

DevOps and Agile

Carlos Nunez

July 26, 2023

dev@carlosnunez.me

Hello! Thank you for reading the speaker notes from my presentation, "DevOps and Agile," given at the Agile Alliance's Agile 2023 on July 26, 2023.

I hope you find the contents therein useful.

Questions? Comments? Feedback? I take them all! Email me anytime at dev@carlosnunez.me or find me on LinkedIn at <https://linkedin.carlosnunez.me>.

Agenda

DevOps 101

But we have a problem...

The Three Things™

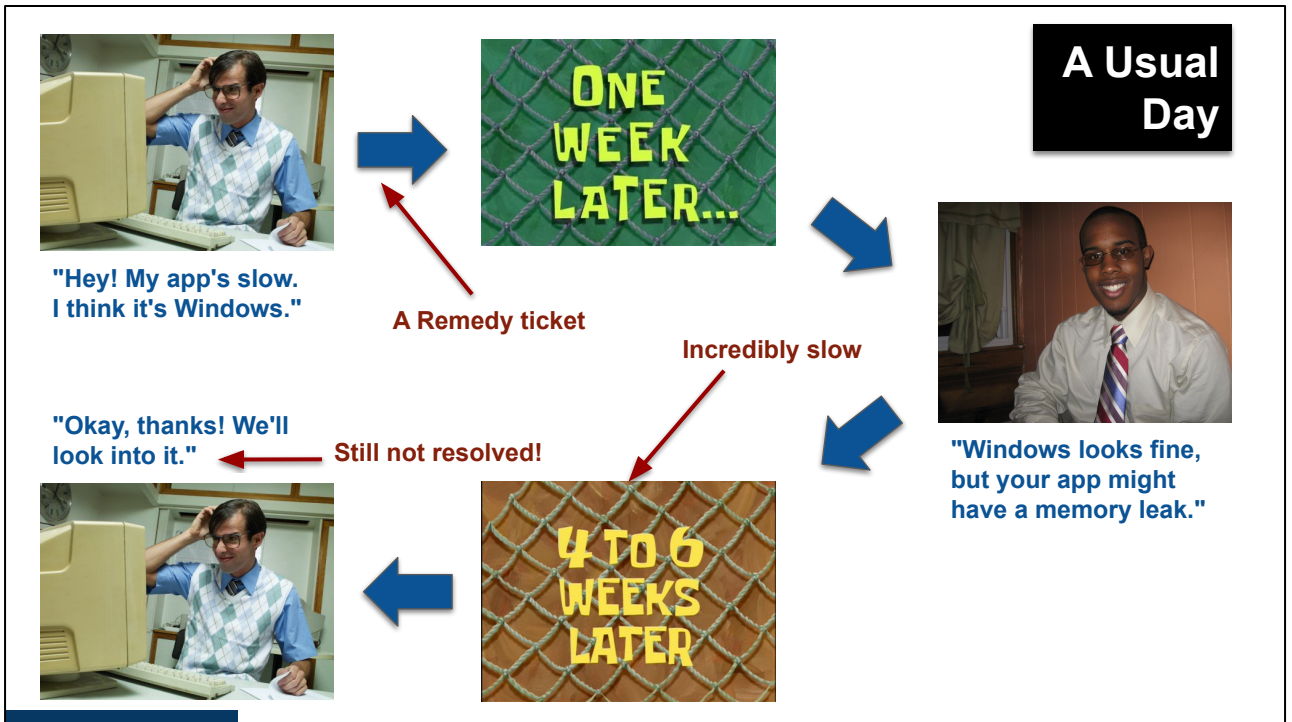
This is what we discussed during this talk.

First, we'll cover a brief crash course on DevOps, a "DevOps 101" of sorts.

We'll talk about the history of DevOps, where DevOps is today and some adaptations of the philosophy seen within the industry.

However, we're not here to *just* learn about DevOps. This is an Agile conference! Unfortunately, I think there's a problem between DevOps and Agile. We'll talk about that afterwards.

From there, I'll discuss three Agile ceremonies that typically catch a lot of flak within DevOps circles: the stand-up, retros and estimation. I'll discuss why they catch the flak that they do and offer some suggestions for how to prevent your teams from suffering the same fate.



History of DevOps

Let's start with a brief history of DevOps and a recap of where it stands today.

What Does DevOps Even Mean?

DevOps means a lot of things to a lot of different people.

For some, DevOps is "the team that gets us stuff from AWS" or "the team that updates our Jenkins pipelines".

For others, DevOps is "the dev team, but they carry pagers now."

For others still, DevOps is literally "dev plus ops; it's in the name!"

While I think these interpretations are directionally correct, they miss the nuance that makes these realities possible. Specifically, I think teams sometimes forget about (or feel disempowered to express) the cultural aspects of what makes DevOps *DevOps*.

To me, DevOps, like Agile, is a *thing that you are*, not a *thing that you hire*. Or, as Nicole Forsgren, author of *Accelerate*, puts it:

“You can’t “implement” culture change.”

Atlassian, author of popular project management tools like Jira and Confluence, formally defines DevOps as follows:

DevOps is a set of [practices](#), [tools](#), and a [cultural philosophy](#) that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.¹

This definition is fine, but it's difficult for people who have never done development or operations to grasp. I like to use my experience in a past life to paint a better picture of what this really means.

Life of a Request

In 2008, I was a Windows systems administrator at an Australian investment bank in New York. (This is a picture of what the author wore on his first day!) While we were largely responsible for tending to our fleet of Windows servers worldwide, a big part of that involved responding to requests from development teams who used these servers to run business-critical applications.

This slide illustrates a typical interaction between a "Wintel" sysadmin (me) and a project manager or (if we were lucky!) a software developer that we peripherally supported.

As you can see, this interaction is pretty boring.

A developer is noticing some slowness in their applications' performance. Out of ideas, they turn to us thinking that the operating system might be the issue.

We would do some traditional systems troubleshooting, usually find nothing wrong with the OS, inform the developer that the system is working fine and close the ticket.

The developer thanks me for the time and continues to troubleshoot on their own.

While it seems like there isn't much going on in this conversation, a lot is lurking beneath the surface.

Communication

At this bank, the developers spoke to us through Remedy problem tickets. An example of the UI that developers would have used is shown above...and this was an upgrade to what we had before!

The Remedy tickets lacked a lot of context. Often, you wouldn't know what app the developer was talking about, let alone the source code behind the application.

Moreover, actually filling out these tickets was painful! What could have been an easy conversation on Lync took several minutes and tens of clicks instead. Lync had Group Chat, but it was awful, and few used it.

Turnaround Time

Since there were many more developers than there were Windows sysadmins, tickets queued up. A turn around time of a week was pretty common. Consequently, many tickets were "urgent" and escalated through more senior leadership.

This added to the frustration that developers often felt when working with us.

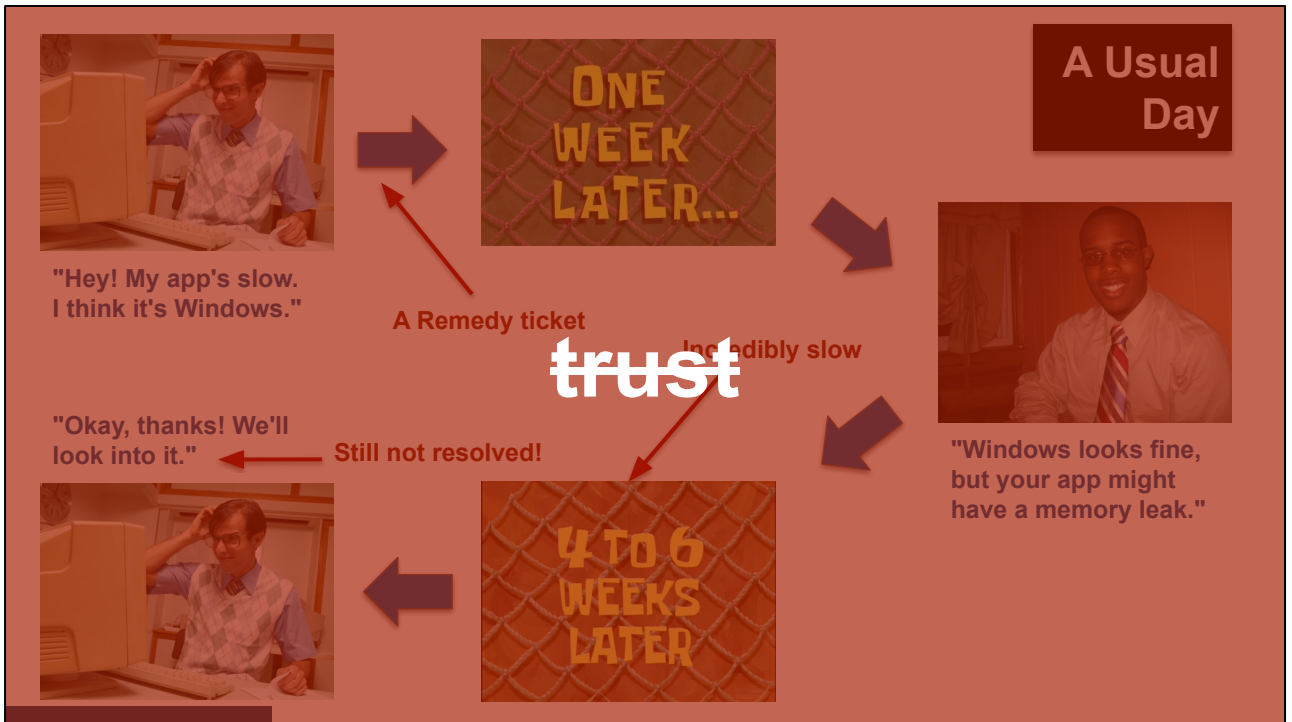
Misunderstandings

The biggest problem hidden in this conversation is demonstrated by the resolution.

The developers thought their app was fine and that the operating system was the problem. Who could fault the developer for thinking this? They know their app and know how to write software, but didn't know very much about their systems.

We thought the operating system was fine and that the app was the problem. Who could fault us sysadmins for thinking this? We know how Windows worked and racked and stacked our servers, but knew next to nothing about writing software.²

Put another way, devs knew nothing about ops, and ops knew nothing about dev.



History of DevOps

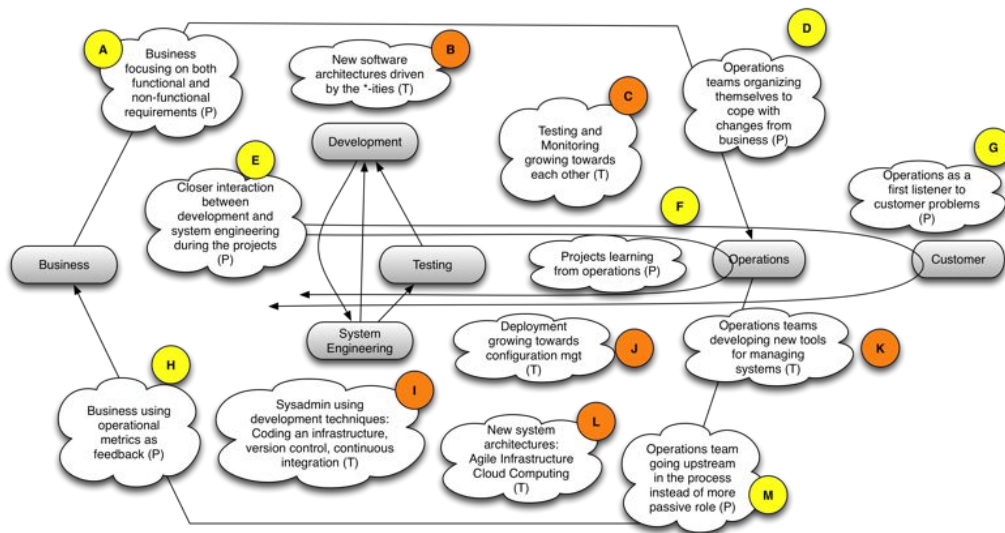
All of these inefficiencies and constraints eventually lead to one thing: distrust.

Distrust from developers in the sysadmins being skilled enough to do right by them.

Distrust from sysadmins in the developer's ability to understand the platforms their software runs on.

Distrust from the business in engineering's ability to fix things quickly when problems arise, or ensure that the fixes they put in place are permanent.

Distrust from customers in the business's ability to meet their needs.



...maybe the idea was too crazy; developers and ops working together.

Patrick DuBois

Talking Points

- Silos like this existed everywhere
- "But what if we worked more closely together" was thought by many, but talked about by few
 - Trading floor
 - Limoncelli
- "What if Ops did Agile?"
 - DevOpsDays 2009
 - Conferences popped up

History of DevOps

And Now for Something Completely Different...

Many sysadmins and developers experimented with ways of tearing down these barriers.

At the Australian bank that I worked at, I proposed having us sysadmins sit on the trading floor with the traders and quants (devs) so that we could (a) see some really cool Wall St action happen, and (b) better understand what our customers needed and struggled with so we could help them work more efficiently. We tried a very limited alpha of that. It didn't last long, as they weren't organizationally ready for that yet.

Tom Limoncelli, SRE manager at Stack Exchange, even wrote an awesome book in 2001 that, if you glance hard enough, looks like it could have been written today! It's called *The Practice of Systems and Network Administration*, and it's an awesome read regardless of how technical you are!

However, it wasn't until 2009 when Patrick Dubois, Paul Nasrat, Rick Simmons, Charles Loomis and others around the world started a conference to discuss one question that's been thrown around for years prior:

"What if Ops did Agile?"

(Fun fact: Rick Simmons, an Agile Coach, presented on this topic right here at the Agile Conference in 2009! It was called "What does Agile ops look like," and I could absolutely NOT find a slide deck from it for the life of me.)

The initial sketch of answers to this question looked like the image on this slide. In fact, was one of the first outputs of the first DevOpsDays conference in the world, DevOpsDays Ghent 2009. As you can see, figuring out how to apply "people over process" to IT operations was not easy! Here's a quote from Patrick's blog post two weeks after that conference:

"I'll be honest, for the past few years, when I went to some of the Agile conferences, it felt like preaching in the dessert. I was kinda giving up, maybe the idea was too crazy: developers and ops working together." [0]

Footnotes

[0] **Dubois, Patrick.** *Devopsdays 2009 - Two weeks later.* 2009 Nov 15.

<https://www.jedi.be/blog/2009/11/15/devopsdays09-two-weeks-later/>

[1] **Dubois, Patrick.** *Charting out devops ideas.*

<https://www.jedi.be/blog/2009/12/22/charting-out-devops-ideas/>



Talking Points

- Conference talks intensified
- "We release 10 times a day" really got people's attention in 2009
- The explosion happened after *Accelerate*
 - Translated "nerd" talk into numbers and *money*
- The explosion intensified after *Phoenix Project*
 - Parts Unlimited
 - Brent
 - *Sounds like someone you know?*

History of DevOps

The Explosion

As it happened, this idea of "Agile Ops", or "DevOps," was slowly (and thankfully!) proliferating throughout the world.

An eye-catching example of that at the time was Paul Hammond and John Allspaw showing the world how they deployed changes to Flickr into production multiple times per day at O'Reilly's Velocity conference in 2009.

However, it wasn't until Nicole Forsgren, Gene Kim, founder of Tripwire, and Jez Humble, founder of the DevOps Research Assessment (DORA) published

"Accelerate" that DevOps crossed the chasm from experimentation to reality.

The authors of *Accelerate* surveyed hundreds of organizations about the efficiency of their IT operations to answer multiple questions like "How does downtime happen" and "What causes IT projects to fail?" By using the scientific method and empirical analysis, Dr. Forsgren and Kim concluded this, amongst other things:

"Furthermore, over the last few years we've found that the high-performing cluster is pulling away from the 800 pack. The DevOps mantra of continuous improvement is both exciting and real, pushing companies to be their 400 best, and leaving behind those who do not improve. Clearly, what was state of the art three years ago is just not good enough for today's business environment." [0]

The magic behind *Accelerate* was its ability to adapt what "devops practitioners" have been saying for ages (up to this point) into quantitative data that fits well in senior and executive leadership circles.

...and then Brent appeared

This tour de force accelerated harder in 2014 after the release of *The Phoenix Project* by Gene Kim, Eric Behr and George Spafford.

The Phoenix Project chronicles the transformational journey of *Parts Unlimited* from a old-world Waterfall stalwart on the brink of death to a burgeoning auto parts manufacturer bolstered by modern software practices and, of course, DevOps philosophies.

(I think *The Phoenix Project* is a must-read. Well, the first 150 pages of it or so. It's 2021 follow-up, *The Unicorn Project*, ehhhh...you can skip it. :D)

Central to *The Phoenix Project* is Brent, a Parts Unlimited veteran and classic old-school sysadmin. He knew where the bodies were buried...and attended several corporate funerals of his own. More importantly, he knew all of the systems powering Parts Unlimited: how they worked, where they were, how to fix them, and when to upgrade them.

While this made Brent indispensable to the company, it also made Brent a constraint. Everything involving Parts Unlimited, from routine ops to major new releases, had to flow through him.

As you'd expect, he never took vacations or sick days, was always available, worked harder than almost anyone else at the company, and, consequently, was beyond burned out.

Does this sound like someone you know? It probably does!

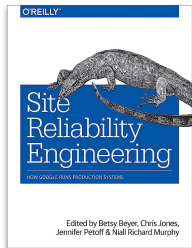
A major goal of the DevOps movement was to both save the Brents of the world and prevent more of them from sprouting.

Footnotes

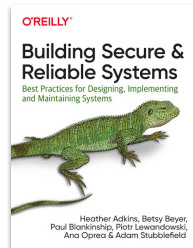
[0] **Forsgren, Kim, Humble.** *Accelerate: Building and Scaling High Performing Technology Operations.* **2018.** pg. 44, ISBN: 978-1942788331

LEFT:: A picture from DevOpsDays Ghent 2009

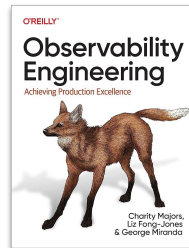
RIGHT: A picture of "Accelerate", "The DevOps Handbook", "The Phoenix Project"



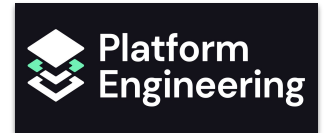
2018



2020



2022



2023

Many of the teams that are 'doing DevOps' well don't even talk about DevOps anymore—it's simply how they work.

Puppet Labs

Talking Points

- **Ask the audience:** *How many of y'all ship to production*
 - Whenever you want!
 - Hourly
 - Daily
 - Monthly
 - Yearly?
- DevOps has evolved quite a lot!
 - two-fold increase in people shipping daily to monthly
- At least seven million tools have come out since then
- Three ideologies
 - SRE
 - Observability Engineering
 - Secure Software Supply Chains
 - Platform Engineering
- One thing in common: DEVOPS

History of DevOps

Five years later

Several off-shoots of the DevOps ideology have emerged since *The Phoenix Project* took the world by storm.

The major books that spearheaded these movements are shown above.

Site Reliability Engineering, released in 2018, describes, well *Site Reliability Engineering* as practiced by Google, the inventor of the practice. SREs use software engineering, systems knowledge, and lots and lots and lots of data to help software engineers keep their applications up and running in production (or production-like environments), no matter the scale. While it might not seem that way from reading the book, developers and operations working together is central to the SRE ideology.

Observability Engineering, released in 2022, generalizes the SRE concept beyond web-based architecture and provides a framework that brings Reliability Engineering to many companies small and large. Just like the SRE paradigm it came from, developers and operations working together is central to Observability Engineering.

Building Secure and Reliable Systems, a COVID-19 special, introduces "Secure Software Supply Chains", or processes for knowing the origin and authenticity of every artifact produced by a software release pipeline and every system and piece of software that it touches along the way. This book helped formalize the idea of "DevSecOps," an ideology that reminds developers and operations to think security-first.

Finally, there has been an uptick in the adoption of "Platform as Product," an idea that merges DevOps with Lean, User-Centered Design, and other core Product Management tenets. Put simply, "Platform as Product" challenges engineering teams to build and treat platforms like products customers use, i.e. interview your customers, build iteratively, and experiment. The popularity of "Platform as Product" has brought about the "Platform Engineer," a software engineer that tends to the platform and, if you glance hard enough, looks kinda, sorta, *maybe* like an SRE of yore.

Lately, I've been seeing and hearing talks of DevOps being dead. Given the lack of talks highlighting DevOps in talk circles lately, this thought is pretty understandable.

However, instead of thinking that DevOps has died, I like to think of DevOps as having *evolved*.

Nothing shows this transformation better than the 2022 *Accelerate* State of DevOps report from DORA and Google Cloud. The number of companies shipping to production every month has increased almost two-fold from 2018 (from 37% to 69%)!



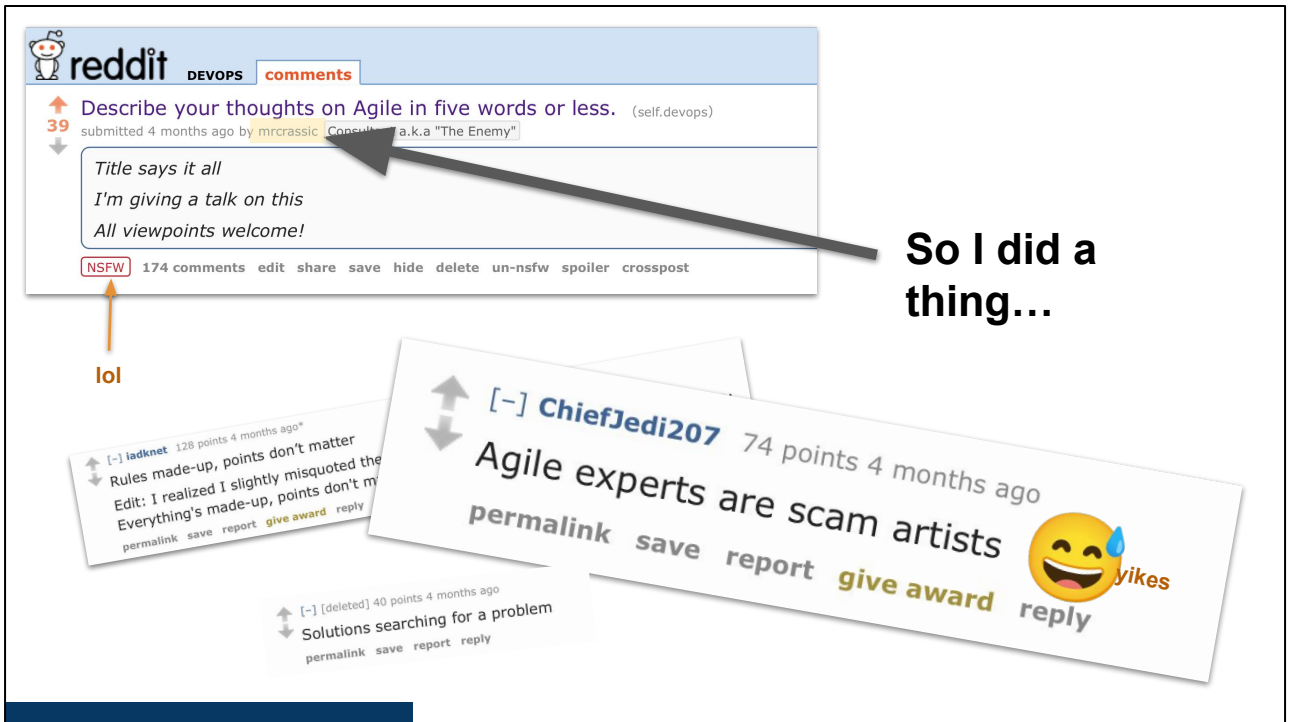
Talking Points

- DevOps progress in breaking down silos
- But it seems like everyone hates it on the Internet?

But we have a problem?

Despite all of the progress that the DevOps ideology has made in getting engineering teams to work together better, I've noticed a pattern in many conversations about DevOps on Reddit, Hacker News, Twitter, and elsewhere.

As crazy as it sounds, it feels like DevOps hates Agile sometimes.



So I did a thing...

lol

Talking Points

- I made a Reddit post to learn what the community thought
- The responses were...fruity
- Here are some examples
 - "...and my favorite, 'Agile experts are scam artists'"

This conflict appears **everywhere!**

Here are some screenshots of posts from a Reddit post I made on /r/devops that make me think this.

It's crazy to think about a movement **that was literally born and raised in Agile conferences like this one** eventually hating...itself?

I thought a second opinion was in order, so I went deeper.

do devops people hate agile?

As an AI language model, I can't speak for every individual in the field of DevOps or Agile, but I can provide some general insights.

DevOps and Agile are two distinct approaches to software development and delivery that are often used together. While there may be varying opinions among professionals, it is not accurate to say that DevOps people hate Agile or vice versa. In fact, both DevOps and Agile methodologies share similar principles and values, and they are often used in conjunction to improve software development and delivery processes.

DevOps emphasizes collaboration, communication, and automation between software development teams and operations teams, with the goal of delivering software quickly, reliably, and with high quality. Agile, on the other hand, is a software development approach that focuses on iterative development, continuous feedback, and adaptability to change.

Many DevOps practitioners appreciate Agile methodologies because they align well with the DevOps philosophy of delivering value quickly and continuously improving processes. Agile methodologies such as Scrum or Kanban provide frameworks for managing development iterations, organizing work, and facilitating collaboration.

That being said, like any professional field, there can be different opinions and preferences within DevOps regarding the specific implementation of Agile methodologies. Some individuals may have had negative experiences with Agile due to various reasons such as poor implementation, lack of understanding, or misalignment with organizational goals. However, it would be inaccurate to generalize these experiences to the entire DevOps community.

In summary, DevOps and Agile methodologies are not inherently opposed to each other, and many DevOps practitioners appreciate the benefits that Agile brings to the software development process. Both approaches aim to improve software development and delivery, and their effective combination can lead to successful outcomes in many organizations.

I even turned to Special Forces for answers.

interesting

Does DevOps Hate Agile? *Let's Ask The Expert.*

Unsatisfied with my crowdsourced bank of responses, I decided to ask the experts: ChatGPT.

This is what ChatGPT had to say about the relationship between DevOps and Agile.

Compared to my responses from Reddit, this is overwhelmingly positive.

That doesn't make sense! I needed to dig deeper.

The Three Things™

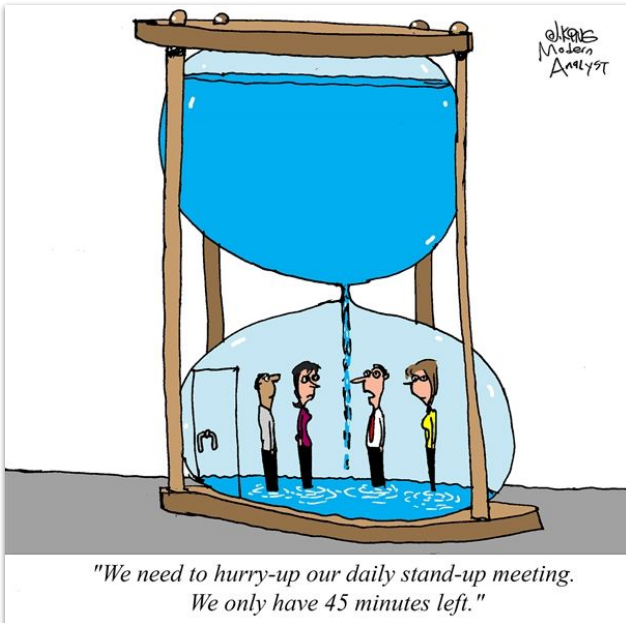
- alright, we're gonna talk about the three things.
- *What do y'all think the three things are going to be?*

Standups. Planning. Retros.

So I did a
thing...



the slide.



Let's talk about standups.

The Three Things: Standups

Ask people to keep hands raised

Who here:

- Can identify with this picture?
- Has been in a standup that:
 - Was scheduled for 50 minutes?
 - Was scheduled for 15 minutes but was actually 50 minutes 60% of the time everytime?
 - Had to hear "let's take this offline?" at least once?
 - Was ridiculously early for some and late for others?
 - Had more than 15 people?
 - Had more than 30 people?
 - Had more than 50 people?
- Let's talk about standups
 - From XP
 - Standing is to make the meeting shorter
 - Short meetings are meant to let devs focus on what they're good at

The Three Things **Standups**

The first ceremony we'll focus on is one we all know and love: the stand up.

History

For those that haven't practiced stand-ups before, the purpose of a standup is to bring a team together to communicate status in a short and light-weight way: what you did yesterday, what you're doing today, and any blockers you might be facing.

As far as I know, its first known appearance was in *Extreme Programming* by Kent Beck and Don Wells. They used stand-ups during their C3 payroll project at Chrysler to minimize the number of meetings developers were involved in throughout the day.[0]

The "standing" in a stand-up is meant to disincentivize long soliloquies about the annals of the project they're working on. [5]

The length of a standup is meant to prevent it from becoming a traditional status meeting that tends to encourage a very large number of inactive participants and waste a lot of time.

Standups

無駄 (*muda*)

JIRA?!?!

lossy

Async

trust

Experiment

Talking Points

RIGHT 1: Standups over Slack?

RIGHT 2: Use your project tracker!

RIGHT 3: Experiment with the format

The Three Things

Standups

Solutions

Stand ups don't have to be this way. Here are some ways you can make your stand up more engaging.

Use your bug tracker! Yes, even Jira can be useful! If everyone makes it a habit to update Jira at the end of their day, then assuming a highly-refined storyboard, you'll already answer the "what I did yesterday" and "what I did today" questions that always come up during standups.

Many bug trackers make it painful to add comments. This is where automation comes in handy. I'm a huge fan of bringing tools into working environments instead of forcing users to use the tool.

Here's an example. Most developers spend the majority of their time in an IDE or terminal of some sort. Pulling them away from that to open Jira is a moderate context switch that's easy to forget.

Jira can pull Git commits from Bitbucket, GitHub, and other source control systems. By configuring Jira to do this, developers can stay in their working environment while providing the rest of the team with the shared context they'd typically wait until stand up to get.

Try having standups on Slack/Teams. Another tactic that helps prevent stand ups from becoming status meetings is by simply moving them online! Try replacing the 15-minute daily standup with a daily standup thread that team members can contribute to *at whatever time works best for them*.

Combining this with better bug tracker hygiene makes it possible to get every member on a team caught up without needing any meetings at all!

Some might lament the loss of face-to-face time by doing this, especially in today's world that is increasingly remote. I think that this "loss" can be reframed as a "gain" in higher-quality face-to-face meetings. For example, the time spent during standups can be reallocated into a really good end-of-week retro or an optional, once-weekly stand down to keep a pulse on team health.

Lastly, you can **try changing the format of your standups**. Many standups are centered around the yesterday/today/blocker format. As we saw earlier, this yields highly-embellished information at best or, at worst, nothing. There are many other ways of running a standup that might produce higher-quality shared context.

<https://funstandups.com/> has lots of really great formats to try.

Remember: status meeting stand ups will always happen when **trust** fades away. Trust your team, and high quality stand ups that will help your team ship software faster will naturally follow.

Example: A colleague of mine ran standups for an extremely large greenfield initiative at a national media company. His standups typically had 50+ people in them. They almost always finished in two minutes and were frequently well-attended. Good standups are not impossible.

Footnotes

[0] <http://www.extremeprogramming.org/rules/standupmeeting.html>

[1] <https://contextkeeper.io/blog/the-real-cost-of-an-interruption-and-context-switching/>

[2] Paul Graham, a software developer turned famed venture capitalist, calls this dynamic the "maker's schedule vs manager's schedule". Learn more here:

<http://www.paulgraham.com/makersschedule.html>

[3] This is sometimes called the "bus factor".

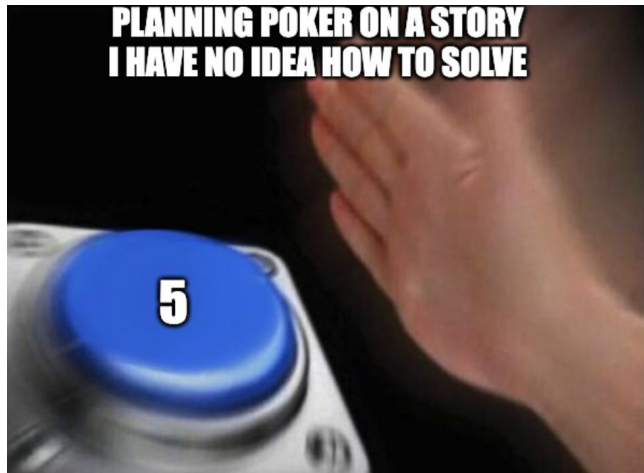
<https://www.5whys.com/articles/team-bus-factors-how-to-reduce-them-and-how-to-prevent-them.html>

[4] Here's a real-life story of a "Brent" in the wild. Toxic employees can cause serious damage to businesses; sharing context can help avoid this.

<https://www.freecodecamp.org/news/we-fired-our-top-talent-best-decision-we-ever-made-4c0a99728fde>

[5] I've seen lots of variations to standing up, though. Wall sits are popular, for example. There's one picture of a team planking during a stand up. Don't try these at home!

"Sprint Planning"



The Three Dislikes

Estimation

Next, let's talk about estimation, my next biggest dislike.

You knew this was coming. I hope you weren't surprised.

History

The history of estimating things probably predates history itself. Instead of summarizing the Wikipedia entry for "estimation," I'll give you a quote from Ron Jeffries, a founder of Extreme Programming and widely regarded as the "father of the story point" [0]:

"In XP, stories were originally estimated in time: the time it would take to implement the story. We quickly went to what we called "Ideal Days", which was informally described as how long it would take a pair to do it if the bastards would just leave you alone. We multiplied Ideal Days by a "load factor" to convert to actual implementation time. Load factor tended to be about three: three real days to get an Ideal Day's work done.

We spoke of our estimates in days, usually leaving "ideal" out. The result was that our stakeholders were often confused by how it could keep taking three days to get a

day's work done, or, looking at the other side of the coin, why we couldn't do 50 "days" of work in three weeks.

So, as I recall it, we started calling our "ideal days" just "points". So a story would be estimated at three points, which meant it would take about nine days to complete. And we really only used the points to decide how much work to take into an iteration anyway, so if we said it was about 20 points, no one really objected."

[0] The rest of this post is a really good read.

<https://ronjeffries.com/articles/019-01ff/story-points/Index.html>

DevOps and Agile: The Three Things

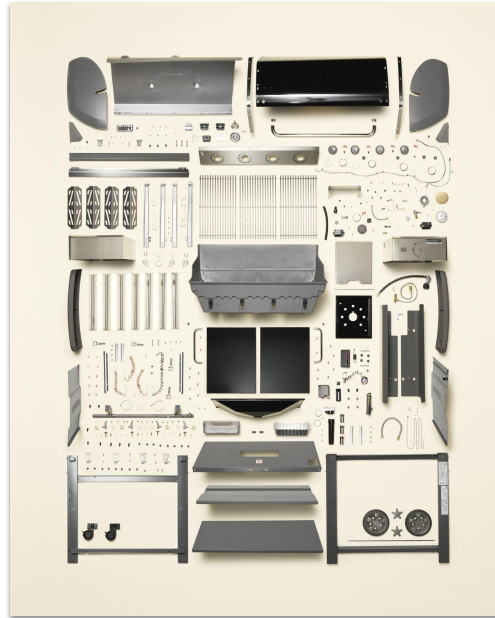
TOP MIDDLE: Rituals

LEFT 1: They're long

LEFT 2: They turn into status meetings

LEFT 3: Words don't mean what they used to mean

- Retros turn into end-of-week status meetings instead of a relaxing end to the week
- Acceptance Criteria become a checklist instead of an agreed upon definition of done
- User Stories become ticket descriptions instead of *stories*



The Three Dislikes

A Thought Experiment

My dislike for estimation can be explained with a quick thought experiment.

Exercise

You get a call from your friend one day. "Hey, dude. We just moved into town, and need some help moving in. Can you stop by if you're not busy?"

You excitedly agree to help them out the next day...only to walk into this.

How long would it take you to put this together?

Think this through. There's no wrong answer!

Solution

There is a nearly infinite set of possibilities for attacking this problem.

Perhaps you asked yourself a few follow-up questions, like:

- "What is this?"
- "What tools do I have?"

- "Where is it going, and how much space do I have to work with there?"
- "Will this friend be helping?"
- "Do we need every part that's here?"
- "Why didn't you tell me this thing came with, like, 10 million parts?!"

All of which are totally valid...your friend simply asked you to help them build a grill!
Without telling you!

You might have built a grill just before reading this and have a very accurate idea of how long it will take to assemble it.

Maybe you've never seen a grill before, and because you're not very handy, you're imagining this taking a day or so to get done.

Planning

"refinement"
too f'n big



The Three Dislikes

Planning

Back to Reality

This is similar to patterns I've seen in so many planning sessions.

Problems

Refining During Planning

I've been to many planning meetings where "planning poker" is less "agreeing to a story point that won't blow out the sprint and make someone with a spreadsheet real mad" and more "oh, this user story has a single sentence in it..."

Refining stories during planning is *literally the worst*.

First, the person that threw the story into the backlog usually isn't in the planning meeting! This not only risks adding more *muda* into the backlog that will stale over time, but the team is now at-risk for potentially not working in a story that might be considered business-critical!

While this might seem like a common sense suggestion, I've been to an extremely surprisingly high number of planning sessions where this happens.

Planning

"refinement"

smaller!

too f'n big

async

~~man hours~~

person hours?

The Three Dislikes

Solutions

Here are some tactics that I've seen work at high-performing teams!

Yes, your stories can be smaller!

Engineers and/or interested stakeholders Hoovering entire planning meetings over a very small subset of stories happens all of the time.

In almost every instance I've seen this happen, a big driver of the conflict was the story simply being too large or ambiguous!

We as Agilists know that the user story is meant to be the smallest unit of work a team member can accomplish. That is unlikely to happen if a story has ten acceptance criteria, for example!

I recommend spending a lot of effort making stories as small as possible. Knowing when stories can be made smaller while still being playable is a difficult task, but there are usually a few warning signs you can look for, like:

- Does the user story have multiple stories in it? (This might sound obvious, but I've seen so many stories like this!)
- Do multiple engineers not understand the story? Why?

- Are multiple engineers throwing a five (or higher) against that story during planning? Why?
- Is the list of acceptance criteria ten miles long, even after de-duplicating them?

Continuous refinement

What if I told you that your backlog refinement sessions could be less than 15 minutes?

I've seen it happen!

In my experience, the trick to getting there is with continuous refinement. "Continuous refinement" is the practice of refining stories as they enter the backlog, i.e. validating their user story bodies and acceptance criteria and, if possible, prioritizing them as they come in.

Trying to continuously refine stories manually would work for a good five minutes before losing your sanity. Thus, trying continuous refinement forces teams to (a) be very rigid about the stories they accept, and/or (b) automate as much of the refinement process as possible.

Many project trackers either have facilities for ticket automation (like automatically assigning labels or closing stories that don't pass validation rules) or have APIs that you can take advantage of to build it yourself.

To be clear, I am **not** recommending canceling your planning and weekly refinement meetings. I think there is a lot of value in gathering together to make sure that the backlog looks right and that the team has a theme they are working towards. All I'm saying is that automating the *muda* away before it even makes it into your story board makes the meetings an impactful formality rather than a useless necessity.

Maybe use person hours after all?

If story points/T-shirt sizes/automobile types/whatever are working well for your team, by all means, continue using them!

Unfortunately, in almost every team I've worked with, adopting story points or other non-time-based estimation units have caused more problems than they solve.

The stress of constant negotiation and re-negotiation to chase inaccurate velocities that were silently converted into time anyway appeals to absolutely no one (unless you like the drama of stress, of course).

Why should we continue stressing ourselves out for unrealistic goals? If story points are destined to be inaccurate and poorly correlate to the tasks at hand, then what's

the harm in going back to estimating in ~~man~~ person hours?

Hear me out here.

You're on a team that's working towards a critical release date a month from now. The release date was set by executives to try and beat out the competition, whose releasing their thing a few days later.

The team knows (or *feels*) that their backlog is at least three months deep.

Does expressing everything as story points change this?

Sure, if your team was given enough executive sponsorship to be truthful about their estimates. But that's an unlikely reality.

If we know that this is unlikely to happen, then would it not be easier and more useful for team members to think in terms of how long it actually takes them to finish work?

Sure, your backlog will be represented as being three months deep instead of 72 story points deep. Sure, senior management will put a lot of pressure on you to magically turn that three into a one. *But you were going to have those conversations anyway because estimations are just that, estimates.* So if all estimates are just best-guesses, why not just discuss estimates in hours?



Retros.

The Three Things **Retros**

Let's wrap up this presentation with my favorite most-abused ceremony that DevOps practitioners typically grow to hate: the retrospective.

History

The Agile Alliance (yes, that Agile Alliance) defines retrospectives as:

definition

If you're thinking "Well, that sounds like something I've been doing with teams for my entire career," you're not wrong! The idea of a team gathering together to do a self health check has been a thing for a very long time.

Unfortunately, those self health checks are typically sporadic and long overdue. In my experience, before retrospectives were a "thing", they were chatter during the end-of-year holiday party or around the proverbial watercooler. While these feedback sessions are extremely healthy for keeping team morale high, because of their sporadic nature by which they happen, teams often did not use them to build themselves better.

What the concept of a retrospective aims to do differently is best described by Alistair

Cockburn, a computer scientist often proclaimed as the "father of the retrospective."
From *Surviving Object-Oriented Projects: A Manager's Guide*:

Lesson 4: Get training; keep learning. Not only did Project Ingrid get initial training, they established internal study meetings. Continuous learning is a key factor in building a successful development organization. Even the masters of object technology continually meet to trade ideas and learn. One week, even four weeks, of training does not constitute "enough learning."

The retrospective aims to move these disconnected and infrequent "venting sessions" into a formal, weekly practice (or after every iteration). Their goal is to give the team a safe space to discuss how they felt about the iteration, the team, and the environment they're working in. Output from the retro is meant to be captured as action items that the team can, optionally, track alongside the rest of their work (or by some other means).

Retros

inaction items

scenery

~~safe space~~

continuous

Vegas

The Three Things

Retros

Fortunately, I've seen just as many bad retros as good ones. You'll learn (or are probably learning about) all sorts of ways of having good retros, so I'll only talk about some traits of successful retros that I've seen in DevOps circles.

Change the scenery!

You'll learn (or have learned) that retros should be *the* meeting that your team looks forward to every week. It's difficult to make that happen when the team has a drab meeting room to look forward to.

Many teams have enjoyed significantly better retros simply by changing the scenery a bit.

I'm a huge fan of having retros during lunch hour with nothing but the team and a notepad. Food brings people together!

Make it continuous

You might not be able to convince the powers that be to expense a weekly lunch for the team to hold their retro. If that's not possible, then another tactic that teams respond well to is holding fewer, but higher-quality retros instead.

Here's one way this could work. The team has a shared "wall" of post-its: physical or virtual. Whenever someone on the team has a grievance that they want to talk to the team about, they create a "sticky" and paste it to the board. Sometime after the middle of the iteration, the scrum master or team lead looks at the board and, if any post-its are on the board, they ask the team if they want to hold a retro at iteration end or sooner.

The team won't always agree to have a retro --- some iterations are pretty uneventful! However, when they do, the chances of it being an impactful, high-quality retro with real action items to address is very high.

This also helps prevent retros that center around one or two big grievances, which can shut important voices out. [0]

What happens at retro, stays at retro [1]

The retro should be a completely safe space. The team members should feel empowered to say *anything* that helps drive the team forward.

I've been to some retrospectives where people even addressed personnel conflicts within the meeting! (This is obviously very risky and requires a high degree of mutual trust; proceed with caution!)

This cannot happen when the team feels like their job is done for if they say the wrong thing. This is more likely to happen when the audience within the retro grows beyond the team, and it *definitely* happens when management shows up.

I highly recommend that team retros have no management or outsider presence. In other words, retros should be "Vegas" style: *whatever happens within the retro, stays within the retro.**

You might need to hold two separate retros for this to be successful: one for the team, and another for management. This way, you can safely gather as much useful feedback about both how the team is doing and how the team thinks their management is doing as possible.

Footnotes

[0] I often see retro facilitators (who should not always be the scrum master!) try to timebox retro talking points on a board.

I'm generally not a fan of this for two reasons.

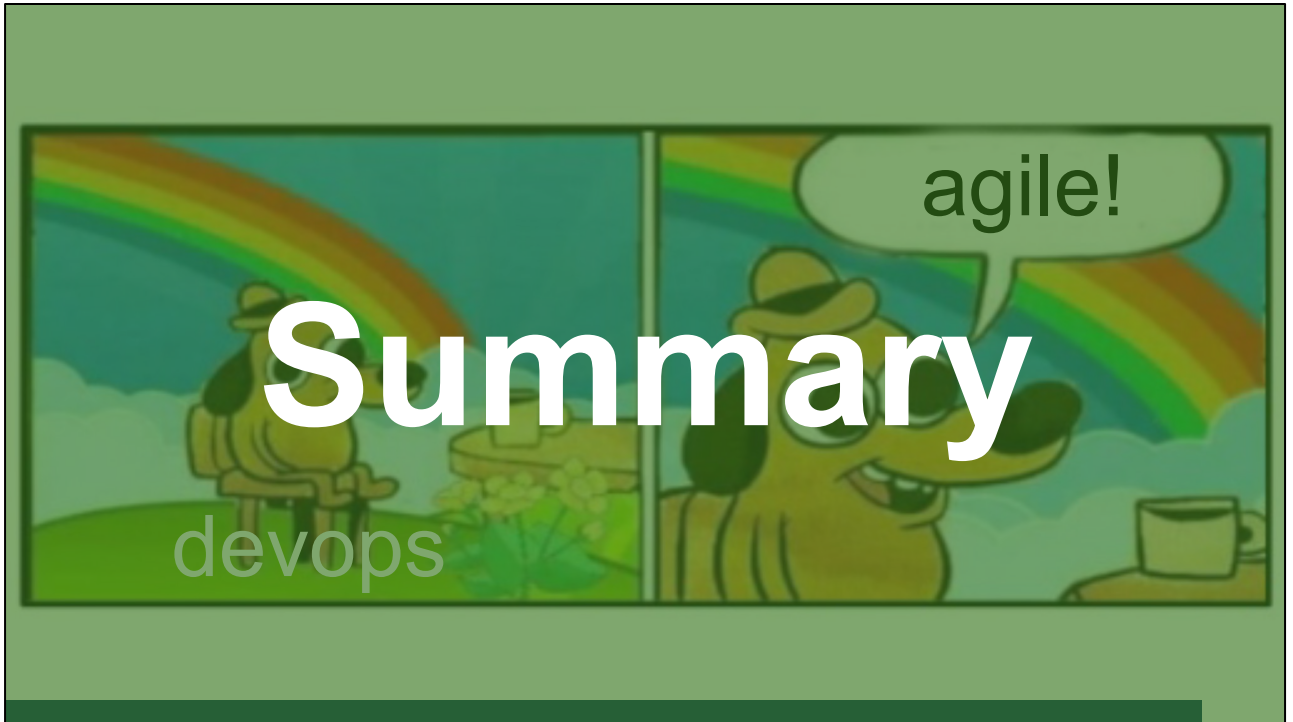
First, some of the more serious talking points often go way beyond the timebox

anyway, and I think there's value in centering the retro around one issue, if that's what the team wants to do.

Second, the timebox being required is often a sign of a team that has a LOT to talk about, which is a symptom of a team that doesn't talk to each other often.

If this happens frequently, it might be worthwhile to understand why the team has so many issues and how they can be brought to bear sooner!

[1] Unless it's something that the team wants to make an action item. Also, the retro facilitator can, and should, be an outsider without bias!



SUMMARY

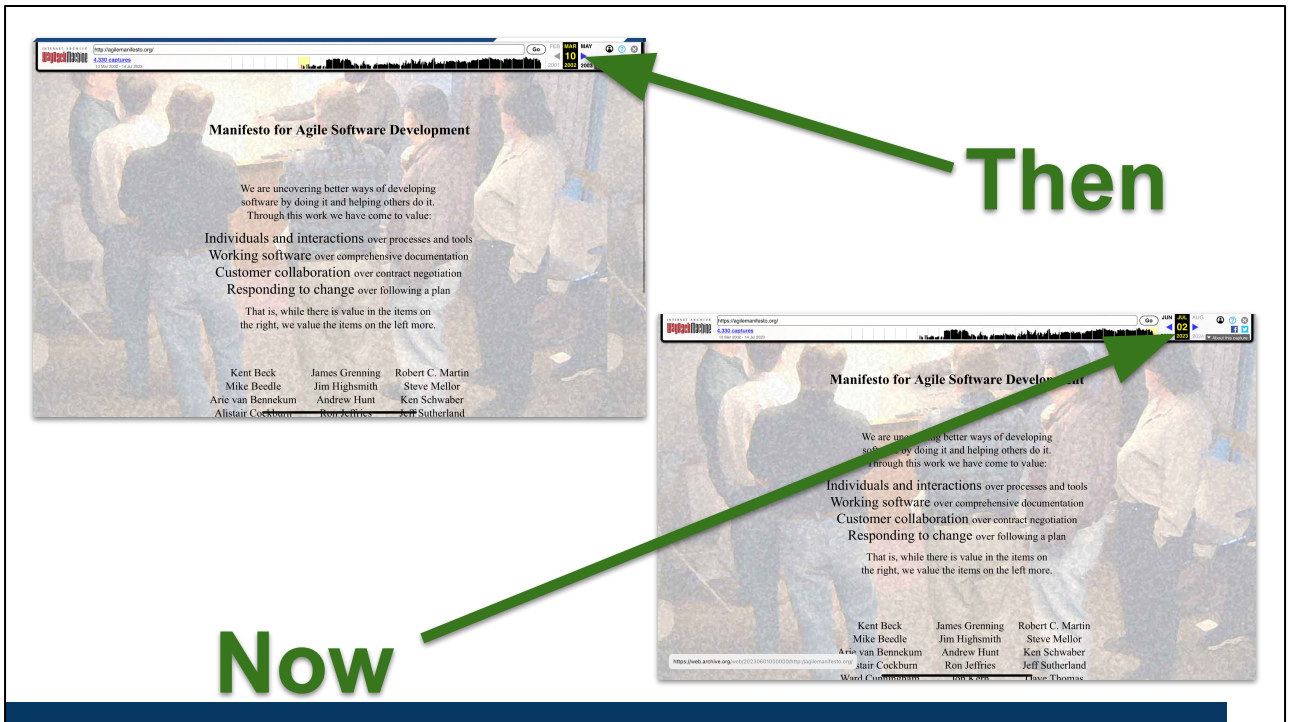
Thanks for reading!

In this presentation, we delved into the relationship between DevOps and Agile.

We started with a brief primer of DevOps: how it started, why it started, and where it's going.

We then used some spicy screenshots from the Internet to posit a controversial question: does DevOps hate Agile?

From there, we discussed three popular ceremonies that drive this viewpoint, how they do so, and what Agilistas can do to buck the trend: standups, planning, and retros.



Final Word

I always like to throw in the original Agile Manifesto into these kinds of talks somewhere.

I do so to remind teams that while there are so many "official" ways of doing capital-A-Agile (Scrum™, SaFE™, Leading Agile, DaD, etc), **this** is what agile ultimately is.

Agile, like DevOps, is about working together more closely and more often to achieve a common goal. That's it!

You don't **have** to do standups.

You don't **have** to do planning.

You don't even **have** to do retros!

While these ceremonies are helpful for getting teams talking to each other and working together, don't forget that talking to each other and working together is the end goal, not the ceremonies themselves!

This was the goal in 2001, and it's still the goal today!

Thanks!

dev@carlosnunez.me

Thanks for reading this presentation!

I'd love to hear your feedback. What did you like? What did I miss?

Email me anytime at dev@carlosnunez.me or find me on LinkedIn at <https://linkedin.carlosnunez.me>.

Take care!