the informatizer
volume 6 issue 10
The Software Industry Cheat Sheet
Published by Stout Systems
StoutSystems.com
877.663.0877

Software Estimation

This article should take about 10 to 1200 minutes to read.

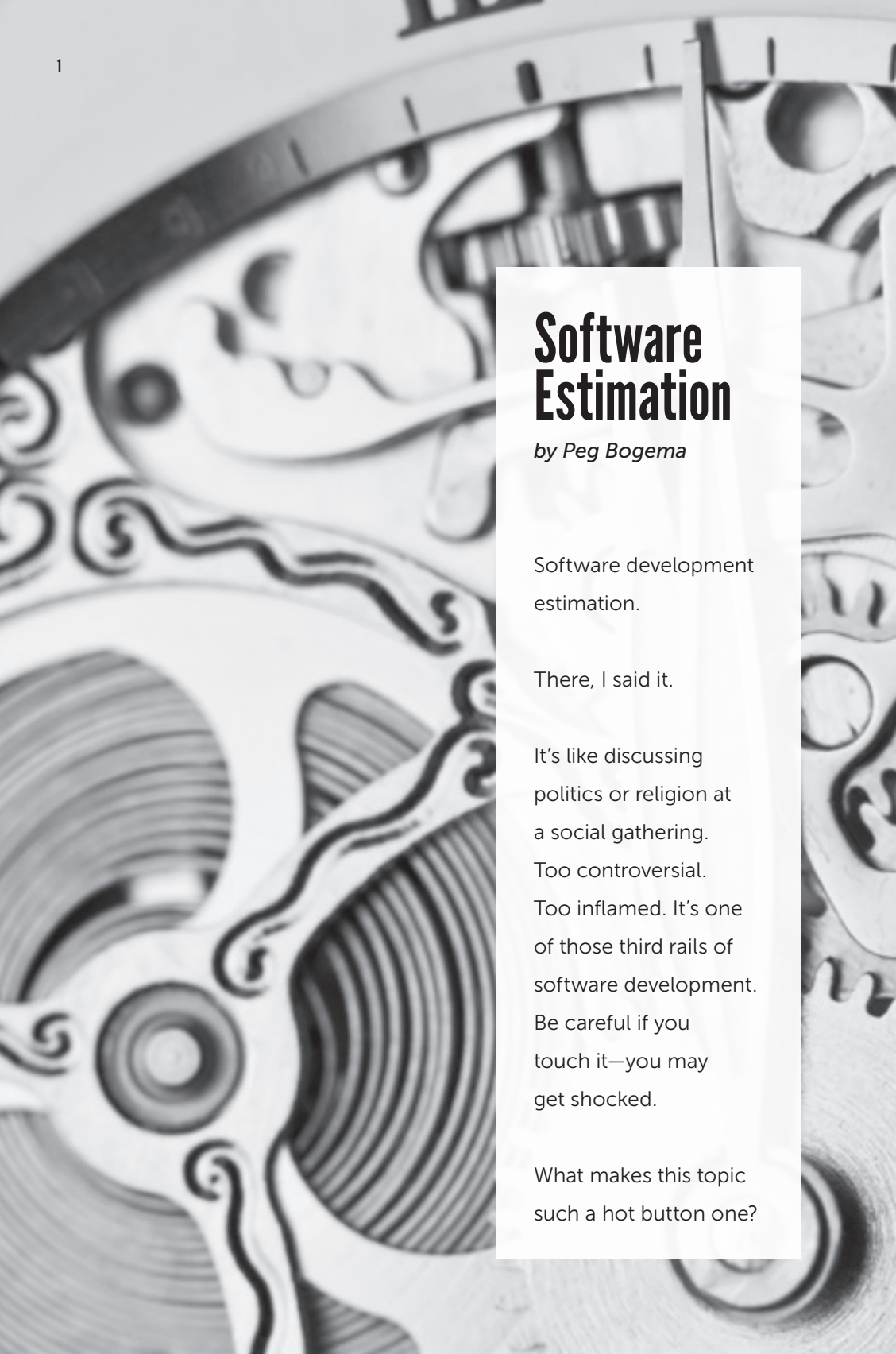
PAGE 1

If It's on Your Resume...

...They Will Ask You About It.

PAGE 9

linkd.in/stoutsystems
twitter.com/stoutsystems
facebook.com/stoutsystems



Software Estimation

by Peg Bogema

Software development estimation.

There, I said it.

It's like discussing politics or religion at a social gathering. Too controversial. Too inflamed. It's one of those third rails of software development. Be careful if you touch it—you may get shocked.

What makes this topic such a hot button one?

For one thing, it seems like developers are really bad at it. Ask for an estimate on a feature, get the answer that it will be done by the end of the week, and discover that two weeks later it's still a work in progress. Or ask a developer for an estimate and the answer is, "I don't know. Two to two hundred hours." What are you supposed to do with that?

For another thing, developers seem to resent being asked for estimates. Whether they are or aren't good at estimating, they seem to find the whole activity to be a time waster. Something that is a useless exercise that just gets in the way of doing productive work.

Let's look at a few things that the software development manager can do to transform the team culture into one that appreciates the value of estimating and is actually quite good at it.

First—and probably foremost—the software development manager needs to determine how many actual coding hours the team members have per week. At first blush, you might say, why, 40, of course! But this is a common misconception that has to be addressed.

Team members all have commitments that take them away from writing code. Just a 15 minute stand-up meeting every day takes more than an hour per week out of coding time. What about status reporting? What about team meetings? What about mandatory documentation? Mentoring junior developers? Reviewing code? And what about developers who are pulled off their main line of work to support earlier releases, other products or end users? An extremely efficient team gets 35 coding hours per week. 32 coding hours per week is still pretty good. But it's not unusual for a coding team—or key members of it—to have only 20 actual coding hours per week on their main project.

An extremely productive exercise is to have every member of the team track every task for every minute of the day. Is it coding? Or is it something else? Done during

a week or two weeks that seem to be pretty routine, the software development manager can roll up the results and determine just how much actual coding time the team has.

A two-week sprint for four developers might be viewed as 80 hours X 4 developers = 320 hours available for coding. But if the team is actually only coding for 32 hours per week, you have 64 X 4 developers = 256 hours. That means that developers are going to be off their estimates by a full 64 hours each and every sprint. No wonder they feel harried.

The manager has to impress upon the team that estimation is a way to manage workloads.

Once the coding throughput is truly and well understood, the team can select work scope that fits within the actual time available in the sprint or time period allocated for the next release.

Another situation where estimating goes off the rails and into the weeds is when the feature set that is being estimated is too large. There are multiple reasons why this is a point of failure. For one, a large feature set is rarely defined fully as to requirements and design. A slide on a PowerPoint with an image and some bullet points usually doesn't provide enough detail to give good estimates.

When features are large, WAGS (google it if you don't know what it means) should be used. Have the most experienced estimator on the team ballpark the effort on the feature with a large margin of error. That permits stakeholders to decide on the value, timing and so forth of the feature in the broadest of terms. But it doesn't hold the team to a number in the same way that an estimate does.

It's usually easy to estimate small features and tasks. The requirements are relatively simple to grasp. If they aren't highly defined, a Q&A session will usually flush out all the details.

If your next chunk of work is a large feature than cannot be broken down

into smaller feature subsets, then there is an additional step that the software development manager can do. Provide the team with an initial period of time or block of hours during which the initial highest risk areas can be prototyped or implemented, the requirements can be sorted out, etc. Then take that work and use it as the springboard for the final estimates. This tackles two problems—the unknowns, undefineds and risks are out of the way and the team is now very familiar with the remainder of the work. The tasks can be broken down at a very granular level, estimated and then rolled up into an overall statement of scope.

Estimates are always going to be more accurate when the requirements are well defined. When requirements are poorly defined, there is a greater likelihood of what looks like scope creep. Permitting enough time to hammer out the requirements is optimal. When that isn't possible, using a risk multiplier on the estimates makes sense. For example, an educated guess about the requirements might suggest that the task is five days of work. But unknowns could double the time on the task. So the risk multiplier is a factor of two and the task is estimated at five to ten days. This practice should only be applied where the requirements are agreed by everyone to be hazy. Where requirements are clear, estimates should not have such a high risk multiplier.

With estimates in place, developers then need infrastructure to track their estimates versus actuals. Tracking is the only way that a feedback loop can help the team refine, refine, refine on estimating. And when estimates are wildly off, that feedback can identify a problem in the overall software development process that can then be identified and fixed. Requirements weren't clear? Requirements were changed? Technical assumptions were made that were proven false? People were pulled off to do other things? All of these can be remedied, provided the manager actually knows what the problem was.

As a final point, the software development manager must

remember that for the team to meet its estimates, he or she must then prevent other things from encroaching on their actual coding hours. Developers appreciate this kind of shielding by their manager. It breeds fantastic loyalty. Because there is nothing more frustrating than missing a deadline and appearing to be incompetent, lazy or uncommitted. And there is nothing more rewarding that meeting a deadline and delighting stakeholders.

One other important point about estimating. Less experienced developers are usually horrible at estimating. They don't have enough experience to break a feature down into tasks (problem one). And then when someone else helps them with this, they really don't have any frame of reference for determining how long the tasks will take (problem two).

Have a more experienced developer create an implementation plan for the more junior developer. Have the junior developer estimate it and then track estimates versus actuals. Over time, the junior developer becomes familiar with tasks that are similar in size and complexity and can use prior experience to provide meaningful estimates. But without tracking the actual time, the junior developer never develops the skill. He or she is simply never introspective enough about how long a task took to glean insight that can be used to inform estimating on future features and tasks.

The software development manager has to impress upon the team that estimation is a way to manage workloads, to ensure that release scope is attainable within a given time period, and to keep management, sales and other stakeholder expectations set properly. The software development manager must discourage a culture that punishes for missing estimates. And, yes, the software development manager must also quietly recognize that a developer who is missing estimates time after time after time probably needs training or other intervention to speed up development time and quality.

Peg Bogema is Stout's Vice President of Operations. With Stout since 1997, her duties include the supervision of development projects and personnel, recruiting, and accounts.

Accidental Software Development Manager Workshops Deliver Value

On September 30th, Stout presented the third in a series of workshops designed for the "Accidental or Reluctant Software Development Manager." The workshop, "Accurately Estimating Software Projects," was our most successful event in the series based on feedback from attendees.

J.R., a director of software development said, "I have found the Stout seminars to be very informative. I appreciate the comprehensive yet unbiased presentation of each of the topics."

D.R., a senior software engineer said, "I have really enjoyed the seminars provided by Stout for the Reluctant Software Development Manager. Really good information presented by highly experienced people!"

John W. Stout, Founder and President of Stout Systems said, "Our goals in developing these workshops is to give back to the technical community and to provide the unique opportunity to learn from our most seasoned technical leadership. Our surveys indicated that the task of accurately estimating software projects was the most troublesome issue faced by Software Development Managers. So the Stout team put their heads together and formulated an event that we believed would offer significant value in addressing this issue. I'm proud to say that, based on the overwhelming response of attendees, we thoroughly achieved our goals and will be offering additional workshops in the near future."


Software budgets are sometimes reasonable, often challenging, and **frequently wasted.**

At Stout, we'll work *with* you to find the software solutions that fit you best—both technically and financially.

StoutSystems.com

The unknowns, the
undefineds, and the risks.

No, it's not the name of
a band. It's the day-to-day
of software development.

And Stout has done it
for over 20 years.
We can handle it.

Call Now
877.663.0877
StoutSystems.com

If It's on Your Resume...

by Brian Skory

I recently read an HR article called "How to Tell When a Candidate is Lying on His or Her Resume." I thought this title was a bit harsh. I've dealt with hundreds and hundreds of candidates—and although some have leaned toward exaggerating their skills or experiences a bit (a practice I don't recommend) I rarely felt that someone was outright lying. It did bring to mind an important point, however: "If you put it on your resume, they will ask you about it."

Recently, I've had several candidates go through an interview where they were asked to speak to a particular project they had been involved in or asked a couple of technical questions about a software language they had listed, and their response left the interviewer questioning that person's actual experience in that area.

I don't believe that any of those developers lied about what they put on

their resume—nor do I think that they even exaggerated their experience or involvement. In each of these instances, I truly believe it was simply a case of not being prepared to speak to that skill or experience.

As I've often said over the years, "An ounce of preparation is worth a pound of 'saving face' when it comes to interviewing." So what, specifically, is that "ounce of preparation?" A couple of things come to mind.

First, when preparing your resume, always be repeating to yourself, "If I put it down, they will ask me about it." Obviously, they might not...but smart interviewees don't take that chance.

Second, take the time to put yourself through a mock interview. Every person I know who was dead serious about getting a job, took the time to do this.

The steps are simple:

- Put yourself in the shoes of the interviewer and then look at your resume.
- Write down the 10 or 20 questions you might expect your interviewer to ask.

  If you're not sure what those questions might be, simply google "common questions asked about your resume."

- Find a friend who will ask you those questions while you answer them back.

  It doesn't matter a bit if your friend doesn't know the difference between C Sharp and B Flat. The benefit comes from a live person asking you a question that then requires you to formulate your thoughts and articulate an answer back.

If you commit to this minimal upfront investment, you certainly will give a better interview—potentially even tipping the balance in your favor in the event of a close race.

Brian Skory is Stout's Technical Talent Manager. He began his technical career 20 years ago and the key to his success has been his consistent ability to find and hire the best talent.