Feature Articles

How long will this take, anyway?

Discuss this article


By timmac & rhudson
TopCoder Members

It's Friday afternoon, about an hour before the end of the day. Monday morning is a national holiday, so there is a long weekend in your future — until the hot coffee breath on the back of your neck makes you think otherwise. It's the boss.

"Congratulations. You just logged hour 300 on the 200 hour project," he says. "When are you going to finish this thing?"

At this point, you have no idea how much longer it will take; honestly, if the parameters keep changing it could be anywhere from 3 weeks to 3 months. The boss -- an accountant by training -- won't accept such imprecision. You're tempted to placate him by blurting out "another 20 hours or so," but this kind of "shoot from the hip" estimate is what landed you in this mess to begin with.

The project -- a web-based "what-if" financial application -- seemed straightforward enough at first. Everyone, from analyst to manager to developer, had the same understanding when you started working. Or so you thought. You didn't know at the time that the "final" requirements on which you based your estimate still needed management approval. In the meantime, one of the project stakeholders made a few "minor modifications" to the design. "If I can make this change to a spreadsheet in five minutes," he says, "it shouldn't take an expert like you more than five seconds to do it in a web application."

Ultimately, you're completely stressed and your boss is frustrated. You're going to have to explain to your significant other -- again -- why you have to work all weekend. You wonder if there's any way to contain this situation, or at least avoid it in the future.

This scenario, based on real events, is all too common in today's world. As a developer, especially a contractor, you have to understand both the client's business, how that translates into requirements, and how many of those requirements can be implemented in a particular amount of time. And while you're at it, you need to train the client to provide specific descriptions of business processes… and appreciate the effort involved in creating an application… and understand that it's a cooperative process… and fit the entire effort into a project plan with milestones that you can meet.

It's not easy. Most of us either throw out a guess that we "think" will give us enough time, or invest hours using Microsoft Project to work backwards from a deadline date. Or, we try to "do it right" by spending hours poring over books on UML and Agile Development, which eats into development time (not to mention the hours you spend trying to explain how a "business rule" is different from a "business requirement").

Though you can never get a "perfect estimate," there are several techniques you can use to get "closer" and reduce the risk of the never ending software project. In the process of developing your timelines, you can build a partnership with your client, and initiate a cooperative, flexible process that will enable you to produce useful software.

The following five practices, combined with a small set of documents, make the software development process much more efficient.

1. Use a simple tool to document your estimate.
2. Keep a standard set of software development tasks handy.
3. Prepare multiple levels of estimation detail.
4. Estimate the smallest possible meaningful deliverable.
5. Identify what you're "waiting for" from the client.

Managing development tasks in Microsoft Project quickly becomes an hours-long task in itself. In small projects, a simple Excel spreadsheet provides plenty of useful information for both developer and manager. In his article "Painless Software Schedules," well-known software pundit Joel Spolsky describes an Excel approach to manage specific tasks by capturing essential information like Feature, Task, Priority, Estimated Time, Elapsed Time, and Remaining Time (http://www.joelonsoftware.com/articles

/fog0000000245.html). As the spreadsheet is updated, clients can follow a project's progress, anticipate next steps, and track how much time is remaining.

For this approach to be useful, you need to provide an exhaustive list of project tasks. This gives the client a realistic idea of what a particular job involves, and gives the developer a sense of comfort in having "all bases covered." We typically copy a standard set of these tasks, when possible, into all new project schedules. For example, when developing a web screen, we show the following breakdown:

1. Define requirements
2. Define scenario
3. Develop screen prototype
4. Develop underlying presentation layer objects
5. Connect presentation objects with business tier
6. Code web screen interface
7. Develop field validations
8. Integrate screen with application
9. Unit test screen
10. Ensure validation with requirements

We supplement this generic model with screen-specific tasks, like integrating extended client-side functionality, or providing "live" AJAX communication. At the very least, we've captured standard tasks, included unit test time, and provided some "cushion" for doing the work. The client gains an appreciation for the steps involved in making "a simple web screen."

In some cases, however, this level of detail may appear intimidating. In these instances, we create a "rolled up," or "condensed" schedule that summarizes detailed tasks into higher-level options. For example, our web screen might include only three tasks:

- Develop requirements
- Develop screen prototype
- Code screen

You should keep the detailed version for your own estimates, however - and make sure that the hours match in each version.

While you're estimating tasks for a project, try to divide them into multiple short-term deliverables. It's easier to work with a 40-hour phase that runs 5 hours over the estimate than a 200-hour project that is 100 hours late. With shorter time periods, you'll be able to identify underlying changes to logic and data structures that can be adjusted for future deliveries. To divide the work, help the client create basic scenarios, or use cases. Then, prioritize these sequentially into a schedule. For example, in a financial web system, you might create analysis and report screens first, and then add data entry later.

To facilitate the scenario-building process, we use data flow diagrams (DFDs) and generalized use cases. The DFD is useful to identify sources of information, and to identify who is responsible for providing that information. This document also serves as a scoping vehicle, as it clearly shows data that is internal to the system and excludes communication between outside contributors. For example, your system may include summarized financial data from Accounting, but might not include the intermediate calculations and negotiations between Accounting and Marketing.

Our Use Cases follow a very loose format. We ask the clients to provide a narrative, or a "story" describing how they might use a particular part of the system. For example: "I want to pick a year and a department, and see a list of amortized costs for that department over a period of 12 months." Then, we request a detailed list of inputs, outputs, and processes that support the story. We often derive requirements and validation rules from these basic use cases.

A final guideline to estimating is to constantly update and maintain a "Waiting For" list that contains any dependencies, including information you need from the client and technical access to systems. Be firm about what you absolutely cannot accomplish until this condition is satisfied — you can't start loading data for a scenario if you don't have database access, for instance — and don't assume clients are aware of these limitations. Document each limitation and spell out precisely which tasks are "blocked" while you're waiting. If you can, indicate the parties responsible for these dependencies. If you don't know, ask. In some cases, it's best to request

that your direct manager "motivate" the responsible parties. As a developer or contractor, you may have limited influence and simply cannot stimulate efficient action.

Though it only takes a few seconds to fire off a quick answer to "how long will this take?" the wrong easy answer could cost you a few hundred hours of unpaid hard work. Investing a few hours developing a plan, sharing the vision with the client, and gathering requirements and feedback dramatically reduces the risk of emergency overages. Remember these basics next time you're asked for an estimate:

- Keep plans simple, but complete. Remember that you'll have to maintain any documentation you produce.
- Estimate for small deliveries. This reduces your risk of going over time.
- Do not back down on your estimates. Lowering your estimate will not lower the amount of work involved.
- Be sure to budget the client's time as well, and encourage partnerships.
- Keep several methods in your back pocket when working with clients. If one method doesn't work, switch to another. Clients prefer the environment they understand best.

Just as you cannot build a house without knowing how many bedrooms it needs, you cannot build a system without the requirements. Similarly, a client cannot provide helpful information if they do not understand what is presented to them. Getting everyone on the same page will lead to projects that everyone can be happy with — which, down the road, could mean more business for you.