Are code freezes still relevant when using a continuous integration build setup?

Asked 16 years, 1 month ago Modified 16 years, 1 month ago

Viewed 4k times Part of CI/CD Collective

5

I've used a Continuous Integration server in the past with great success, and hadn't had the need to ever perform a code freeze on the source control system.

However, lately it seems that everywhere I look, most shops are using the concept of code freezes when preparing for a release, or even a new test version of their product. This idea runs even in my current project.

When you check-in early and often, and use unit tests, integration tests, acceptance tests, etc., are code freezes still needed?

CI / CD

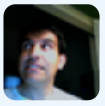
continuous-integration development-process code-freeze

Share

Improve this question

Follow

asked Nov 11, 2008 at 20:54



casademora

69.5k ● 18 ● 71 ● 78

5 Answers Sorted by: Highest score (default)

6

Continuous integration is a "build" but it's part of the programming part of the development cycle. Just like the "tests" in TDD are part of the programming part of the development cycle.

There will still be builds and testing as part of the overall development cycle.

The point of continuous integration and tests is to shorten the feedback loops and give programmers more visibility. Ultimately, this does mean less problems in testing and builds, but it doesn't mean you no longer do the original parts of your development cycle - they are just more effective and can be raised to a higher level, since more tivial problems are being caught earlier in the development cycle.

So you will still have to have a code freeze (or at least a branch) in order to ensure the baseline for what you are shipping is as expected. Just because someone can implement something with a high degree of confidence does not mean it goes into your release without passing through the same final cycles, and the code freeze is an important part of that.

With CI, your code freezes can be very short, since your final build, testing and release may be very reliable, and code freeze may not even exist on small projects, since there is no need for a branch - you release and go right

back into development on the next set of features very quickly.

I'd also like to add that CI and TDD allow the final build and testing phase to revert back closer to the traditional waterfall (do all dev, do all testing, then release), as opposed to the more continual QA which has been done on projects with weekly or monthly builds. Your testers can use the CI builds to give early feedback, but it's effectively a different sort of feedback than in the final testing, where you are looking for stability and reliability as opposed to functionality (which obviously was missed in the unit "tests" which the developers had built).

Share Improve this answer Follow

edited Nov 11, 2008 at 21:52

answered Nov 11, 2008 at 21:04

Cade Roux

89.6k ● 40 ● 184 ● 266

1 This was my impression. Also, I think that since you have so many builds with a CI server, you don't NEED a code freeze because you have so many point in time builds to choose from to pick a "good" version. – casademora Nov 11, 2008 at 21:12

Casdemora, Just because something compiled doesn't mean it works at all. – FlySwat Nov 11, 2008 at 21:14

I very much doubt that your QA staff would like to randomly select builds (out of those which compiled and ran all the tests) and QA them for release. If one fails, do you look for a

newer or older CI build? And which features are left out of this build, or that one... – Cade Roux Nov 11, 2008 at 21:48

3

Code freezes are important, because continues integration does not replace runtime regression testing.

Having an application build and pass unit testing is only a small part of the challenge, ideally, when you freeze code for a release, you are signing off on two things:

- This code has been fully regressioned, and is defect free
- This code is EXACTLY the code that should be in production (for SOX compliance).

If your using a modern SCM, just fork the code at that point and start work on the next release in a branch, and do a merge when the project is deployed. (Of course, place a label so you can rollback that point if you need to apply a breaking patch).

Once code is in "release mode", it should not be touched.

Our typical process:

```
Development
   ||
   \/
   QAT
   ||
   \/
   UAT => Freeze until deploy date => Deploy  =>
Merge and repeat
```

```
              \
/
               \- New Branch for future dev -------/
```

Of course, we usually have many parallel branches during development, that merge back up into the release stream before UAT.

Share Improve this answer

Follow

answered Nov 11, 2008 at 20:59

FlySwat

175k ● 75 ● 248 ● 314

1

The code freeze has more to do with QA than it has to do with Dev. The code freeze is the point where QA has said: "Enough. We only have bandwidth to fully test the new features added in so far." That doesn't mean dev doesn't have the bandwidth to add more features, it's just that QA needs time with a quiescent code base to ensure that everything works together.

If you're all in continuous integration mode (QA included) this could be just a freeze of a very short time while QA puts the final seal of approval on the whole package just before it goes out the door.

It all depends on how tightly your QA and regression testing are integrated into the dev cycle.

I'd second the votes already mentioned about SCM branching and allowing dev to continue on a different code branch than what QA is testing. It all goes back to

the same thing. QA and regression testing need a quiescent code base for a period of time prior to release.

Share Improve this answer

Follow

answered Nov 11, 2008 at 21:10

Steven M. Cherry

1,375 ● 1 ● 11 ● 14

0

I think that code freezes are important because each new feature is a potential new source of bugs. Sure regression tests are great and help address this issue. But code freezes allow the developers to focus on fixing *currently outstanding bugs* and get the current feature set into a release worthy state.

At best, if I really wanted to develop new code during a code freeze, I would fork the frozen tree, do my work there, then after the freeze, merge the forked tree back in.

Share Improve this answer

Follow

answered Nov 11, 2008 at 20:58

Evan Teran

90.3k ● 32 ● 187 ● 243

0

I'm going to sound like one of the context-driven people but the answer is "it depends".

Code Freeze is a strategy to cope with a problem. If you don't have the problem it is good at addressing, then no, it isn't needed. If you have another technique for addressing the problem, then no, it isn't needed.

Code Freeze is one technique for reducing risk. The advantages if brings are stability and simplicity. The disadvantage it brings are

Another technique is to use Branching, such as with "Feature Branches". The disadvantage of Branching is cost of dealing with the branches, of merging changes.

The technique you're describing for reducing risk is Automated Testing to give fast feedback. The trade-off here is increased velocity for some increased risk (you will miss some bugs).

Of these approaches I'm with you, I prefer the Automated Testing. But there are some situations, such as very high cost of failure, where a Code Freeze does provide a lot of value.

answered Nov 11, 2008 at 21:47

community wiki
Jeffrey Fredrick