Calculating Project Programming Times [closed]

Asked 14 years, 4 months ago Modified 6 years, 10 months ago

Viewed 7k times

21

As a lead developer I often get handed specifications for a new project, and get asked how long it'll take to complete the programming side of the work involved, in terms of hours.

I was just wondering how other developers calculate these times accurately?

Thanks!

Oh and I hope this isn't considered as a argumentitive question, I'm just interested in finding the best technique!

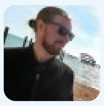
project-management time-management

Share

Improve this question

Follow

asked Aug 6, 2010 at 11:57



Curtis

103k ● 68 ● 276 ● 357

12 Ask for worst case estimate and multiply by 5. – Oded Aug 6, 2010 at 11:59

4 @Oded: In my experience, 8 is a safer number :) – leppie Aug 6, 2010 at 12:05

4 @leppie - I am an optimist at heart... – Oded Aug 6, 2010 at 12:09

@Oded: ...and that's exactly the most commonly made mistake in estimating programming times :-) – Adrian Grigore Aug 6, 2010 at 12:11

1 We always figure out roughly how long it will take, and then multiply by a random number between e and π. The key is random, cause if you always use the same one, the people will figure out the multiplier :) Under promise/over deliver.. – Doon Aug 6, 2010 at 12:29

6 Answers

Sorted by: Highest score (default)

21

Estimation is often considered a black art, but it's actually much more manageable than people think.

At Inntec, we do contract software development, most if which involves working against a fixed cost. If our estimates were constantly way off, we would be out of business in no time.

But we've been in business for 15 years and we're profitable, so clearly this whole estimation thing is solvable.

Getting Started

Most people who insist that estimation is impossible are making wild guesses. That can work sporadically for the smallest projects, but it definitely does not scale. To get consistent accuracy, you need a systematic approach.

Years ago, my mentor told me what worked for him. It's a lot like Joel Spolsky's old estimation method, which you can read about here: Joel on Estimation. This is a simple, low-tech approach, and it works great for small teams. It may break down or require modification for larger teams, where communication and process overhead start to take up a significant percent of each developer's time.

In a nutshell, I use a spreadsheet to break the project down into small (less than 8 hour) chunks, taking into account everything from testing to communication to documentation. At the end I add in a 20% multiplier for unexpected items and bugs (which we have to fix for free for 30 days).

It is very hard to hold someone to an estimate that they had no part in devising. Some people like to have the whole team estimate each item and go with the highest number. I would say that at the very least, you should make pessimistic estimates and give your team a chance to speak up if they think you're off.

Learning and Improving

You need feedback to improve. That means tracking the actual hours you spend so that you can make a comparison and tune your estimation sense.

Right now at Inntec, before we start work on a big project, the spreadsheet line items become sticky notes on our kanban board, and the project manager tracks progress on them every day. Any time we go over or have an item we didn't consider, that goes up as a tiny red sticky, and it also goes into our burn-down report. Those two tools together provide invaluable feedback to the whole team.

Here's a pic of a typical kanban board, about halfway through a small project.

You might not be able to read the column headers, but they say Backlog, Brian, Keith, and Done. The backlog is broken down by groups (admin area, etc), and the developers have a column that shows the item(s) they're working on.

If you could look closely, all those notes have the estimated number of hours on them, and the ones in my column, if you were to add them up, should equal around 8, since that's how many hours are in my work day. It's unusual to have four in one day. Keith's column is empty, so he was probably out on this day.

If you have no idea what I'm talking about re: stand-up meetings, scrum, burn-down reports, etc then take a look at the scrum methodology. We don't follow it to the letter, but it has some great ideas not only for doing estimations, but for learning how to predict when your project will ship as new work is added and estimates are missed or met or exceeded (it does happen). You can look at this awesome tool called a burn-down report and say: we can indeed ship in one month, and let's look at our burn-down report to decide which features we're cutting.

FogBugz has something called Evidence-Based Scheduling which might be an easier, more automated way of getting the benefits I described above. Right now I am trying it out on a small project that starts in a few weeks. It has a built-in burn down report and it adapts to your scheduling inaccuracies, so that could be quite powerful.

Update: Just a quick note. A few years have passed, but so far I think everything in this post still holds up today. I updated it to use the word kanban, since the image above is actually a kanban board.

Share Improve this answer Follow

edited Feb 8, 2017 at 14:29

community wiki
18 revs
Brian MacKay

1 I know this is 3 years old, but would you be able to repost the image of your project board which you posted about two-thirds down your post? – William Grand Dec 18, 2013 at 19:05

+1 to give us a good reference like 'Joel on Estimation' – kta Jan 10, 2014 at 6:17

@WilliamGrand I couldn't find that image, but I plugged in a similar one and edited around it. – Brian MacKay Jan 10, 2014 at 14:24

4

There is no general technique. You will have to rely on your (and your developers') experience. You will have to take into account all the environment and development process variables as well. And even if cope with this, there is a big chance you will miss something out.

I do not see any point in estimating the programming times only. The development process is so interconnected, that estimation of one side of it along won't produce any valuable result. The whole thing should be estimated, including programming, testing, deploying, developing architecture, writing doc (tech docs and user manual), creating and managing tickets in an issue tracker, meetings, vacations, sick leaves (sometime it is batter to wait for the guy, then assigning task to another one), planning sessions, coffee breaks.

Here is an example: it takes only 3 minutes for the egg to become roasted once you drop it on the frying pen. But if you say that it takes 3 minutes to make a fried egg, *you are wrong*. You missed out:

- taking the frying pen (do you have one ready? Do you have to go and by one? Do you have to wait in the queue for this frying pen?)
- making the fire (do you have an oven? will you have to get logs to build a fireplace?)
- getting the oil (have any? have to buy some?)
- getting an egg
- frying it
- serving it to the plate (have any ready? clean? wash it? buy it? wait for the dishwasher to finish?)
- cleaning up after cooking (you won't the dirty frying pen, will you?)

Here is a good starting book on project estimation:

http://www.amazon.com/Software-Estimation-Demystifying-Practices-Microsoft/dp/0735605351

It has a good description on several estimation techniques. It get get you up in couple of hours of reading.

Share Improve this answer Follow

answered Aug 10, 2010 at 6:42

Max Kosyakov

292 ● 2 ● 5

1 Just thought I would say that I have this book, and for my purposes it's not very practical. It felt almost academic. But maybe some other folks got some value out of it.
– Brian MacKay Aug 10, 2010 at 17:18

2

Good estimation comes with experience, or sometimes not at all.

At my current job, my 2 co-workers (that apparently had a lot more experience than me) usually underestimated times by 8 (yes, **EIGHT**) times. I, OTOH, have only once in the last 18 months gone over an original estimate.

Why does it happen? Neither of them, appeared to not actually know what they are doing, codewise, so they were literally thumb sucking.

Bottom line:

Never underestimate, it is much safer to overestimate. Given the latter you can always 'speed up' development, if needed. If you are already on a tight time-line, there is not much you can do.

Share Improve this answer Follow

answered Aug 6, 2010 at 12:03

leppie
117k ● 18 ● 200 ● 300

2

If you're using FogBugz, then its Evidence Based Scheduling makes estimating a completion date very easy.

You may not be, but perhaps you could apply the principles of EBS to come up with an estimation.

Share Improve this answer Follow

answered Aug 6, 2010 at 12:10

bcmcfc
26.7k ● 30 ● 113 ● 183

1

This is probably one of the big misteries in the IT business. Countless failed software projects have shown that there is no perfect solution to this yet, but the closest thing to solving this I have found so far is to use the adaptive estimation mechanism built in to FogBugz.

Basically you are breaking your milestones into small tasks and guess how long it will take you to complete them. No task should be longer than about 8 hours. Then you enter all these tasks as planned features into Fogbugz. When completing the tasks, you track your time with FogBugz.

Fogbugz then evaluates your past estimates and actual time comsumption and uses that information to predict a window (with probabilities) in which you will have fulfilled your next few milestones.

Share Improve this answer

edited Feb 22, 2018 at 15:28

Follow

Ikem Krueger
193 ● 1 ● 1 ● 12

answered Aug 6, 2010 at 12:10

Adrian Grigore
33.3k ● 36 ● 132 ● 215

Ah wish we still had FogBugz! We used to use this, but the team didn't adapt to it well so the account was cancelled :(
– Curtis Aug 6, 2010 at 13:15

0

Overestimation is rather better than underestimation. That's because we don't know the "unknown" and (in most cases) specifications do change during the software development lifecycle.

In my experience, we use iterative steps (just like in Agile Methodologies) to determine our timeline. We break down projects into components and overestimate on those components. The sum of these estimations used and add extra time for regression testing and deployment, and all the good work....

I think that you have to go back from your past projects, and learn from those mistakes to see how you can estimate wisely.

Share Improve this answer

Follow

answered Aug 6, 2010 at 12:26

Buhake Sindi
89.1k ● 30 ● 174 ● 232