How not to rush yourself? [closed]

Asked 16 years ago Modified 15 years, 4 months ago Viewed 891 times

7

I often find that I do a less than complete work on a feature, especially in the Design phase. I detect several reasons:

1. I'm over-optimistic
2. I feel the need to provide quick solutions, so sometimes I fool myself into thinking the design is fool-proof when in fact it's still full of holes, just to get the job done faster. Of course I end up paying dearly later.

I'm aware of this behavior of mine for some time, yet I still find I don't manage to compensate. Have you encountered similar problems? How do you approach solving them?

project-management

Share

Improve this question

Follow

edited Aug 23, 2009 at 12:03

community wiki
4 revs, 2 users 100%
ripper234

I'm voting to close this question as off-topic because this is not about programming – Luuklag Nov 14, 2018 at 15:28

13 Answers

Sorted by: Highest score (default)

11

I use a couple of techniques. The first is a simple paper to-do list. In the morning I write down my tasks for the day. I try to work on a task until I can cross it off. I cross it off only when I'm done to my own satisfaction. My to-do list helps me stay focused. When an interruption comes in, I can consciously choose whether it is important enough to interrupt what I'm doing now.

The second technique I use is to give up on the idea of "done" for a design. Instead, I focus on what I've started calling "successions", where a design goes through predictable stages. Each stage supports the current functionality well and will be succeeded at some point by the next stage. This lets me do a good job, a job I can be proud of, without over-designing.

I have the intuition that there is a small catalog of such successions (like http://www.threeriversinstitute.org/FirstOneThenMany.html) that would cover most of design. In the meantime, I try to remember that "sufficient to the day are the troubles thereof".

Share Improve this answer Follow

answered Dec 3, 2008 at 18:07
Kent Beck
5,011 ● 2 ● 31 ● 31

Here's a working link for FirstOneThenMany: web.archive.org/web/20120726012122/http://… – maligree May 9, 2017 at 10:14

6

I run into this problem a lot.

My solution is a notebook. (The old fashioned paper kind).

I write out how I'm planning on implementing the solution as an bulleted overview list, and then I try and flesh out each point on the list.

Often, during that process, I come across issues I hadn't thought of.

Of course, the 80/20 rule still applies... I still come across things when I'm actually doing the implementation that hadn't occurred to me, but with experience these tend to diminish.

EDIT: If I'm still not sure at the end of this process, I put together a throwaway prototype testbed... It's important to make sure it's throwaway, because otherwise you run the risk of including some nasty hacks in your real codebase.

Share Improve this answer Follow

answered Dec 3, 2008 at 17:39

Andrew Rollings
14.6k ● 7 ● 53 ● 51

Agreed. Sit down and decompose the problem. – jim Dec 3, 2008 at 17:50

3

It's very common to miss edge-cases and detail when you're in the planning phase of a project, especially in the software development field. Please don't feel that this is a personal failing; it's something endemic.

To counter this, many software development *methodologies* have emerged. Most recently there has been a shift by many development teams to 'agile' methods, where there is a focus on rapid development with little up-front technical design (after all, many complexities are only discovered when you actually begin developing). I'm currently using the Scrum system, which has been excellent in my small team:

http://en.wikipedia.org/wiki/Agile_methods

http://en.wikipedia.org/wiki/Scrum_%28development%29

If you find that your organisation will not accept what they may regard as a radical shift in approach, it may be worth investigating whether they will agree to the development of a prototype system. This means that you could code up a feature to investigate the technologies involved and judge whether it's feasible, without having to commit to full development, a quality bar, testing schedules etc. The prototype should be thrown away once the feasibility has been proved or disproved, then proper development may begin, including all that you've learned in the process.

If your problem is more related to time management, then I'd recommend the Getting Things Done approach (http://en.wikipedia.org/wiki/Getting_things_done). This is pragmatic and simple, concentrating on making you productive without overloading you with information that isn't immediately relevant to your current work. I've found that I get overwhelmed with project/feature ideas at times and it really helps to write everything down and file it for a later time when I have the resources available to work effectively.

I hope this helps and best of luck!

Share Improve this answer Follow

answered Dec 3, 2008 at 17:52

Dave R.

7,282 ● 3 ● 32 ● 53

3

Communication.

The best way to not rush yourself into programming mistakes is communication. Yes, good ol' fashioned accountability. If another person in the office is involved in the process, the better the outcome. If a programmer just takes on the task without any concern for anybody else, then there is a higher possiblity for mistakes.

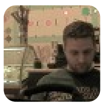
Accountability Checklist:

1. How do we support this?
2. Who needs to know what has changed?
3. Why are we doing this in the first place?
4. Will there be anybody who doesn't want this changed?
5. Will someone else understand how I did this?
6. How will the user perceive and use this change?

A skepticle comrad is usually good enough to help. Functional Specifications are good, they usually answer all of these thoughts. But, sometimes a conversation with another person can help you with it and you can get changes out the door faster.

Share Improve this answer Follow

answered Dec 3, 2008 at 18:13



Jeremiah
5,516 ● 9 ● 40 ● 46

0

I have learned, through years of mistakes (though still making them), that almost anything I want to use repeatedly, or distribute, needs to be designed properly. So getting burned enough times will end your optimism.

When getting pressure from management, I tell them I will have to put in the thought anyway, so I should do it when it's cheap. I think on paper as well, so I can actually prove that I'm doing something and it keeps my fingers on the keyboard, both of which provides a soothing effect to management. ;-)

Share Improve this answer Follow

answered Dec 3, 2008 at 17:43

John MacIntyre

13k ● 13 ● 69 ● 107

0

At the risk of sounding obvious - be pessimistic. I had a few experiences where I thought "that should take a few hours" and it ended up taking a couple *days* because of all the little things that pop up unexpectedly.

By far the best way I've found to manage things is to (much like Andrew's answer) write out the design and requirements as a starting point. Then I go through and look for weak points in the design, gotchas and additional use cases etc. I try to look at this as a critical exercise - there's no code written yet, so this is the time to be totally ruthless and look for every weak point. Look for error conditions you'll have to handle, and whatever amount of time you think it will take to complete each feature/function, pad that amount by a lot. I've had times where I've doubled my initial estimate and still not been that far off the mark.

It's very hard as a programmer to realistically project debugging time - writing the code is easy to estimate, but debugging that into functioning, valid code is something else entirely. Therefore I find there's no exact science to it but I just pad tasks by a whole bunch, so that I have plenty of breathing room for debugging.

See also Evidence Based Scheduling which is a fascinating concept in scheduling developed by FogCreek for their FogBugz product.

Share Improve this answer Follow

answered Dec 3, 2008 at 17:55

Jay
42.6k ● 14 ● 68 ● 83

0

You and the rest of the world.

You need more a more detailed design, more accurate estimate, and the willingness to accept that sometimes the optimal solution is not necessarily the best solution (e.g., you could code some loop in assembler to get optimal performance, but that's going to take a lot longer than just doing

```
for (i=1; i<=10; i++) {}
```

). Is the time spent doing it really worth it for an accounting package over a missile system.

Share Improve this answer Follow

answered Dec 3, 2008 at 17:55

BIBD
15.4k ● 26 ● 87 ● 141

0

I like to designing, but over time I've found that much design up front is a lot like building castles into the sky - it's too much speculation, however well-educated, missing critical feedback from actually implementing and using the design.

So today I'm much more into accepting that while implementing a design I *will* learn a lot of new stuff about it, and need to feed that learning back into the design. Doing that is a skill that is fun to learn, including the skills to keep a design flexible by keeping it simple, free of duplication and cohesive and decoupled, of changing the design in small, controlled steps (=refactoring), and writing the necessary extensive suite of automated tests that make this kind of changes safe.

This seems to be a much more effective approach to me than getting better at "up front design speculation" - and addtionally it makes me equally well prepared for the inevitable moment when the design needs to be changed due to a simply unforseeable change in the requirements.

answered Dec 3, 2008 at 18:08

Share Improve this answer

Follow

Ilja Preuß
2,421 ● 17 ● 15

0

Divide, divide, divide. List *all* the steps that will be required to finish the project, then list all the steps *those* steps will require to be concluded, and so on until you reach atomic items you are absolutely sure you can finish in a day or less. Add the duration of all these values to arrive at a length of time.

Then double it. Now you have a number that, if depressing, is at least somewhat realistic.

Share Improve this answer

Follow

answered Dec 3, 2008 at 18:22

zaratustra
8,698 ● 9 ● 38 ● 42

0

If possible "Sleep on your design" before publishing it. I find after I leave work, I usually think of things I have missed. This usually happens while I am lying in bed before falling asleep or even while showering the next day.

I also find it valuable to have a peer/friend that I trust review what I have before distributing it. Somebody else almost always sees something I didn't think of or miscommunicated.

Share Improve this answer

Follow

answered Dec 3, 2008 at 18:24

Jim Anderson
3,622 ● 2 ● 26 ● 21

0

I like to do as others stated here. Write down in pseudo code what the flow of your app will be. This immediately highlights some detailed areas that may require further attention that where not apparent up front.

Pseudo code is also readable to business users who can verify your approach meets their needs.

Using pseudo code also creates a nice set of methods that could be put to use as an interface in the final solution. Once the pseudo code is fairly tight, look for patterns and review some common GOF patterns. They do not have to be perfect but using them will sheild you from having to rewrite the code later during the revisions that are bound to come along.

Just taking an hour or two write psuedo code, yields some invaluable time saving pieces later on: 1. An object model emerges 2. The program's flow is clearly defined for others 3. It can be used as documentation of your design with some refinement 4. Comments are easier to add and will be clearer for someone else reviewing your code.

Best of luck to you!

Share Improve this answer Follow

answered Dec 3, 2008 at 21:22
iamlouis

0

I've found that the best way to make sure you've chosen a good design is to make sure that you understand the problem, know the limitations you have, and know what things are must-haves vs. nice-to-haves.

Understanding the problem will involve talking to the people who have the need and keeping them anchored to what needs to get done first instead of how they think it ought to get done. Once you know what actually has to happen, you can go back and talk over requirements about how.

Knowing your limitations may be quite easy: needs to run on the iPhone; has to be a web application; needs to integrate with the already-existing Java code and deployment setup; and so on. It may be quite difficult: you don't know what the potential size of your user base is (hundreds? thousands? millions?); you don't know whether you'll need to localize it (though if you're not sure, assume you will have to).

Must-haves vs, nice-to-haves: this is possibly the most difficult part. Users very often have emotional attachments to "requirements" ("It should look just like Excel") that are not actually part of the "has to happen" stuff. You often have to juggle functionality vs. desires to get an acceptable implementation. You can't always give everyone a pony.

Make sure you write all this down! Even if it evolves along the way, or the design is small, having a "this is what we're planning to do now" guide to refer to when you need ot make a decision about committing resources makes it easier to restrain yourself from implementing a really cool whiz-bang feature instead of a boring must-do.

Share Improve this answer Follow

answered Dec 3, 2008 at 21:47

Joe McMahon

3,384 ● 23 ● 34

0

Since you recognize that you feel the need to provide a quick solution, perhaps it will slow you down to realize that you can probably solve the problem faster and deliver it sooner if you spend more upfront time in design. For instance if you spend 3 hours designing and 30 hours writting code, it probably means that if you spend 6 hours designing you might need to only spend 10 hours writing code. (These are not actual figures just examples). You might try to quantify this for yourself on the next few projects you do. Do a couple where you behave as you normally would and see what ratio of design/codewriting/testing&debugging you actually do. Then on the next project deliberately increase the percentage of time you spend on design phase and see if it does shorten the time needed for the other phases. You will have to try for several projects on this as well to get a true baseline since the projects may be quite different. Do it as a test to see if you can improve your performance on the the other phases and thus deliver a faster product if you spend 20% more time or 50% more time or 100% more time on design.

Remember the later in the process you find the problem with a design the harder (and more time-consuming) it is to fix.

answered Dec 3, 2008 at 22:03

HLGEM
96.4k ● 15 ● 119 ● 189