How to manage frequently modified production code? [closed]

Asked 16 years, 1 month ago Modified 15 years, 4 months ago

Viewed 1k times

5

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 5 years ago.

Improve this question

My project invovles me to make a lot of changes on the production code. The requirement keeps coming up and I need to make changes and deploy it as soon as possible. I sometimes end up creating patch work sort of code because some of the requirement would not fit into the overall design of the software. How can this be handled effectively? Any design pattern to handle this?

design-patterns version-control project-management

Share

Improve this question

Follow

edited Aug 6, 2009 at 12:47

Jeremy Cron

2,412 ● 3 ● 25 ● 30

asked Nov 18, 2008 at 10:29

Manoj

5,087 ● 14 ● 55 ● 77

Frequently modified production code?!?! DEAR GOD NOOOOOOO! – OJ. Nov 18, 2008 at 10:42

There is always other options... – Ilya Nov 18, 2008 at 12:18

12 Answers

Sorted by: Highest score (default)

14

I've seen this happen many times and it always ends in tears. The last time the customer lost millions of dollars before they improved their process.

Users always want new requirements to be made available "as soon as possible," but they do not understand the risks of making the changes in the same way that you do. They see the cost of *not* having the feature but they don't anticipate what would happen and how much money would be lost if the change breaks something. (I don't mean to suggest that you're not a good developer, just that in any non-trivial software there will be bugs and that uncontrolled changes are likely to expose them more quickly.)

So, I think you need to take a step back and try to instigate a regular release schedule. There will be resistance but you can be more effective that way. Of course sometimes there *will* be a change that needs to be made immediately, but if there's a schedule then the onus will be on the user to justify why breaking the release cycle makes sense.

Oh, and as everyone else suggests, you need technical infrastructure like a staging/testing system, one-click release procedure, etc. (See The Joel Test.)

Share Improve this answer

Follow

answered Nov 18, 2008 at 10:57

Stephen Darlington

52.5k ● 12 ● 108 ● 153

I totally Agree with you Stephen Darlington, but there are times products meet unexpected situation when they get released, and they need to be fixed fast. Say think about any OS on the market. They follow major releasing but there will always be a need for hot-fixes because its too critical – Robert Gould Nov 18, 2008 at 11:31

@Robert Gould, I'm not sure if you read his whole post but he states that there are times when you might have to release immediately. "Of course sometimes there will be a change that needs to be made immediately" – mmcdole Nov 18, 2008 at 12:14

@Robert There is absolutely a place for hot-fixes. However these should be very much the exception. You need to make a concious decision to apply one; it should not be the default option. – Stephen Darlington Nov 18, 2008 at 13:21

5

Testing, you need a solid testing framework to be sure your fixes don't break anything else.

Edit: Answer to comment's question.

Unfortunately I can't think of a truly sound pattern/solution to keep the architecture intact besides taking your time to refactor the "hacks". But you probably have little time to spare since your in production already. So it's not easy...

However more importantly if the architecture is getting **spoiled** because you really need to "hack" the solution in, this might actually be a sign that *the original design wasn't meeting the product's actual requirements*, because if it was, you should be able to fix/patch within the Architecture's current framework.

So trying to be positive about the whole situation you should take note of your fixes, and how the current Architecture isn't helping/complying, so you can later down the road, when the hot-fixes begin to settle, have data and hints as to redesign whatever parts of the Architecture necessitate design now that you have the more accurate requirements you discovered during production.

edited Nov 18, 2008 at 11:28

answered Nov 18, 2008 at 10:45

Robert Gould
69.7k • 61 • 191 • 275

Thanks for that suggestion! I am worried about one more thing. I am actually spoiling the whole architecture modifying the code regularly. Any suggestion regarding that? – Manoj Nov 18, 2008 at 10:58

+1: If the change "breaks" the architecture, then usually the architecture wasn't right to begin with. Rarely does the architecture break because the change is a toweringly bad idea. – S.Lott Nov 18, 2008 at 11:35

Yeah, I think I need to take a re look at the architecture. I should be taking note of the fixes and try to redesign it down the road. Thanks for the suggestion! – Manoj Nov 18, 2008 at 11:54

@Robert - " ... begin to settle, have data and hints as to redesign whatever parts of the Architecture ...". This can be dangerous, especially if business aren't aware that you're doing this. In this case, unless you have a safety net of regression tests, you're on a hiding to nothing. – Eric Smith Feb 20, 2009 at 10:10

@Eric that's actually true, one has to ALSO be careful of not breaking bussiness backwards compatibility! – Robert Gould Feb 20, 2009 at 13:04

4

Everybody here has suggested pretty good things, like testing, etc. But I think I should point out that you may be asking the wrong question. You are asking if there's a "pattern" that can help with this situation. A pattern is a design choice to solve design problems. What you essentially have is a "process" problem.

I would ask "What process can I use to prevent this?"

Unfortunately, (and it is the same where I work) design is an issue for developers and architects, but process is an issue for managers. And it really takes leadership to implement good process, and stick to it. Sadly that is often lacking.

Share Improve this answer

Follow

answered Nov 18, 2008 at 13:28

dviljoen

1,632 ● 2 ● 17 ● 28

2

The may be too basic of an answer, but I think you should do some research on agile software development.

Share Improve this answer

Follow

answered Nov 18, 2008 at 10:41

ng.mangine

2,977 ● 1 ● 17 ● 7

2

I'm in the lucky position where my HEAD is almost always releasable. At least once a week, I have code in HEAD that, as a developer, I would be happy to release. That doesn't mean that every releasable version actually gets a release, but it could. In my environment, a weekly release is actually quite practical, and usually gets made...

Immediately before deploying to staging, I promote my code to a Release branch. I always deploy the same code to live that has previously been tested on staging.

Any urgent fixes can then be made in the release branch and tested on Staging, before being deployed. If the fix is good enough, I can merge it back into HEAD. If it was a horrible hack, I can re-implement it properly in HEAD later.

I have a good suite of developer tests that are run automatically on every check-in, which confirm I haven't broken anything important. My application also runs internal tests every time it is deployed, again making me confident.

Actually, luck is less a factor than you might think. This didn't just happen by accident; I had to work to make it possible. I had to commit to writing and maintaining good automated tests, and to getting a continuous integration server and a one-click build-and-deploy capability.

I regularly spend time cleaning up my code, as part of my normal development activities. This has two benefits. First, it means that my code base starts out relatively clean, so the architecture is pretty flexible. Second, it means I'm good at refactoring, since I do it all the time. By this, I mean refactoring in the sense of making a series of individually small transformations to existing code, rather than in the throw-it-all-away-and-re-implement sense (which is somewhat more dangerous).

In my opinion, this "continuous releasability" is the single biggest benefit of Agile methodologies.

Share Improve this answer

Follow

answered Nov 18, 2008 at 11:58

Bill Michell

8,350 ● 3 ● 30 ● 33

1

Besides the obvious answers of revision control, unit tests, and an automated build system, it sounds like you also need to do some heavy refactoring. If your design keeps changing based on changing requirements then you should try to isolate the parts of your code that keep changing in their own modules.

Share Improve this answer

Follow

answered Nov 18, 2008 at 13:20

Bill the Lizard

405k ● 211 ● 572 ● 889

1

You need to impose discipline on your requirements backlog. At the risk of sounding old school and crotchety, tell people no and that they need to sit with you and learn the specific points of pain and effort for drastic design changes. For databases, explain about cardinality and how this causes heartache. Throw your hay against the cage and insist on a scheduled release cycle when you will only role into production every Tuesday, but only after user acceptance has take place. Break each request into segments of 2,4 and 8 weeks and be rigid about what you will include in those time frames.

There are many design patterns that can help you if you have a defined domain of problems and their solutions. Specifically review the Command Pattern, the Strategy Pattern, and Plug-in architecture, as they will help you extend your solution set more easily. If you are .Net developer, take a look at Migeul Castro's plugin architecture he reviewed on DNRTV.

Share Improve this answer

Follow

answered Nov 21, 2008 at 12:28

David Robbins

10k ● 7 ● 54 ● 83

> Which doesn't help when the app just isn't working in a critical way, and leaving it until Tuesday after next will cost millions of dollars. – David Thornley Feb 20, 2009 at 18:14

0

There are several project lifecycles that you can pattern your approach against. This site lists a few (it sounds like you're using the "Code-And-Fix" one!), but this is by no means a definitive list, a larger list can be found here.

Share Improve this answer

Follow

answered Nov 18, 2008 at 10:43

J c

6,413 ● 3 ● 31 ● 29

0

As Robert Gould pointed out, you really need to have a staging platform to check that you're not going to break anything when deploying to live.

Share Improve this answer

Follow

answered Nov 18, 2008 at 10:50

Rob Wells

37k • 13 • 84 • 147

0

two obvious tools are version control and testing. try to integrate the release system with them, so you can commit your changes at every step, and when all tests pass (including those for the new requirements), the release system picks the 'known good' version, that will be the new one.

i don't know about other systems, but monotone has some hooks specifically to make the tests tag the commits, so there's a command to "give me the last version that passes all tests"

Share Improve this answer

Follow

answered Nov 18, 2008 at 12:23

Javier

62.4k • 9 • 81 • 126

0

Obviously, testing is important. But for me automation is even more. You should be able to deploy the whole project from the source control to the production server in less than 3 commands. Tools like Maven, Ant, Capistrano, (insert your favorite tool here <...>) can help you a lot. Having a continuous integration system that deploy automagically to a test or integration server every time there is a change in source control can also help.

Putting all that automation in place will take time the first time you do it ...

Share Improve this answer

Follow

answered Nov 18, 2008 at 12:52

Guillaume

18.8k ● 8 ● 55 ● 76

0

This is a process thing, not a design thing.

The first thing you need to do is educate your users so they have some idea of the cost of changing requirements. This isn't intended to stop them from changing them, but to avoid changing them and requesting implementation ASAP unless there's a definite business need. (I assume this is a business thing, in which case it's your job to do what the business needs to the best of your ability, and to let people know what the best of your ability is, and what functionality costs in the overall scheme of things.)

The second thing is to establish a two-fix process for fast fixes. First, you get the patch in place to keep the app running more or less satisfactorily, and then you take some time and come up with the real fix. This may also require some user education, and you may find the idea of "technical debt" useful in explaining this.

The third thing is to make sure you have the resources. If you can't keep up with requirements changes like this, you need help, and you need to show why. If the powers that be decide not to bring in more people, and they

understand that limits the amount of modification, that's good too.

Now, it may happen that your users are unteachable, that you can't convince people that quick fixes are very expensive in the long run, and so on. In that case, I recommend the fourth thing: polish your resume and start looking for a new job. Right now, it looks like you're being set up to fail, and that's a very bad place to be. As the old saying goes, change your job or change your job.

Share Improve this answer

Follow

answered Feb 20, 2009 at 18:24

David Thornley
57k • 9 • 95 • 158