Software development, procurement, & management fundamentals
Software development practices

Part 4 of 5

Presented by 18F for:
Office of Childcare, HHS

August, 2022

Hello.
Welcome to software development practices.
My name is Greg Walker and I'm with 18F.

This was created for the Office of Childcare at the Department of Health and Human Services.

https://pixnio.com/objects/metal-gears-mechanism-technical-metal-screws-gear-utility

This presentation is part 4 of The Software development, procurement, & management course, a 5-part series that covers:

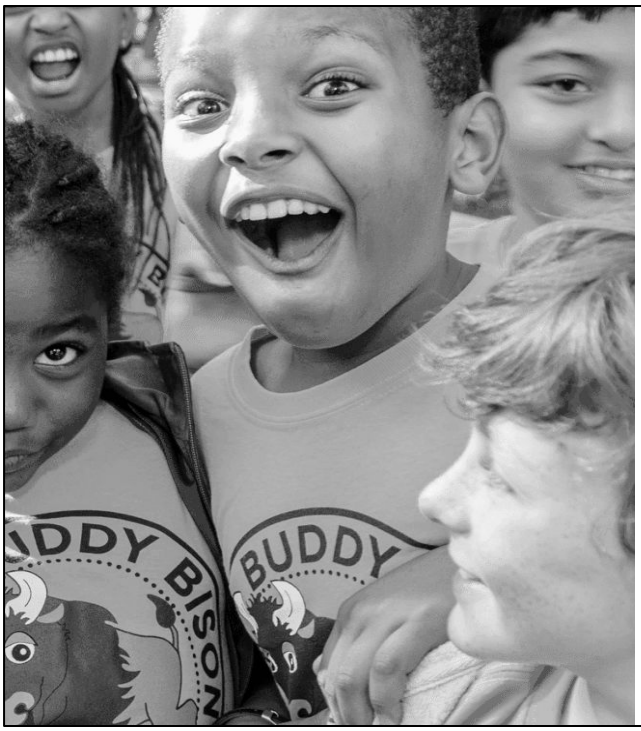
- Agile management
- Product ownership
- User-centered design
- Software development practices
- Agile contracting

What is 18F?

18F is a technology and design consultancy for the U.S. Government, inside the government.

18F is a technology and design consultancy for the U.S. Government, inside the government.

We are **federal employees** who teaches partners how they can improve the user experience of government services by helping them build and buy technology.



We share the same motivations as you: delivering great service to the public.

Greg Walker

Experience

- US Army Corps of Engineers, Research and Development Center: Hurricane and flooding disaster response and recovery
- 18F: State child welfare IT modernization (sorta!)
  - Lots of acquisition
- 18F: State Medicaid IT modernization
  - Lots of acquisition
- 18F: Federal Medicaid IT oversight policy
  - How to make acquisition easier and more successful for state/tribal/territorial partners

4/ Software development practices

18F

We’re mostly going to be talking about a specific set of software development philosophies called DevSecOps, because that pretty well encompasses what most high-functioning, productive, and healthy teams are doing today.

Breaking down silos and enabling continuous, iterative delivery

DevSecOps combines software development, security, and operations into one team, and automates as many of their processes as possible. People focus on the hard problems and let the computers handle the repetitive stuff.

In the agile management presentation, Lindsay talks about cross-functional teams earlier, devsecops is an enabler for that. Automating a bunch of stuff frees up the team members to work together more closely on the hard problems. Use the whole team's brain. The whole team together will always do better work than each person working alone.

And computers are exceptionally good at doing the same thing over and over, so why not let them? People aren't really that great at it - we get bored, we get distracted and forget a step, we forget to do the whole thing.

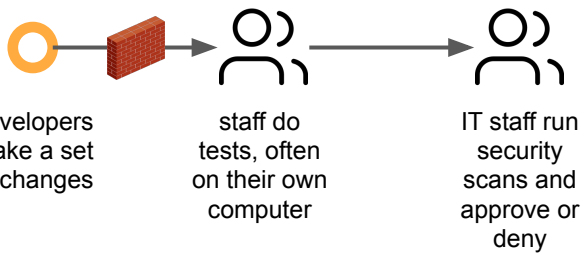
This slide also says devsecops enables continuous, iterative delivery, so let's take a look at what that means, too.

A traditional delivery process

developers
make a set
of changes

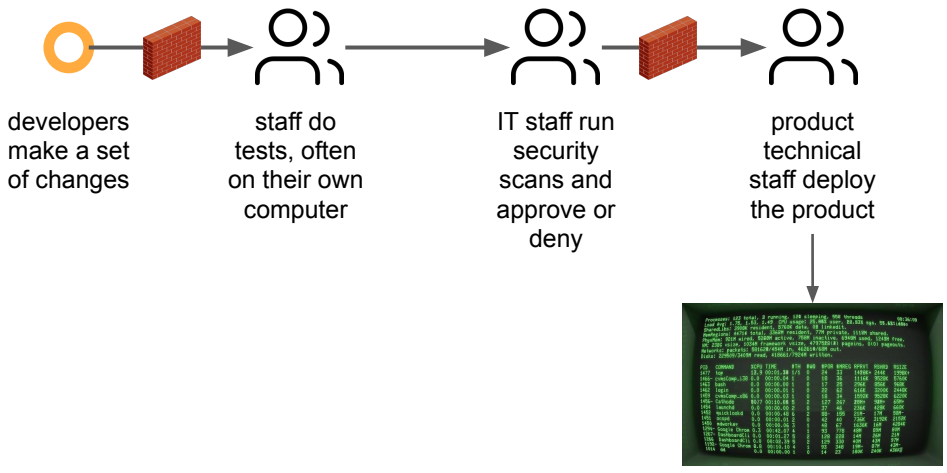
Okay, so a quick deviation to what a traditional, manual deployment looks like. First a change happens. Or probably a bunch of changes, really. It could be several months worth of work, or sometimes even years.

A traditional delivery process



Then the product gets thrown over a wall for testing. This can be a group of staff who do tests by walking through a testing plan or reading change notes to figure out what to test. It can also include IT staff who run security scans or do compliance checks. Depending on just how big the changes were, this step can take a while. When I worked on my last waterfall project, we set aside 3 months for this part.

A traditional delivery process



If everything in that process checks out, they throw it over another wall to technical staff who go about deploying the product. And they do that by typing in all the necessary commands in the right order. And this is another pretty complicated set of things that you have to do exactly right. If you mess up you could bring the whole product down for a while. Or you could do something really bad like deleting a whole database, which has definitely never happened to me more than once. So this is a really hair-raising part of the process. Honestly, doing this manual deployment is terrifying. I hate it.

But anyway, this is traditionally how it's all done. There are walls between the different people involved, there's not much collaboration to keep things moving smoothly, and there are these big delays kind of baked into the process. Nothing happens quickly.

Continuous delivery

Something changes...

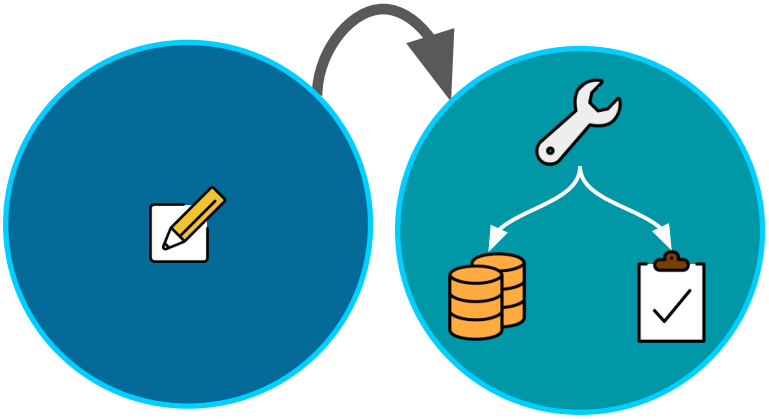
- Code changes
- Updates to text

So what does it look like

Continuous delivery begins with a change. There are lots of changes that could prompt a delivery: a developer changes code, a program SME updates some text to be easier for users to understand. And when those happen, we want to be able to test those out with users, so we need to deliver, so that triggers the delivery process.

“Midnight” kind of stands out. I put it here just to drive home that there really are lots of things that can be considered a change. Some products really do kick off this process on a regular time interval in addition to the other changes. But the important thing is just that you have a lot of flexibility to decide what starts the delivery process.

Continuous delivery



Something changes...

...which is a signal for automated testing to start...

- Functional testing of business logic
- Accessibility testing
- Code quality testing
- Security scanning

12

Once that change happens, automated tests kick off. There are a lot of different things you can test. For example, you can run functional tests that ensure the changes didn't break your business logic, or at least let you know so you don't send it out to users.

You can do accessibility testing to help make sure you're not leaving anyone out. Note that automated accessibility tests can only cover about at most 40% of what you need to test, so you'll still have to do manual testing, but the 40% you can automate is work you don't have to do by hand!

You can test the quality of the code itself, which is really valuable for making sure you are getting good, maintainable code. You can check for consistent, good practices and get a warning if the code is becoming messy. It's much easier to fix when it first pops up than when it has had time to accumulate for a while!

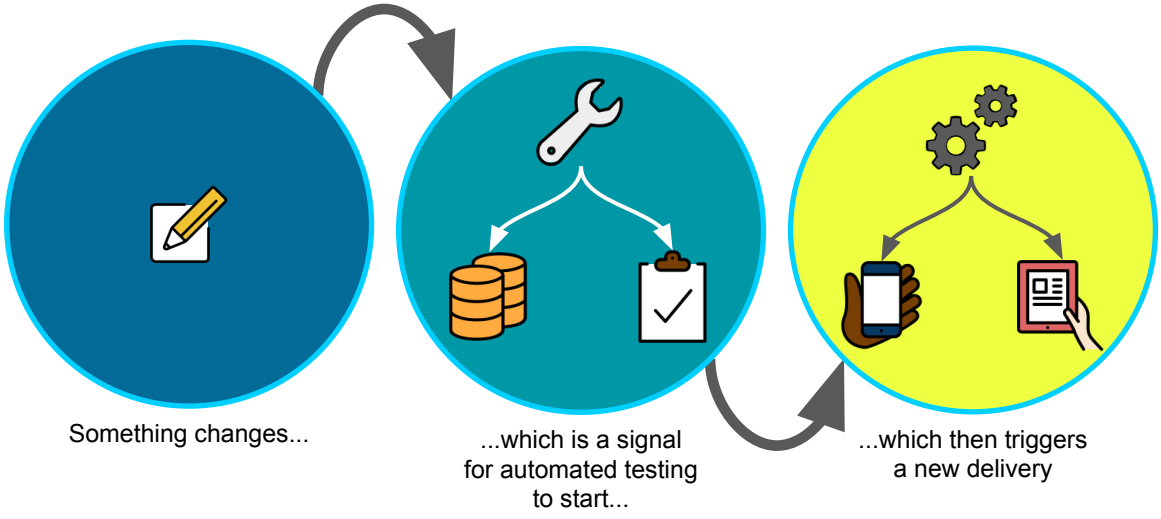
And you can do security scanning. This one is really huge, and it has a bunch of sub-categories of testing itself: static tests, dynamic tests, package management, and more. But the important thing is that having these security checks baked into your process from the beginning will make your life much easier down the road than trying to do all your security work at the end.

I also want to talk about this automated testing can be a catalyst for un-siloing your team. The biggest one that stands out in my mind is actually the security scanning. Like Lindsay mentions in the agile management presentation, include your security team from the beginning. They can help you understand what kind of security testing

is helpful for managing your risks and what will be helpful for any compliance requirements down the road. Working together, you can incorporate those tests into your continuous delivery process and that increases confidence for everyone that your product is more likely to be secure and more likely to comply with your – or even your federal partners' – security requirements.

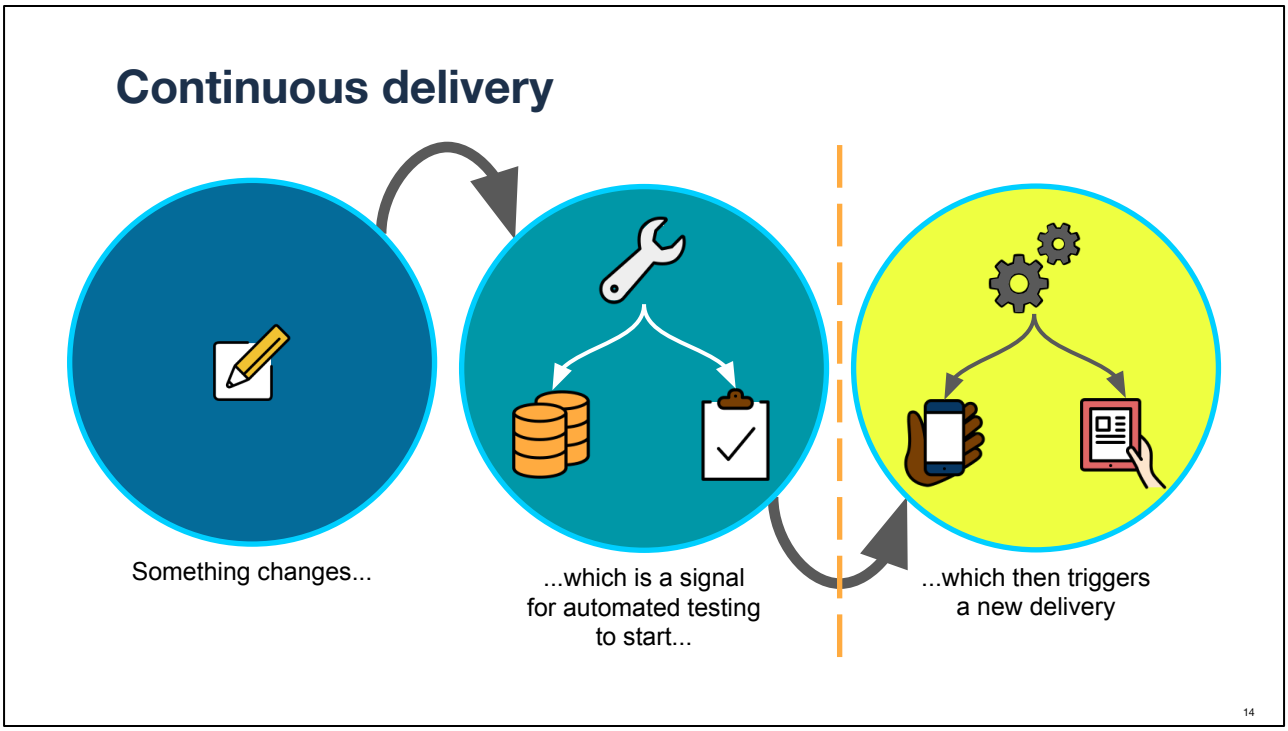
These automated tests are also empowering for the product owner because they should provide a quick pass/fail view of various parts of your product at any given time. Being able to glance at the results of your tests can give you information that will help you have constructive conversations with your team, and it'll help you make sure you're delivering the quality you want. It's not your job as a product owner to interpret the test results to figure out how to address them, but being able to see that issues exist lets you know who to lean on.

Continuous delivery



13

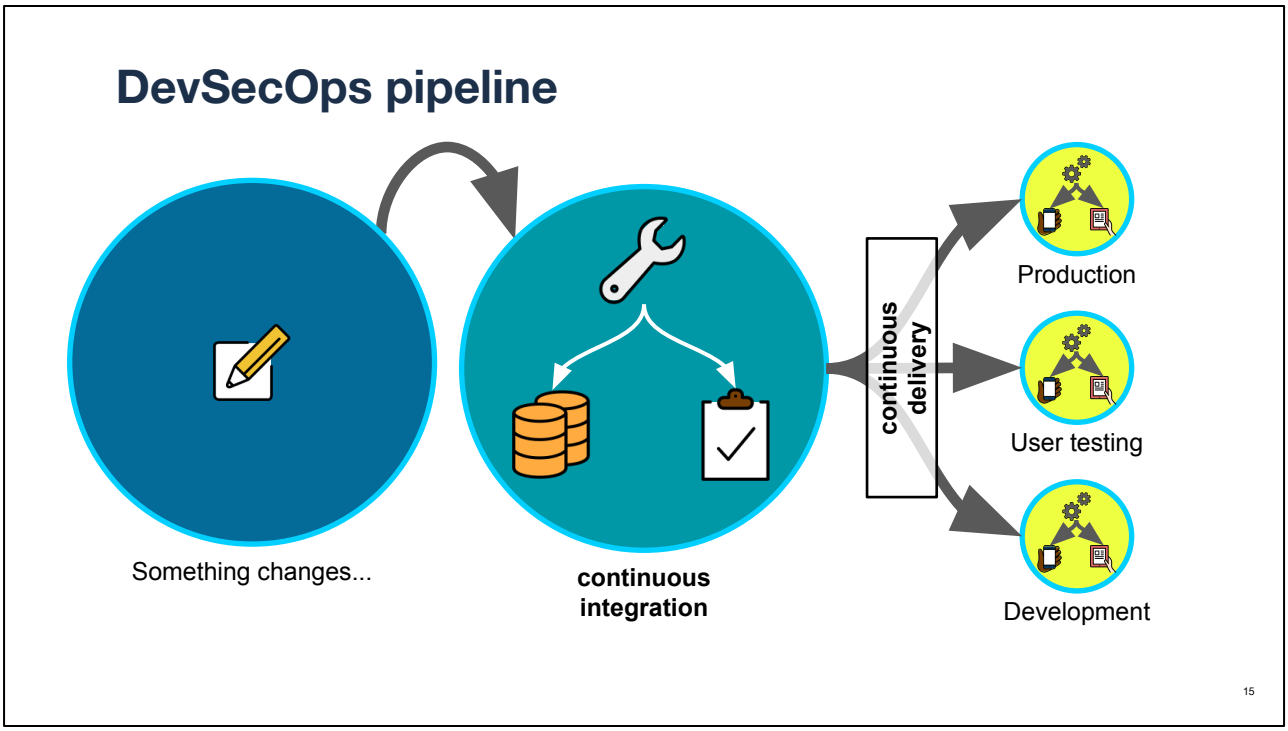
Once all the testing is done, it’s time to deliver!



But one thing that's not in the slides is this manual review step. Some of the things you need to check can't be automated. Like if you've changed some text, a human will need to check that it's in plain language and that it conveys the right information. Or remember that only about 40% of accessibility testing can be automated. If there's a new feature, you may want to have a human do manually accessibility testing on that feature. And you can also review the code to make sure that it's logical. Just because the code is consistent and uses good practices doesn't mean that it makes sense. So having people doing some manual review is also a normal part of this process.

Adding that manual step might sound like it introduces delays that the automation was meant to eliminate, but because this process is happening for every change, the reviews are usually very small. You're just proofreading a few sentences, for example. Or you just need to make sure this one new button shows up correctly in a screen reader. Having smaller, more frequent changes means it's easier to check everything and it also means that manual reviews should be pretty quick.

Once all the tests and manual reviews are finished and everything passes, now it's really time for a delivery, and that can mean several things. Maybe it means your website gets updated, or maybe something happens to your database. If you were building a mobile app, maybe it would mean the new version gets published to the app store. This process is also automated, just like the tests, so once you've got the delivery process working, it'll keep on working.



One last thing to mention, your automated deployments can go to different places. You're probably familiar with production and staging or development. In a devops world, those are called deployment environments, and you can conditionally deliver to many of them. For example, maybe every single change goes to the development environment, but you don't deliver to user testing until a whole new feature or related set of features is ready for user feedback. And then you might hold a production delivery until the end of a sprint or something similar. The important bit is that you don't have to deliver every change directly to end users because devops embraces multiple environments and only delivering to them as appropriate.

How does all this help?

Continuous delivery is making the latest changes available as often as possible

Continuous delivery is about frequently delivering small pieces of your product. This enables continuous user research and feedback to steer the product direction.

Continuous delivery is the engine that powers quick, iterative development. If you're not able to get changes in front of users, you can't test those changes to make sure they're meeting user needs or find out how they're missing. Delivering small changes frequently is what enables you to know if you're on the right track and help you figure out the problem you need to solve.

Continuous integration is testing everything, constantly, to catch problems early

Continuous integration should increase your confidence that the product is working as expected and not regressing unexpectedly. The regular, integrated security scans reduce your risk of a security incident and may reduce your compliance burden.

19

Continuous integration, meanwhile, is about making sure the thing you're delivering does what you expect it to do. So this piece is more about satisfying YOUR needs rather than your users'. It's a chance to enshrine business requirements right into the process, and reducing the risk of things going off the rails. Testing code quality reduces long-term costs and prevents vendor lock-in; security scanning help prevent incidents and streamline compliance requirements; testing business logic increases confidence that everything works as expected.

And in the agile management presentation, Lindsay talks about a good sign that your agile processes are going well being that you catch problems early. Continuous integration can help you do that.

This is especially important for big systems with lots of pieces. It's a normal practice to develop those pieces separately, maybe even by separate teams if you're working in parallel. I wouldn't recommend *starting* that way because it's good to start small to get your bearings and build good process habits, but at some point, it's normal to stand up second, third, and so on teams to work on pieces. Healthcare.gov did that, and that might not sound like an endorsement. We all know how that went, but the thing is… one piece failed and that cascaded throughout the system. They were seeing problems at the ends and had to walk backwards through the whole system to try to find the original problems, and the team didn't find it until they had already delivered it… because they weren't doing continuous integration. They had never put all the pieces together and tested them. If they had been doing this continuous integration testing from the very beginning, when all the pieces were small and fairly simple, they

would have been much more likely to catch these issues before delivering the product. Lesson learned – they do continuous integration now.

Automation is doing the same thing, the same way, every time

Automation reduces the risk of a mistake bringing whole systems down by ensuring that the delivery process is consistent.

And automation reduces the risk of any of those frequently-run processes going astray because someone made a mistake. People aren't particularly good at doing the exact same thing the exact same way over and over. We get tired. We forget. We get distracted. And for things like deploying a product, there are so many things that can go wrong. So we automate it – we figure it out once and then we let the computers do it themselves after that. Computers are great at repetition!

And automation also speeds things up. If you think about more traditional approaches where you take a whole bunch of changes and you hand them off to your QA team and then do user-acceptance testing. How long does your QA team need to check everything out? It depends on how busy they are. We know that's often not their whole job, they have other things to do. It could be a few days before they can get to it, or a few weeks. But with these automations, you can do most of that work in minutes and remove that burden from staff. Without automations, we really couldn't do continuous user research and feedback.

How do we do it?

Someone on your team needs to be fluent in software development practices

All of the automation in the world is meaningless if you don't understand the outputs. To review code, interpret tests, and provide feedback on technical direction, you need a tech lead. Without them, you won't know when you're off course or how to correct it.

In the agile contracting presentation, Randy talks about the QASP and that is a great tool, but it won't be very useful to you if you don't have anyone who can evaluate your product against its metrics. That's one of the ways a tech lead will help you. They can review code to make sure it's solving the right problem in the right ways. They can look at tests – and this is important because it is actually very easy to create tests that don't do anything but will make your reports look great, and that's pointless and risky. A tech lead can ensure your tests actually accomplish what they're supposed to. Your tech lead can also be your advisor on technical direction, helping set the product up for long term success.

Perhaps most importantly, having a good tech lead in place will help you see when things are going off course, and they can do it sooner than you'd do it otherwise. They'll also be able to help steer the product back on track. In this way, having a tech lead is a huge risk reduction.

And I want to acknowledge that this is a huge, huge request. Hiring is hard in the best of times. We've experienced it ourselves hiring at 18F, and we've seen and heard from our state partners in the past how much harder it can be for them with even more constrained resources. That said, we've also seen this role be the difference between success and not.

This role could be filled by a state employee or a contractor to the state, but if they are a contractor, they should not be employed by any of the contractors working on your product, to avoid conflicts of interest.

What makes a good tech lead

The tech lead understands software, product management, user experience, agile, and user-centered design. They are not necessarily experts at any of it but they know enough to identify “good” and “bad.”

25

The tech lead isn’t doing all of this work. They’re more of an oversight and advisor role. In the product management presentation, Sheel talks about product ownership sitting at the intersection of user needs, stakeholders, technology, and business. The tech lead is the person who can help inform the product owner about what’s technically possible and be their advisor on the technical strategy and direction of the product to help them make better decisions.

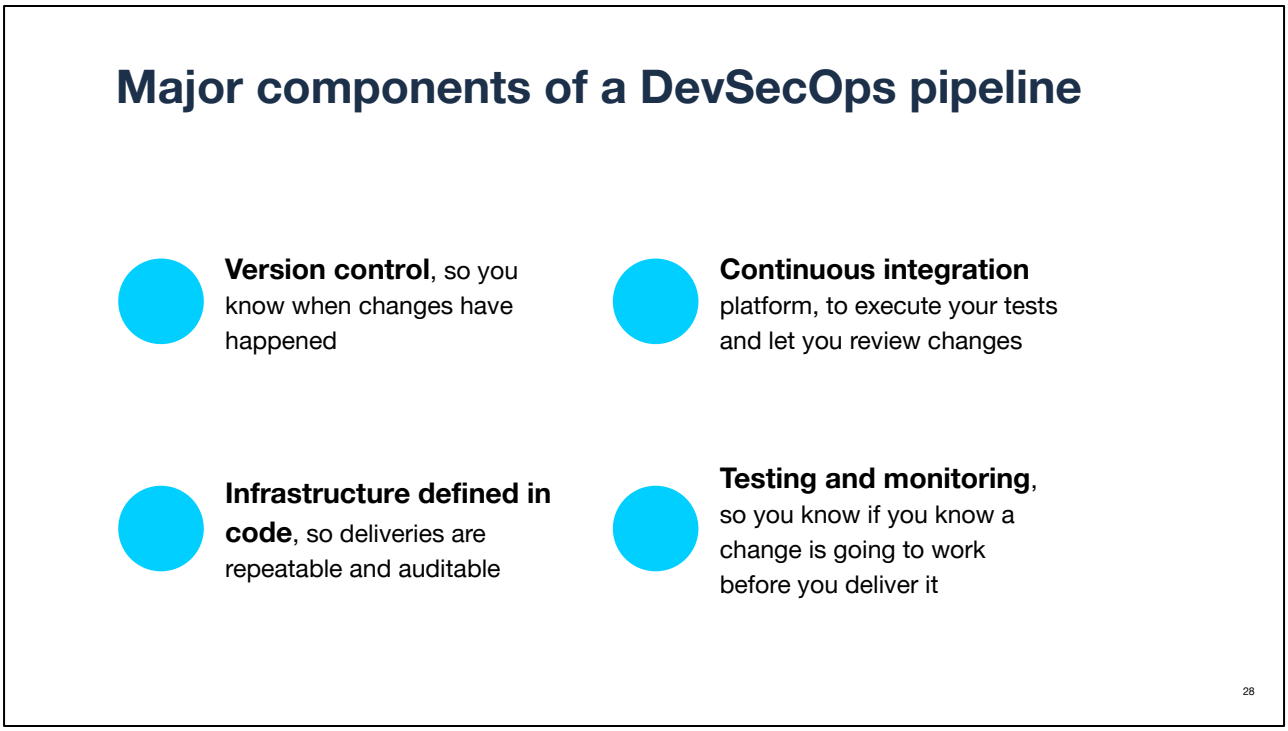
A good analogy for this role is a construction foreperson - you don’t want them laying tile but they can tell you if the foundation is sound. They’re the person who oversees and coordinates the client and all the other experts doing the work. They aren’t a plumber or an electrician, but they know enough about both fields to guide a client towards making good choices about how to approach their plumbing and electrical problems and needs and can tell when anything is going off-course. They help a client understand, fundamentally, *what’s possible* and they have a big-picture understanding of the overall project. Their job is to connect their big-picture understanding to the specific subject matter expertise that the subcontractors bring in, and to make sure that the subject matter experts are delivering properly. Their job is mostly about communication, to reduce the imbalance of expertise often found in

government/vendor relations.

Build your DevSecOps pipeline at the start and insist that your teams use it

The sooner you have a pipeline, the sooner you can have small, frequent deployments, which makes it easier to make changes and helps the project move ahead smoothly. In healthy software development projects, the DevSecOps pipeline is often the very first thing that gets built.

Building the pipeline early is also easier. Retrofitting it into an existing project can be a challenge. And doing it early lets you benefit right from the beginning, and have a sort of compounding benefit.



Whether you're building a devops pipeline or talking to a vendor about it, there are some common major components that should always show up.

The first one is version control, and that's basically a system for keeping track of changes to your product over time. You've probably worked on documents before where it's named like report.doc, and then report_final.doc and then report_final_1.doc and so on. That's kind of a form of version control. It's not a very friendly one, but it serves a similar purpose. For software projects, we use version control tools that track all of our changes without having to keep multiple copies of a file. These tools let us see when changes happened and who made them. They let us go back in time if we need to revert something, too. But the most important property for devops is that they create events when a change happens, and we can use that to kick off all the pipeline.

Continuous integration is one we've already talked about. It's the part where your tests and reviews happen.

Infrastructure defined as code is not always there, but you should push for it, especially if you're working in a cloud environment. Today we mostly work with virtual computers. So the server you're delivering your product to probably isn't a single physical server. It's a logical, virtual server. But because of that, we can kind of write its specifications into a file and use that specification to quickly create more virtual servers from it. That's basically what infrastructure as code is, and it lets you very quickly and easily create whole delivery environments in a way that is repeatable and

auditable. That auditing is a huge boon for compliance and security purposes, too. Once you've nailed it down, you can just do it the same way again.

Testing and monitoring help you know that changes won't break your product or will let you know if anything does go wrong. Devops practices should help increase your uptime this way because tests are catching problems before they get to users and monitoring is watching your product to make sure it's still working. If anything goes wrong, good devops practices will help you see it sooner.

Key takeaway:

DevOps enables continuous, iterative development

18F

DevOps does not create continuous improvement, but it creates an environment where it's easier and more possible. Without the constant tests, it's harder to be sure you're not breaking your product. Without the constant deploys, it's harder to get good user testing and feedback. And without the automation, you're spending a lot of human time doing things other than building a product, which slows the whole process down. Ultimately, devops is a tool to help enable everything else we've talked about: agile practices, product management, user-centered design, and it is also an enabler for agile contracting, which Randy talks about in the agile contracting presentation.

I hope you found this useful, and I am happy to answer any questions you may have in the Q&A sessions.

Agile principles https://agile.18f.gov/
Change control boards: https://18f.gsa.gov/2021/03/02/using-agile-and-devops-to-get-better-results-than-a-change-control-board/

Image https://pxhere.com/en/photo/92830