it.

Be professional about it.

Can help to openly talk about it.

How can we work together?

Can we minimize contact?

May need to discuss
with your manager.

And sometimes you just need
to change the scenery.

A close-up photograph of a white ceramic cup filled with coffee. The coffee has a light brown color with a white, swirling latte art pattern on top. The cup sits on a matching saucer. To the left of the cup, a portion of a striped napkin is visible. The background is dark and out of focus.

COMMUNICATION.

Code itself is communication.

To and for other developers.

OK, and a compiler/runtime.

But that isn't as important as the
humans reading the code.

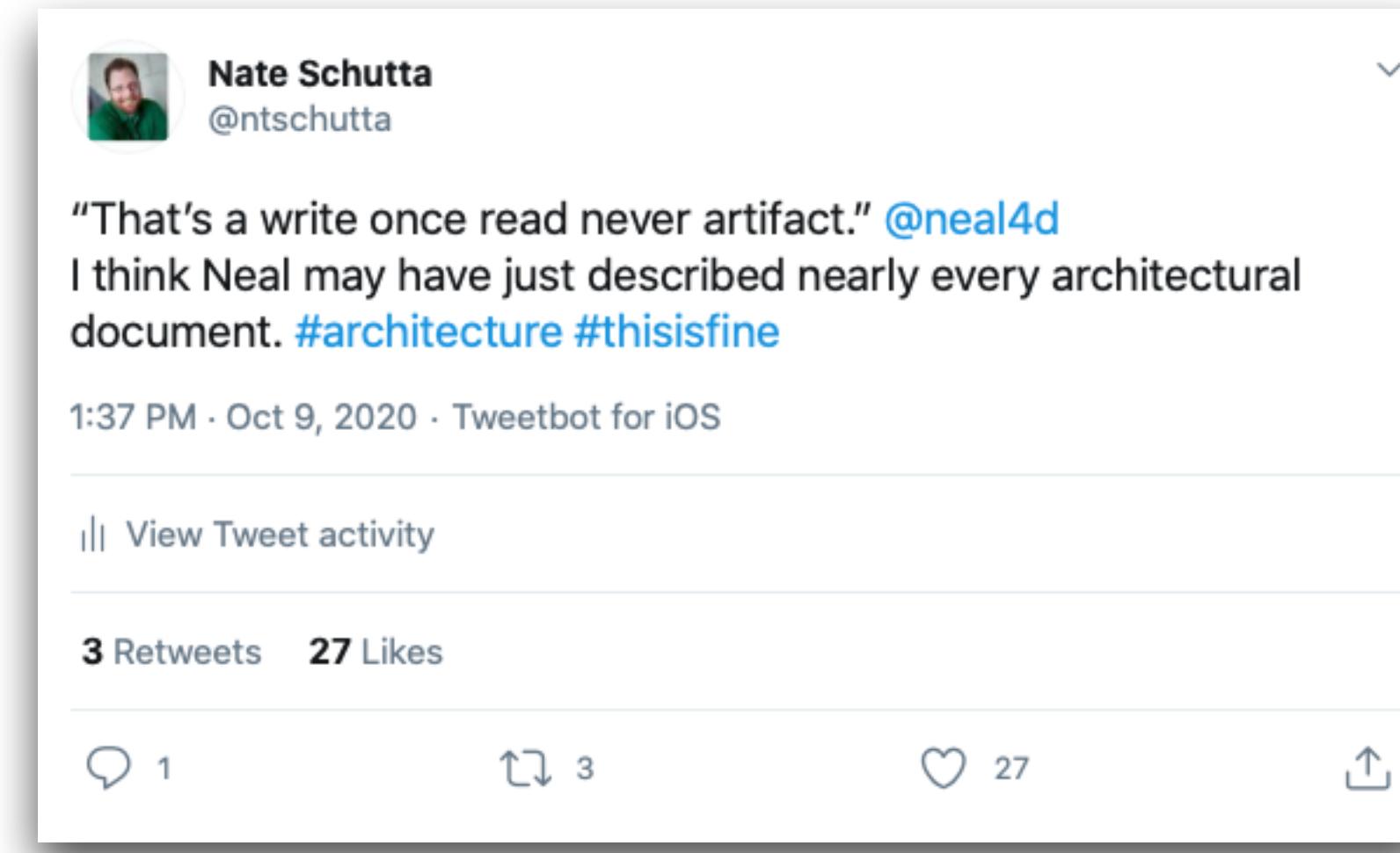
Optimize for the human!

Software engineering is more
than just slinging bits.

Communication skills are
vital for developers.

Don't neglect your
communication skills!

Presenting, software
documents, email.



Nate Schutta
@ntschutta

"That's a write once read never artifact." @neal4d
I think Neal may have just described nearly every architectural document. #architecture #thisisfine

1:37 PM · Oct 9, 2020 · Tweetbot for iOS

[View Tweet activity](#)

3 Retweets 27 Likes

1 reply 3 retweets 27 likes

<https://twitter.com/ntschutta/status/1314636196270739457>

We don't just talk to
other developers.

Must be able to explain it up
and down the org.

What about the people that
sign the checks?

They won't be impressed by
spewing acronyms.

Learn from sales engineers.

How are products marketed?

To whom?

Some vendors speak exec
better than others.

Take notes...

How you communicate
matters more than you think.

Banish RTFM from your repertoire.

Presentations matter!



Presentation **PATTERNS**

Techniques for Crafting Better Presentations



NEAL FORD | MATTHEW MCCULLOUGH | NATHANIEL SCHUTTA



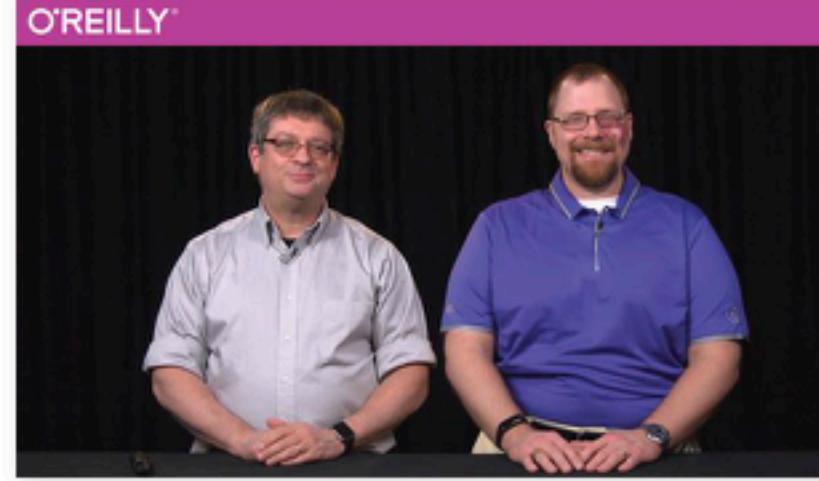
O'REILLY®

Safari

safaribooksonline.com

Search... Enterprise Pricing Sign In START FREE TRIAL

O'REILLY



Presentation Patterns

by Neal Ford, Nate Schutta

Publisher: O'Reilly Media, Inc.

Release Date: March 2016

ISBN: 9781491954980

Topics: [Presentations](#)

[View table of contents](#)

[START WATCHING](#)

Video Description

You're giving a talk on a subject you know inside and out and your audience is staring at their cell phones. You're boring your audience. Maybe you could use some help. In this fast paced humorous video, presentation pros Neal Ford and Nathaniel Schutta provide that help. They've spent thousands of hours giving talks at seminars around the world and even more hours listening to bad ones. They've used this experience to de-construct "The Presentation" into a set of patterns and anti-patterns. What are patterns and anti-patterns? They're simply names (often funny ones) for the building blocks of good presentation practices (patterns) and the stumbling blocks of bad ones (anti-patterns). Ford and Schutta offer concrete instruction in how to plan your presentation, handle a wide variety of presentation types, manage your audiences, and deal with constraints and surprises. They show you what to do, and perhaps more importantly, what not to.

- Learn great recipes for building and delivering effective presentations
- Explore creativity patterns (narrative arc, triad, talklet) that frame and inspire presentations
- Discover how presentation tools (like PowerPoint) can seduce you into doing bad things
- Pick up patterns that can thrill your audience: Composite animation and make it rain
- Learn why hiccup words, laser weapons, and bunkers drive audiences to the snack stands
- Discover the Carnegie Hall, posse, and lipsync patterns that keep you calm while you speak

Neal Ford is Software Architect and Meme Wrangler at ThoughtWorks. His language proficiencies include Java, C#/.NET, Ruby, Groovy, functional languages, Scheme, Object Pascal, C++, and C. He has spoken at over 200 developer conferences and presented more than 2000 talks over the last decade.

Nathaniel T. Schutta is a senior software engineer focused on making usable applications. A proponent of polyglot programming, Nate has written two books on Ajax and speaks regularly at various No Fluff Just Stuff symposia, universities, and Java user groups.

Presentation Patterns.

<http://presentationpatterns.com/>

What is most effective in
your organization?

What style works best?

Embrace that...even if you
have to plug your nose.

What is the pain point?

Can you tie things back
to exec level goals?

Even if it's a stretch...

Sonar, Jenkins.

Listen.

What are the concerns?

Some legitimate issues.

Don't just dismiss them!

Address them.

Have a conversation!

This won't work...



How are you communicating?

With respect?

With derision?

What works better?

Be ready for the game of
“enterprise operator”.

“I heard X said Y is our
enterprise standard”.

“Bob said we’re out of
compliance with...”

You can't stop that.

Especially in large
organizations.

It ***will*** happen, be prepared.

Can you tell the story?

1, 5, 30 minute.

What would you say to
the VP in the elevator?

What would you say at
a team meeting?

What about at an SLT meeting?

What if you were asked
for an overview?

Practice it!

You will repeat yourself.

“Say it 3 times”.

Consider an architectural
concept document.

One or two pages.

Key concerns, gaps,
capabilities, etc.

High level.

Whatever helps you tell the story.

Looking to get promoted?

Be a better communicator!

Take a writing course.

Join Toastmasters.

Speak at a career day
at the local school.

Present at a meetup.

Practice, practice, practice!



ESTIMATES.

Estimates aren't commitments.

“I need a date.”

When has the date ever been right?



“Yeah...I still need a date.”

Estimates are...estimates.

We were often forced to give a date very early in the process.

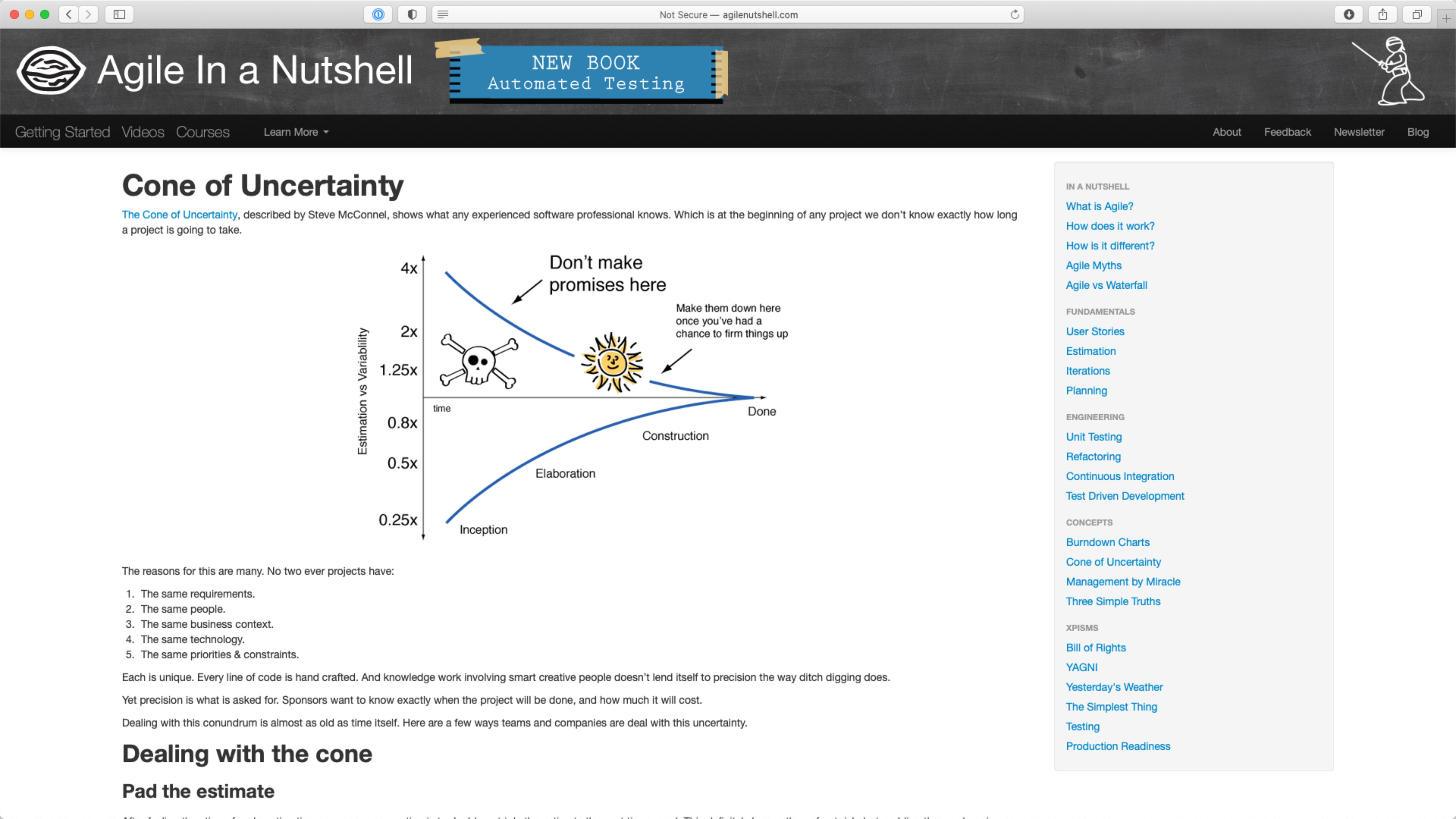
Often when we knew the least.

For example - how much
capacity will you need?



Estimates should be ranges. With assumptions and constraints.

Cone of Uncertainty.



Pad the numbers.

Use relative sizing.

Building this service will take
twice as long as that service.

Show your work - assuming this,
our best guess based on...

Favor incremental funding.

Software is closer to art than
science. At least for now.

Very creative process.

“Building castles in the sky
made out of clouds.”

We're often doing something novel.

How do we accurately estimate
something we've never done?

Hedge your bets...

Three legged stool.

Time, quality and cost.

Pick two.

Want it fast and cheap?

Forget quality.

Want high quality but need it
yesterday? It'll be expensive.

Maybe there's only two choices?

A screenshot of a Twitter post from user @joshholmes. The post features a profile picture of a man with a beard. The text of the tweet reads: "You can either dictate deadline or scope. That is a zero sum game. If you need more scope, then deadline has to move. If you need a closer deadline, then we have to cut scope. The only other lever is quality and I'm not going to compromise on that." Below the tweet, the timestamp "3:23 PM · Mar 5, 2021 from Washington, USA · Twitter for Android" is visible. The engagement metrics show "23 Retweets", "2 Quote Tweets", and "79 Likes". At the bottom, there are icons for replying, retweeting, favoriting (with a red heart), and sharing.

Josh Holmes
@joshholmes

You can either dictate deadline or scope. That is a zero sum game. If you need more scope, then deadline has to move. If you need a closer deadline, then we have to cut scope. The only other lever is quality and I'm not going to compromise on that.

3:23 PM · Mar 5, 2021 from Washington, USA · Twitter for Android

23 Retweets 2 Quote Tweets 79 Likes

<https://twitter.com/joshholmes/status/1367948792964673539>

Eternal struggle of
the software project.

We covered a lot of ground today!

We have a ways to go yet!

Next week:

Design patterns.

Legacy code.

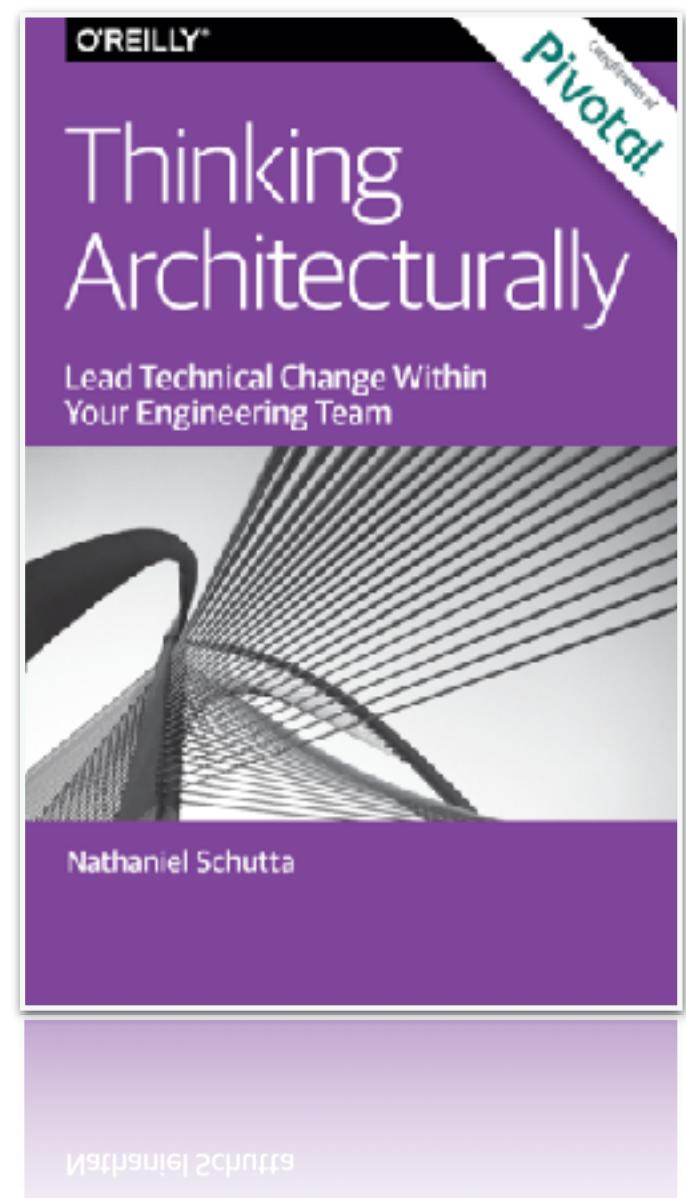
Testing.

Automation.

Learning!

And more!

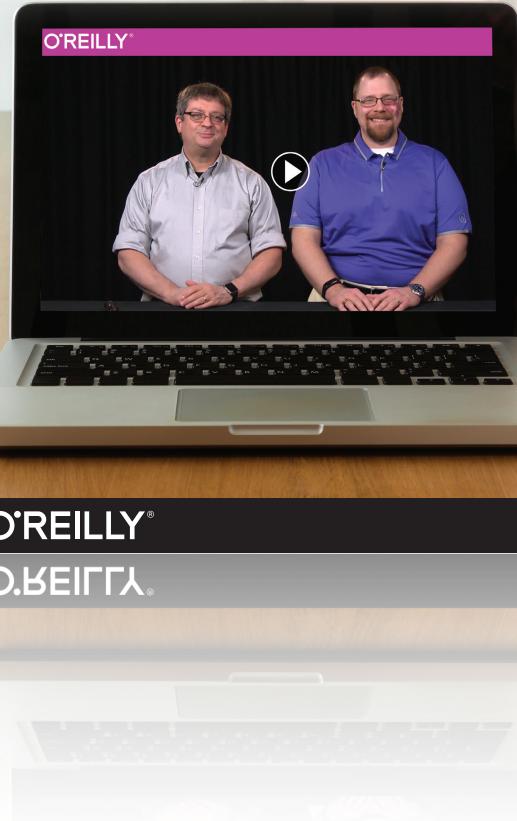
Thanks!



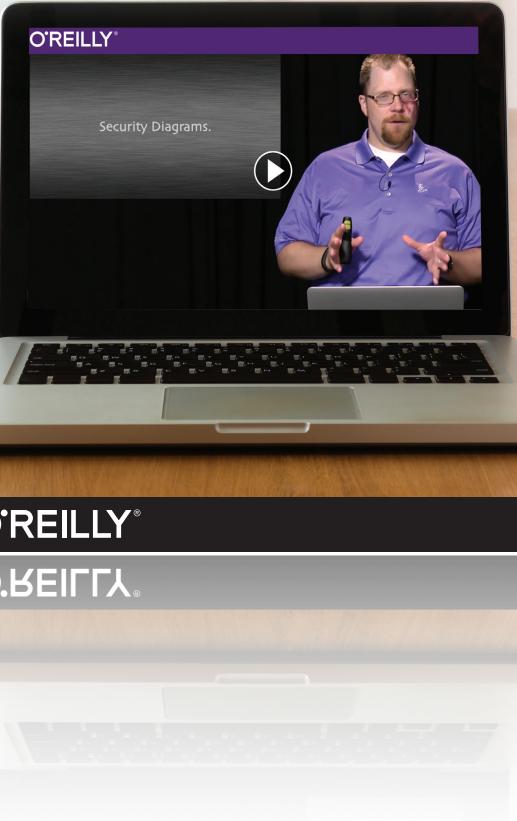
I'm a Software
Architect,
Now What?
with Nate Shutta



Presentation
Patterns
with Neal Ford & Nate Schutta



Modeling for
Software
Architects
with Nate Shutta



Nathaniel T. Schutta
@ntschutta
ntschutta.io

Succeeding with a Microservices Architecture
Best practices for making the move to microservices

Topic: System Administration



NATHANIEL SCHUTTA

Between Chair and Keyboard



Most Mondays,
around noon Central
<https://www.twitch.tv/vmwaretanzu>

Nate Schutta
Software Architect
VMware
@ntschutta

VMware Tanzu Developer Center

Learn Topics Tanzu.TV Community

Tanzu.TV Shows

Explore Tanzu.TV on the VMware Tanzu Developer Center for live demos of modern application development technologies, webinars, and episodes covering coding, Kubernetes, and more!

Follow us on [Twitch](#) and subscribe to our YouTube channels [VMware Tanzu](#) & [VMware Cloud Native Apps](#) to stay up to date with the latest Tanzu.TV content.

Tanzu Tuesdays

LIVE EVERY TUESDAY AT 12PM PT

Let's talk about SpringOne 2021!
AUG 31 2021

Clear-up Coté's Kubernetes Questions with Michael Coté
AUG 24 2021

Staying Ahead of the Curve with Spring and Cassandra 4.0 with David Gi...
AUG 10 2021

Code

LIVE EVERY WEDNESDAY AT 12PM PT

15-factors-with-k8s git clone git@gitlab.com:15-factors-with-k8s/tanzu-tuesday.git k8s-demo-app

15-factors-with-k8s git clone git@gitlab.com:15-factors-with-k8s/tanzu-tuesday.git k8s-demo-app curl http://127.0.0.1:8080/starter.tgz -d artifactId=k8s-demo-app -demo-app -d packageName=com.example.starter

15-factors-with-k8s git clone git@gitlab.com:15-factors-with-k8s/tanzu-tuesday.git k8s-demo-app curl http://127.0.0.1:8080/starter.tgz -d artifactId=k8s-demo-app -demo-app -d packageName=com.example.starter --progress-bar