Project Velocity

ProjectVelocity is the measurement of the event rate of a project. This is useful for a number of things, including the determination of whether a project is slowing down, high or low productivity, and many other factors. It is especially useful as a so-called "canary in a coal mine" (DeadCanary); when used correctly, ProjectVelocity measurements can detect a minor downturn or a major pothole before other traditional means. -- DionHinchcliffe

---

I like this Idea. What's a good definition of "event" to count?

Things that come to mind include: number of releases to the configuration; number of class versions ... what else? -- RonJeffries

---

We use the abstract "estimated craft unit" as a measurement of progress. The important thing to us is to be able to predict how many iterations we'll need to finish the current release. So we estimate every story or task in terms of craft units (we assume 1 pair-day, with no meaningful interruptions, is 1 craft unit), and then plan an iteration based on how many estimated craft units we can do.

This lets us assign craft units to other (i.e. non-programming) tasks as they come in, and we get a good count of (a) how much "estimated work" we can do each week, and (b) how much of that work was spent on "other" jobs. Check out http://www.projectmetrics.com/Projects/XPTracker/index.htm for an example of how we track these things.

Since moving to craft units and tracking like this, we've seen the predictability of our long-term release plans jump; instead of being months off, we're now weeks or days off... -- JasonFredrickson

---

Seeing the discussion in HeroicProgramming, I think ExtremeProgramming's ShortIteration's give you an steady stream of events. If you have 153 stories to complete in 9 months, and you schedule some to be completed every three weeks, then every three weeks you have a new event.

It is easy to circumvent this kind of scheduling, for example by not having or ignoring FunctionalTest scores (which we did at ChryslerComprehensiveCompensation, costing us a four month slip right at the end). -- KentBeck

---

I definitely agree that the best event we have is the stack of cards getting shorter on the "to do" side and taller on the "done" side. I wonder, though, whether there are other velocity indicators that would help the Coach (I promoted myself) spot trouble. Wouldn't substitute for careful observation, but I have this fascination with technology. Kind of a hobby ... -- RonJeffries

- *How about rate of aggregation ProjectVelocity of unit tests? I think people tend to stop writing unit tests when trouble hits.*

A big change in the measured LoadFactor might be an indication? If you have been cruising along with 2.5 and suddenly you jump to 4, this is an indication that either people's ideal estimates are too small or they are spending lots of time doing other things. -- TimMackinnon

Is ProjectVelocity the same thing as (or a superset of) what is often called a project's "pulse", "heartbeat", or "rhythm" by folks like GradyBooch, SteveMcConnell, JimMcCarthy, and others? One might call this the "mean time between releases" where "releases" includes both formal and informal releases (internal as well as external build/integration milestones).

I don't think so. The heartbeat is the periodicity on the calendar (monthly releases, two week (TwoWeeks iterations, annual updates, or whatever). The ProjectVelocity is the amount of features that fit into one beat, estimated in some way that both customers and programmers are comfortable with. -- KentBeck

See TwoWeeks

With LoadFactor deprecated, what is a good starting velocity to assume? In a project that is transitioning to ExtremeProgramming, there hasn't been a measured velocity. So how do we lay out the initial ReleasePlan if we don't know how many stories can fit in an iteration? -- TimWoodard

Are you saying "How do I schedule if I don't know how long it will take?" Many would like the answer to that. From my reading, the initial velocity is a commitment of zero tasks, but you'll certainly complete some tasks on your first iteration. That gives you a true starting point.

I don't quite understand that. You are supposed to base your commitment for the next iteration upon the velocity of the previous iteration. But for the first iteration you have no previous velocity, so is there a rule that you use as a basis for picking an initial set of stories to implement?

Prioritize your stories and just do as many as you can in the first iteration. Don't cheat and try to maximize the number you get done: make sure you stick to CodeUnitTestFirst and so on. If the project is started from scratch (instead of ported from a different methodology), ProjectVelocity may be unstable at the beginning anyway as the team gains insight into different parts of the project. "As many as you can" will give you a rough estimate to use for your second iteration, which will give you a better estimate for the third iteration and so on.

"I think this is equivalent to what was meant above by "the initial velocity is a commitment of zero tasks". You start out with nothing done: congratulations! You have completed all of your expected zero work units for the first iteration! Now do what you always do when you have extra time left at the end of an iteration (and no refactoring, etc to do): take on additional tasks one by one until the iteration is over.'

What happens when you have lots of interdependencies between tasks? Surely that could make the velocity very discontinuous. For example, if you have nine tasks waiting to be done, but they all depend on a tenth task, as soon as the tenth task gets done you could have a sudden spike of output. You can't (or at least you shouldn't) use this spike to say things like "the project is speeding up".

Indeed you should not. Instead, you should use your skill and judgement. Other externalities can affect project velocity, too, both upwards and downwards. We keep a record of project velocity measurements over the last year's worth of iterations, along with the number of developer hours used, and so forth, so anomalies can be easily identified and taken into account.

SanderHoogendoorn advocates the use of UseCases as a UnitOfWork. To me this seems a sensible thing to do, but if you want to do this you have to limit your UseCases to the UserGoal and the subfunction level as defined by AlistairCockburn. Otherwise the amount of work needed to implement a single UseCase will vary too much. -- HuddieKlein

---

When do you cut your losses? Suppose you have a known deadline. The goal is to get as much done by that deadline. You suddenly discover that the deadline is right around the corner and you don't even have all of your stories pointed (in fact you may not have identified all of your stories). When do

ProjectVelocity, the PlanningGame and StoryPoints lose their effectiveness? -- WillSmith

---

Is there any concept of including Operational issues in these project timelines? When I started running ops for a major web site, the dev and pm team never thought about that they might need significant operational change to the site to implement their wonderful concepts. So we put ops representation at the table, early on. Database upgrades, disk expansion, and other ops planned upgrades were then negotiated (want to upgrade to the next version of Oracle? When's that going to happen in the timeline for features?) Is there some way to factor that into your equation based tool? Or is it just a "best practice" when starting the project in general? --al in pt

---

I've had the misfortune to work in an environment where there was a management decree that your personal ProjectVelocity must never drop below 0.5*ideal on pain of adverse employment action. I'm not entirely sure how this confusion of ideas occurred, but I suspect it was because they viewed ProjectVelocity as "percentage of time spent on-task" rather than as "estimate correction factor". I'm quite doubtful that percentage of time spent on-task is a valid way to measure employee effectiveness, but the subtext was that maintaining too low of a percentage was deemed StealingFromTheCompany.

Last edit July 27, 2010, See github about remodeling.