GILES BOWKETT

THE CRACK OF DOOM ON THE HYDROGEN JUKEBOX

WEDNESDAY, SEPTEMBER 17, 2014

Why Scrum Should Basically Just Die In A Fire

Conversations with Panda Strike CEO Dan Yoder inspired this blog post.

Scrum, the Agile methodology allegedly favored by Google and Spotify, is a mess.

Consider story points. If you're not familiar with Scrum, here's how they work: you play a game called "Planning Poker," where somebody calls out a task, and then counts down from three to one. On one, the engineers hold up a card with the number of "story points" which represents the relative cost they estimate for performing the task.

So, for example, a project manager might say "integrating our login system with OpenAuth and Bitcoin," and you might put up the number 20, because it's the maximum allowable value.

Wikipedia describes the goal of this game:

The reason to use Planning Poker is to avoid the influence of the other participants. If a number is spoken, it can sound like a suggestion and influence the other participants' sizing. Planning Poker should force people to think independently and propose their numbers simultaneously. This is accomplished by requiring that all participants show their card at the same time.

I have literally never seen Planning Poker performed in a way which fails to undermine this goal. Literally always, as soon as every engineer has put up a particular number, a different, informal game begins. If it had a name, this informal game would be called something like "the person with the highest status tells everybody else what the number is going to be." If you're lucky, you get a variant called "the person with the highest status *on the dev team* tells everybody else what the number is going to be," but that's as good as it gets.

Wikipedia gives the alleged structure of this process:

- *Everyone calls their cards simultaneously by turning them over.*
- *People with high estimates and low estimates are given a soap box to offer their justification for their estimate and then discussion continues.*
- *Repeat the estimation process until a consensus is reached. The developer who was likely to own the deliverable has a large portion of the "consensus vote", although the Moderator can negotiate the consensus.*
- *To ensure that discussion is structured; the Moderator or the Project Manager may at any point turn over the egg timer and when it runs out all discussion must cease and another round of poker is played. The structure in the conversation is re-introduced by the soap boxes.*

In practice, this "soap box" usually consists of nothing more than questions like "20? ***Really?***". And I've never seen the whole "rinse and repeat" aspect of Planning Poker actually happen; usually, the person with lower status simply agrees to whatever the person with higher status wants the number to be.

In fairness to everybody who's tried this process and seen it fail, how could it *not* devolve? A nontechnical participant has, at any point, the option to pull out an egg timer and tell technical participants "thirty seconds or shut the fuck up." This is not a process designed to facilitate technical conversation; it's so clearly designed to limit such conversation that it almost seems to assume that *any* technical conversation is inherently dysfunctional.

It's ironic to see conversation-limiting devices built into Agile development methodologies, when one of the core principles of the Agile Manifesto is the idea that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation," but I'll get to that a little later on.

For now, I want to point out that Planning Poker isn't the only aspect of Scrum which, in my experience, seems to consistently devolve into something less useful. Another core piece of Scrum is the standup meeting.

You probably know this, but just in case, the idea is that the team for a particular project gathers daily, for a quick, 15-minute meeting. This includes devs, QA, project manager(s), designers, and anyone else who will be working to make the project succeed, or who needs to stay up-to-date with the project's progress. The standup's designed to counter an even older tradition of long, stultifying, mandatory meetings, where a few people talk, and everybody else loses their whole day to no benefit whatsoever. Certainly, if you've got that going on in your company, anything which gets rid of it is an improvement.

However, as with Planning Poker, the 15-minute standup decays very easily. I've twice seen the "15-minute standup" devolve into half-hour or hour-long meetings where everybody stands, except for management.

faq | review

TRINKETS FOR YOUR AMUSEMENT

Hacker Newspaper
minim.al bit.ly
Clueful Google
BTC in USD

STUFF I DREW

At one company, a ponderous, overcomplicated web app formed the centerpiece of the company's Scrum implementation. Somebody had to sit to operate this behemoth, and since that was an informal privilege, it usually went to whomever could command it. In other words: management.

At another company, the Scrum decay took a different route. As with the egg timer in Planning Poker, Scrum standups offer an escape clause. In standups, you can defer discussion of involved topics to a "parking lot," which is where an issue lands if it's too complex to fit within the meeting's normal 15-minute parameters (which also include some constraints on what you can discuss, to prevent talkative or unfocused people from over-lengthening the meeting).

At this second company, virtually everything landed in the parking lot, and it became normal for the 15-minute standup to be a 15-minute prelude to a much longer meeting. We'd just set the agenda during the standup, and the parking lot would be the actual meeting. These standups typically took place in a particular person's office. Since arriving at the parking lot meant the standup was over, that person, whose office we were in, would feel OK about sitting down in their own, personal chair. But the office wasn't big enough to bring any new chairs into, so everyone else had to stand. The person whose office we were always in? A manager.

Scrum's standups are designed to counteract an old tradition of overly long, onerous, dull meetings. However, at both these companies, they replaced that ancient tradition with a new tradition of overly long, onerous, dull meetings *where management got to sit down, and everybody else had to stand*. Scrum's attempt at creating a more egalitarian process backfired, twice, in each case creating something more authoritarian instead.

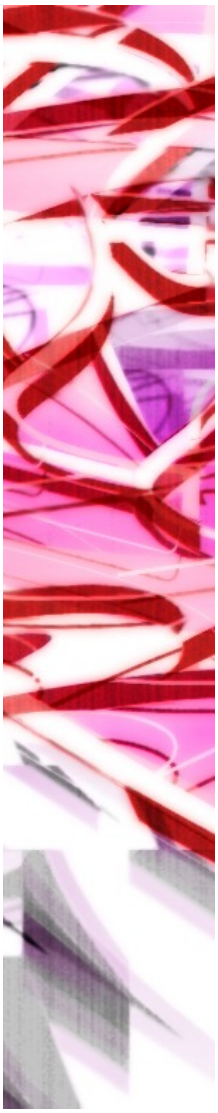
To be fair to Scrum, it's not intended to work that way, and there's an entire subgenre of "Agile coaching" consultants whose job is to repair broken Scrum implementations at various companies. This is pure opinion, but my guess is that's a very lucrative market, because as far as I can tell, Scrum implementations often break.

The Adobe Flash Player or an HTML5 supported browser is required for video playback.
Get the latest Flash Player
Learn more about upgrading to an HTML5 browser

I recommend just skimming the first few seconds of this.

Scrum's ready devolution springs from major conceptual flaws.

Scrum's an Agile development methodology, and one of its major goals is sustainable development. However, it works by time-boxing efforts into iterations of a week or two in length, and refers to these iterations as "sprints." Time-boxed iterations are very useful, but there's a fundamental cognitive dissonance between "sprints" and "sustainable development," because there is no such thing as a



sustainable sprint.

This man's pace is probably not optimized for sustainability.

Likewise, your overall list of goals, features, and work to accomplish is referred to as the "backlog." This is true even on a greenfield project. On day 1, you have a backlog.

Another core idea of the Agile Manifesto, the allegedly defining document for Agile development methodologies: "working software is the primary measure of progress." Scrum disregards this idea in favor of a measure of progress called "velocity." Basically, velocity is the number of "story points" successfully accomplished divided by the amount of time it took to accomplish them.

As I mentioned at the top of the post, a lot of this thinking comes from conversations with my new boss, Panda Strike CEO Dan Yoder. Dan told me he's literally been in meetings where non-technical management said things like, "well, you got through [some number] story points last week, and you only got through [some smaller number] this week, and *coincidentally*, I noticed that [some developer's name] left early yesterday, so it looks pretty easy who to blame."

Of course, musing, considering, mulling things over, and coming to realizations all constitute a significant amount of the actual work in programming. It is impossible to track whether these realizations occur in the office or in the shower. Anecdotally, it's usually the shower. Story points, meanwhile, are completely made-up numbers designed to capture off-the-cuff estimates of relative difficulty. Developers are explicitly encouraged to think of story points as non-binding numbers, yet velocity turns those non-binding estimates into a number they can be held accountable for, and which managers often treat as a synonym for productivity. "Agile" software exists to track velocity, as if it were a meaningful metric, and to compare the relative velocity of different teams within the same organization.

This is an actual thing which sober adults do, on purpose, for a living.

"Velocity" is really too stupid to examine in much further detail, because it very obviously disregards this whole notion of "working software as a measure of progress" in favor of completely unreliable numbers based on almost nothing. (I'm not proud to admit that I've been on a team where we spent an entire month to build an only mostly-functional shopping cart, but I suppose it's some consolation that our velocity was acceptable at the time.)

But, just to be clear, one of velocity's many flaws is that different teams are likely to make different off-the-cuff estimates, as are different members of the same team. Because of this, you can only really

garner anything approaching meaningful insight from these numbers if you compare the ratio of estimated story points to accomplished story points on a per-team, per-week basis. Or, indeed, a per-individual, per-week one. And even then, you're more likely to learn something about a team's or individual's ability to make ballpark estimates than their actual productivity.

Joel Spolsky has an old but interesting blog post about a per-individual, velocity-like metric based on actually using math like a person who understands it, not a person who regards it as some kind of incomprehensible yet infallible magic. However, if there's anything worth keeping in the Agile Manifesto, it's the idea that working software is the primary measure of progress. Indeed, that's the huge, hilarious irony at the center of this bizarre system of faux accountability: with the exception of a few Heisenbugs, engineering work is already inherently more accountable than almost any other kind of work. If you ask for a feature, your team will either deliver it, or fail to deliver it, and you will know fairly rapidly.

If you're tracking velocity, your best-case scenario will be that management realizes it means nothing, even though they're tracking it anyway, which means spending money and time on it. This useless expense is what Andy Hunt and Dave Thomas termed a broken window in their classic book *The Pragmatic Programmer* - a sign of careless indifference, which encourages more of the same. That's not what you want to have in your workplace.

Sacrificing "working software as a measure of progress" to meaningless numbers that your MBAs can track for no good reason is a pretty serious flaw in Scrum. It implies that Scrum's loyalty is not to the Agile Manifesto, nor to working software, nor high-quality software, nor even the success of the overall team or organization. Scrum's loyalty, at least as it pertains to this design decision, is to MBAs who want to point at numbers on a chart, whether those numbers mean anything or not.

I've met very nice MBAs, and I hope everyone out there with an MBA gets to have a great life and stay employed. However, building an entire software development methodology around that goal is, in my opinion, a silly mistake.

The only situation I can think of where a methodology like Scrum could have genuine usefulness is on a rescue engagement, where you're called in as a consultant to save a failing project. In a situation like this, you can track velocity on a team basis to show your CEO client that development's speeding up. Meanwhile, you work on the real question, which is who to fire, because that's what nearly every rescue project comes down to.

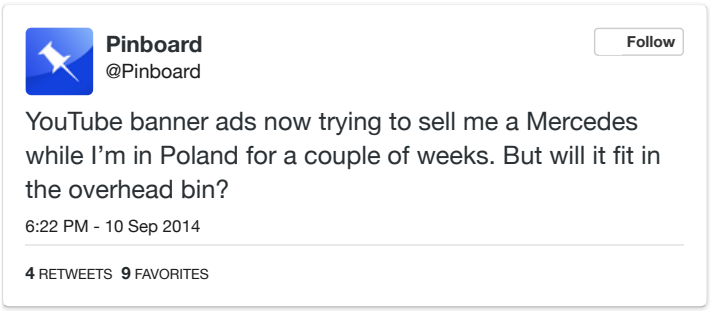
In other words, in its best-case scenario, Scrum's a dog-and-pony show. But that best-case scenario is

rare. In the much more common case, Scrum covers up the inability to recruit (or even recognize) *engineering talent*, which is currently one of the most valuable things in the world, with a process for managing engineers as if they were cogs in a machine, all of equal value.

And one of the most interesting things about Scrum is that it tries to enhance the accountability of a field of work where both failure and success are obvious to the naked eye - yet I've never encountered any similarly elaborate system of rituals whose major purpose is to enhance the accountability of fields which have *actual accountability problems*.

Although marketing is becoming a very data-driven field, and although this sea change began long before the Web existed at all - Dan Kennedy's been writing about data-driven marketing since at least the 1980s - it's still a fact that many marketers do ***totally unaccountable*** work that depends entirely on public perception, mood, and a variety of other factors that are inherently impossible to measure. The oldest joke in marketing: "only half my advertising works, but I don't know which half."

And you never will.



YouTube ads have tried to sell me a service to erase the criminal record I don't have. They've reminded me to use condoms during the gay sex that I don't have either. They've also tried to get me to buy American trucks and country music, neither of which will ever happen. No disrespect to the gay ex-convicts out there who do like American trucks and country music, assuming for the sake of argument that this demographic even exists, it's just not my style. Similarly, Facebook's "targeted" ads usually come from politicians I dislike, and Google's state-of-the-art, futuristic, probablistic, "best-of-breed" ads are worse. The only time they try to sell me anything I even remotely want is when I've researched something expensive but decided not to buy it yet. Then the ad follows me around every web site I visit for the next month.

"It Feels As If It's Reading My Mind"
New D&B Track: Maximum Bass
JavaScript Patterns For Contemporary Dance Music
New eBook: Software As A Disservice - Fixing A Bro...
iOS Fucked Up My Brain
▼ September (4)
Rubyists, A Non-Endangered Species
Nobody Will Ever Win
Dubstep/Chillout Track: Old City Streets
Privateers In The 21st Century
▼ August (8)
It's Not Militarization, It's Piraticization
Dave Winer Should Read This
Conspiracies Don't Have To Be Evil
The Innovation Of Loneliness: An Excellent Short F...
Book Sale Countdown (And Secret Project Prelude?)
My Simple Pricing Strategy For Information Product...
More Praise For My Ember Book
Lies And Propaganda
▼ July (13)
A System For Contorting And Relocating Cat-Shaped ...
New Dubstep Track: Otah-Kvo
Ember Book Table Of Contents
Real Metaprogramming: Experimental Businesses As A...
Doom Bell: Synthy Chillout Dubstep
My Perspective On Ember And Backbone
Picasso vs Stereotype
Feedback On My Ember Book
Free Animated Typeface / Free Typographic Animatio...
Good Talk Slides On Ember.js Architecture
Great Big Comic About Millenial-Bashing In

Please buy it. Please. You looked at it once.

Even in 2014, marketing involves an element of randomness, and probably always will, until the end of time.

Anyway, Scrum gives you demeaning rituals to dumb down your work so that people who will never understand it can pretend to understand it. Meanwhile, work which is genuinely difficult to track doesn't have to deal with this shit.

Why?

I don't think highly of Scrum, but the problem here goes deeper. The Agile Manifesto is flawed too. Consider this core principle of Agile development: "business people and developers must work together."

Why are we supposed to think developers are not business people?

If you join (or start) a startup, you may have to do marketing before your company can hire a marketing person. The same is true for accounting, for sales, for human resources, and for just about anything that any reasonable person would call business. You're in a similar situation if you freelance or do consulting. You're definitely in a better position for any of these things if you hire someone who knows what they're doing, of course, but there's a large number of developers who are also business people.

Perhaps more importantly, if you join or start a startup, you can knock the engineering out of the park and still end up flat fucking broke if the marketing people don't do a good job. But you're probably not going to demand that your accountants or your marketing people jump through bizarre, condescending hoops every day. You're just going to trust them to do their jobs.

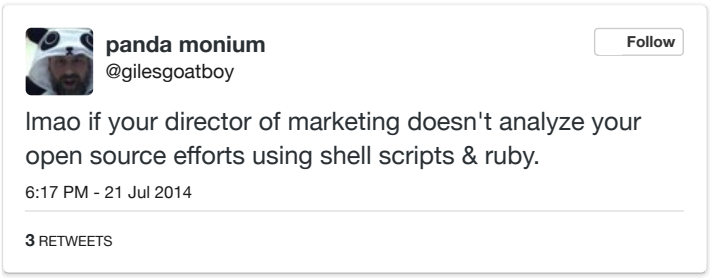
This is a reasonable way to treat engineers as well.

By the way, despite that little Dilbert strip a few paragraphs above, my job title at Panda Strike is Minister of Propaganda. I'm basically the Director of Marketing, except that to call yourself a Director of Marketing is itself very bad marketing when you want to communicate with developers, who traditionally mistrust marketing for various reasons (many of them quite legitimate). This is the same reason the term "growth hacker" exists, but as a job title, that phrase just reeks of dishonesty. So I went with Minister of Propaganda to acknowledge the vested interest I have in saying things which benefit my company.

However, despite having marketing responsibilities, my first act upon joining Panda Strike was to write code which evaluates code. I tweaked my git analysis scripts to produce detailed profiles of the history of many of the company's projects, both open source and internal products, so that I could get a very specific picture of how development works at Panda Strike, and how our projects have been built, and who built them, and when, and with which technologies, and so on.

As an aside, I first developed this technique on a Rails rescue project. It was the first thing I did on the project, but the CTO, having an arrogant and aloof attitude, had no idea. So on my first day, after I did this work, he introduced me to the rest of the team, telling me their names, but nothing else about them. But I recognized the names from my analysis of the git log. I noticed that the number one JavaScript committer had a cynical and sarcastic expression, that most of the team had three commits or less, and that the number one Ruby committer wasn't anywhere in the building.

This CTO who had told me nothing then said to me, "OK, dazzle me." As you can imagine, I did not dazzle him. I fired him. (Or, more accurately, I and my colleagues persuaded his CEO to fire him.)



Anway, the whole point of this is simple: there's absolutely no reason to assume that a developer is not a business person. It's a ridiculous assumption, and the world is full of incredibly successful counterexamples.

The Agile Manifesto might also be to blame for the Scrum standup. It states that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation." In fairness to the manifesto's authors, it was written in 2001, and at that time `git log` did not yet exist. However, in light of today's toolset for distributed collaboration, it's another completely implausible assertion, and even back in 2001 you had to kind of pretend you'd never heard of Linux if you really wanted it to make sense.

Well-written text very often trumps face-to-face communication. You can refer to well-written text later, instead of relying on your memory. You can't produce well-written text unless you think carefully. Also, technically speaking, you can literally never produce good code in the first place unless you produce well-written text. There are several great presentations from GitHub on the value of asynchronous communication, and they're basically required viewing for anybody who wants to work as a programmer, or with programmers.

In fact, GitHub itself was built without face-to-face communication. Basecamp was built without face-to-face communication as well. I'm not saying these people never met each other, but most of the work was done remote. Industrial designer Marc Newson works remote for Apple, so his work on the Apple Watch may also have happened without face-to-face communication. And face-to-face communication plays a minimal role in the majority of open source projects, which usually outperform commercial projects in terms of software quality.

In addition to defying logic and available evidence, both these Agile Manifesto principles encourage a kind of babysitting mentality. I've never seen Scrum-like frameworks for transmuting the work of designers, marketers, or accountants into cartoonish oversimplifications like story points. People are happy to treat these workers as adults and trust them to do their jobs.

I don't know why this same trust does not prevail in the culture of managing programmers. That's a question for another blog post. I suspect that the reasons are historical, and fundamentally irrelevant,

because it really doesn't matter. If you're not doing well at hiring engineers, the answer is not a deeply flawed methodology which collapses under the weight of its own contradictions on a regular basis. The answer is to get better at hiring engineers, and ultimately to get great at it.

I may do a future blog post on this, because it's one of the most valuable skills in the world.

Credit where credit's due: the Agile Manifesto helped usher in a vital paradigm shift, in its day.

POSTED BY GILES BOWKETT AT 6:07 PM

Newer Post Home Older Post