How can you accelerate CI to speed up build and test times and improve developer productivity?

Last Tuesday I participated in an online panel on the subject of CI Acceleration, as part of Continuous Discussions (#c9d9), a series of community panels about Agile, Continuous Delivery and DevOps. Watch a recording of the panel:

<iframe width="560" height="315" src="https://www.youtube.com/embed/_q7Y708j-XU" frameborder="0" allowfullscreen></iframe>

Continuous Discussions is a community initiative by Electric Cloud, which powers Continuous Delivery at businesses like SpaceX, Cisco, GE and E*TRADE by automating their build, test and deployment processes.

Below are a few insights from my contribution to the panel:

How Do You Define Continuous Integration?

Fail early, fail fast, fast as you can. You want to get your cycle time, in order to measure that, you have to know whether you're producing something valuable. Continuous Integration - you can have all these unit tests, but until you actually mix it together with something that's somewhat real, larger sets of systems, you really don't know whether it's going to work as a whole.

And that's really the key there, you can't do any of that without automation, so even though Continuous Integration is a philosophy, you really have to leverage whatever tools you can make available, to make that a reality. Trying to continuously integrate is a great theory, but we just make too many mistakes. Without actually having done something wrong necessarily on purpose, whereas, you can automate the whole process, and make it happen over and over again - that's how you can ensure that you're producing what you think you're producing.

Clearly, it should be a cultural thing, where you're actually trying to check before you commit things in to the main line, but another pattern is everybody using short lived branches - your CM system supports easy to use branching, and then you use your CI automation tool of choice to go and actually do auto-merges across various branches, especially against the main line, and do those permutations that would be annoying for the average developer to do, but to catch things that wouldn't be caught if people weren’t sharing their work in a place where everybody could easily get grasp at. It's a good indicator that "Hey, if everything is building well, it's going to build well when I actually push it up for real.

This would be where you've got 7, 8, 9 people all banging on things where they're crossing over, tens or dozens of classes, and doing completely different capabilities simultaneously, so yes, it doesn't matter on a really small team - but even a small team could have that happen, but it's rare - the bigger the team, the greater the likelihood of the overlap of features, collision, where you're actually trying to add orthogonal features, that shouldn't have any impact, but if you do any refactoring while you're doing it, you want to move the time when you have a discovery of "Oh, that's no longer called that or, you moved it over here in a completely different class" you really want to get everybody's stuff out there, you want them out there visible - they may not be in the trunk, but they're out there in the branches, and they're already automatically being compared by the CI system, so that you can alert everybody early and say "Hey, somebody made a change that breaks somebody else's working branch".

And then, you know, by the end of your sprint you cycle everything in, but it's surprising on a larger team, at certain times - very often it doesn't matter - but when it does, every day counts, every hour counts sometimes.

Builds: Tips and Tricks

In the past I've had mix builds, C++, Java, C - now it's more Java helping people out with those kind of things, but I probably mentioned at one point that I use tools like Gradle, you should be using something at least like Maven. Ideally, there's this notion of "Well the builds should be fast", well that's great, but that's a clean build - what is it when I make one little change, how many ripple effects do I have? In lots of build tools, if you're not careful, you make a change and everything re-complies, everything re-builds an artifact downstream, and it triggers. You have to be really careful to try and get, either your CI system, or your build tool itself - hopefully you have one that can help partition those off.

I remember the old school builds of C++ when templates came in, and hour long builds where not uncommon, and that was spreading them out on multiple machines. So, things are better in technology now, but it comes down to - "How quick you can make it so the developer doesn't go "Ahh, I don't want to make all these changes - I'm going to make all of them at once, because I know the build is going to take too long."

So really the trick is - focusing with somebody who pretends, or really is a build master, and trying to get your builds to function properly - because the pay pack is immense, it's amazing how much, if you can shave off 30 seconds a minute, off from any kind of build, you're talking about huge, like hour to minutes - even for a regular dev during the day, you start shaving off a few minutes here and there, and it makes a big difference in the cycle time.

So you're generating artifacts of some sort, and you should be storing those, and reusing them throughout your pipeline - we haven't talked about pipelining yet, but definitely when you're doing CI, you never want to rebuild the same item unless you absolutely have to at some later point in time. And then use that throughout the different chains throughout your builds, and we'll talk about testing later, because that's when you often use a lot of these generator artifacts, and then, if you've already got it pre-built, grab it and reuse it and go. It's only when you've actually got a version change, that that you should have to have any compilation occurring.

That's not true of a lot of systems but it's getting more true because people are starting to use the contemporary tools and make change, it's not uncommon for everybody to have their own Artifactory, or some type of Nexus server to contain that, publish to it, I mean it used to be considered "Wow", but now it's the norm. And it absolutely should be, and your CI, whatever it is, should be able to pull from it, as necessary, or leverage your build tool - they should be cooperative.

Testing

We've got the separations and the unit integration, and acceptance test - essentially, your unit test should be so fast you should break the build if they're not fast enough. Make your devs not do IOx's, all these other things, they're supposed to be focusing on those classes. And it makes a big impact if you make sure that your devs are running the unit test every single compile, it should be considered part of the compile.