Developers, Can't Live With 'em, Can't Deprecate 'em

Posted At : June 20, 2017 11:20 AM | Posted By : Mark Kruger
Related Categories: Business Of Development

When last we checked in with Bob and the Muse (part 1 - applications) they were trying to come to grips with Bob's complex... I mean his application's complexity. We are now at the stage called "developer involvement" where he and the Muse are speaking with Sugar Sweet - the Muse's crack developer (and the name of a girl I dated in college). Let's go live to the conference call (be careful not to spook them).

- **Muse**: Ok, we've outlined changes to the onboarding and broken it out into 3 programming tasks. Sugar will get us an estimate but it seems straightforward.
- **Sugar**: Muse, a moment.
- **Muse**: Sure what is it sweetie? (Don't judge me - it's her nickname so it's not harassment)
- **Sugar**: I've taken a look at this application and the entire thing will need to be rewritten.
- **Bob**: Wait what? That sounds expensive.
- **Sugar**: I'm afraid we have no choice Bob. It's written in tags and the variable naming is all wrong. Plus the previous developers used... sorry, this is hard, I'm getting a little choked up... cfqueries instead of stored procedures. [small sob]
- **Muse**: I'm going to step in here while Sugar get's a drink of water. We will work to retain as much of the legacy code as we can and focus only on security issues Bob. In other words we *Won't* refactor your code unless it's necessary to protect your investment. How does that sound.
- **Sugar**: [sounding a little hurt] But the queries... and our best practice guide!
- **Muse**: I think we can work with what we have here...
- **Sugar**: This is so hard. I feel beet down. I'm not sure I cane work here anymore.

Developers come in all stripes and their views will greatly impact your estimate. Some of them have a religious devotion to certain technologies, libraries and approaches to development. To *be* a developer (at least on *my* staff) requires high aptitude, intelligence, breadth of knowledge and confidence. Those traits generate towering personalities like the Muse (whose lack of modesty is his only real flaw), and of course egos. They have opinions and they are usually not afraid to use them. Their defensive ferocity increases exponentially the closer you get to their favorite technology. If you don't believe me try yelling "Windows rules" at an IOS conference (and hope it's not an open carry state). For this reason, it's always a good idea to have non-developers involved in your estimates. Around here we call them project managers. They use their knowledge of the developer and the customer to "find the sweet spot" that meets the needs of the application and its stakeholders.

Muse Wisdom: Estimating is usually a joint effort.

Of course, not every estimate needs the once over by a project manager, but larger projects or enhancements should always be a collaboration. Developers need to be able to *justify* the estimates they provide to someone who knows them well enough to understand how they might get tripped up. Here are a few "types" of developer estimators.

The Refactorer

Sugar is in this camp. She will like over-estimate every task on older code because she sees any legacy code as debt that needs to be paid. To her this isn't optional, it's essential. In the real world it's definitely optional. Is script better than tags - in most cases yes (don't email me). Is using jQuery better than custom JS libraries? I believe it is. Are stored procs better than queries? Usually they are (though I've seen some god-awful stored procs). Best practices, frameworks, indentation, file naming conventions - these are all important parts of competent development. But when approaching legacy code there is a lot to consider. For example, has the customer managed to get a return on his initial investment? If he has, he may be ready for a rewrite. If not, we may need to *work with what is there* and not break the bank rewriting huge swaths of code. The business decisions rule the day. It's important for developers to understand the economics in play.

The Blithe Underestimator

Estimating is about imagining what *could* happen. What if the user doesn't check the box - what happens then? What happens when a user doesn't follow your pattern and puts in letters where numbers are indicated? Most developers react to these conditions with frustration. Who would do that? Why would a user use the application in *that* way when it's clearly designed to be used *this* way. A good estimator considers all these what-ifs. Estimating is also about accounting for what you still don't know. It's about challenging your assumptions. The underestimator concentrates on what he sees in front of him. His estimate comes with the caveat "if all my broad assumptions are correct". A good project manager will know the developer well enough to A) challenge him on his assumptions and B) add hours to the final estimate to insure it's adequate.

The Over-Complicator (Chicken Little)

Some developers are the opposite of the blithe underestimator. Instead of broad assumptions they build a huge list of unknowns and caveats. How do you know if you are an overcomplicator? Here's a little test for you. Let's say uou pull up Google and the page fails to load. Order the following list by probability.

1. Google is down
2. The entire internet is down
3. It's the zombie apocalypse
4. You've typed in "gloogle.com" accidently
5. Your wireless connection is bad
6. The network is under attack
7. Your mom changed the wireless password again (for those you working in your basement)
8. The NSA is monitoring your computer
9. DNS is not responding

If anything but number 4, 5, 7 or 9 is near the top of your list you may be an over complicator.

Some time ago I was helping a Muse reader with an error via screen share. On the screen the error debug information had query code and indicated a malformed query. I asked what he had tried. "I tried a new DB driver, checked the network connection and I've been googling how cached execution plans work." I took a closer look at the *debug output on the screen*, highlighted a portion and said "You are missing a comma after this column." A good manager will know his devs well enough to spot an over estimator and adjust estimates accordingly.

The Technology Hammerer

Finally, there is the developer for whom everything is about tech. The saying goes that when your only tool is a hammer everything looks like a nail. Developers need to be reminded that what they are solving are behavior problems, not (typically) technical problems. When we build a new search engine for a company selling nuts and bolts, the problem we are solving is not faster indexing, better sorting, or more granular pattern matching. nope. It's the basic problem of folks rummaging through bins trying to figure out thread and length. We are trying to make that task easier.

The tech hammerer will turn to the technical aspects of a project over and over because that is her comfort zone. She will suggest products, approaches and solutions that may or may not solve the basic problem underlying the project.

Tech hammerers usually need to learn that it is highly beneficial to *gather domain knowledge* about your customer. What does the company do? *How do they make money?* (that's the most important question for a dev company like ours) What sort of problems do *they solve* with their application? These questions inform our decisions in ways that mere technical questions cannot. Moreover, they tend to make us better partners - better at suggesting changes that save or make money for the customer. "What if we chose to do it this way Bob, would that save your users a step? Do you really need this field, it seems like we have this covered with earlier data. Explain how this will help your user - I want to understand."

Conclusion

The main takeaway here is that developers and their personalities have an impact on your estimates. Estimating is a team effort and requires give and take from several sectors. In our next post we are going to throw customers under the bus - gently of course.