Measuring Effort and Productivity of Agile Projects

Measuring Effort

Measuring effort means the same thing in agile and non-agile methods: you need to know the hours spent by people working on the release.

There are two subtle issues, though:

a. Which people should track the effort they expend on the project?
b. Do you track effort based on the phases in the SLIM methodology?

Which people?

Agile methods refer to “the team”, which usually means the people who write code and run test cases. For the purpose of estimating and tracking effort, the product manager and scrum master are included. But there are other people that expend effort on the project: specialists who share time on multiple products, perhaps specialized DBAs or Enterprise Architects, documentation and training materials developers, testers from an enterprise test group that must pass on every delivered product and other such resources that contribute time and thus effort to a release. And don’t forget subject matter experts and other stakeholders who participate in backlog definition, backlog grooming, and sprint reviews. Although they are not full time on the team, the effort of all these people is critical and contributes to the cost of the release.

What phases?

The SLIM methodology divides a project into four phases. The term “phase” has a negative connotation for some people in the agile community. It’s the same word that’s used in “big upfront requirements phase” or “big upfront design phase” that is anathema to many agile methods.

But in the SLIM methodology, “phase” has a different meaning. The four phases in a SLIM project do not imply non-overlapping sequential activities. Instead, in the SLIM methodology, the phases represent different types of work that are carried out in a project, with different characteristics of how resources are used. The time periods during which those types of work are carried out can be configured for a particular project with what the SLIM method calls “phase tuning”. In agile projects, the key types of work that matter for effort tracking are work on requirements, which corresponds to Phase 2 in the SLIM methods, plus work on development and testing of the code and other needed materials, which corresponds to Phase 3. In an agile project, these types of work go on concurrently throughout the release, and SLIM phase tuning is configured to reflect that.

The SLIM methods rely on history of completed projects to calibrate estimates and forecasts, either from your own organization or industry wide data from the SLIM database. Effort is one of the key metrics. In the SLIM tools, you to track effort by phase, that is, by type of work.

The effort expended by team members and stakeholders on getting stories ready to develop, including writing the stories, grooming the backlog, and prioritizing should be tracked as effort expended in the Story Writing Phase (Phase 2 in the SLIM tools). Very importantly, this also includes the time spent on conversations among the stakeholders and development team to refine the details of the stories.

Effort expended on detailed design, developing the code, refactoring, testing, debugging and other related activities should be tracked in the Story Development Phase (Phase 3).

Since these activities occur concurrently, the tools you use to track effort may need to be used creatively to make this distinction. For example, a time tracking tool that just lets a team member track what hours each day were spent on the project may not split out the time, since there's no set distinction of which days and which hours were spent on story writing, and which days and hours were spent on story development. On any given day, work was likely done on both. Many agile tracking tools let team members track the time spent on a particular story by having a sub record of the Story for each team member to use for time tracking. Several participants in the Agile Estimation Round Table indicated the members of agile teams use such tools.

Most of these tools allow customization that would let team members to track two effort numbers for a story, splitting out the effort on story writing (including, most importantly, the conversation time about the details) and the effort on development.

Note that very precise time tracking is usually not practical, and making the distinction between story writing effort and story development effort makes that even more difficult. Fortunately, since SLIM develops trends from multiple projects, very precise tracking is not needed for the SLIM methods, so to be practical, team members can split the time they spend by their best guess of the percentage spent on story writing vs story development.

Tracking effort by story within the agile tracking tools is often not available to stakeholders who are not part of the core development team. In this case, a product owner may estimate the actual time spent by stakeholders on story writing (and development if applicable) and track that separately, perhaps as attributes on the story itself in the tracking tool.

No matter how you track the time, collating it and entering it in the SLIM tools can either be done manually, or may be automated through the use of the SLIM API (application programming interface) together with your tracking tool's api. Through such automation, and by appropriately sacrificing precision for ease of use, you can track the effort expended throughout an agile project on story writing and story development. This provides the historical basis for future estimates.

Measuring Productivity

Productivity in software development is surprisingly complicated to measure or even to define. We think we intuitively know what it means to be productive: a team can get a lot done in a short period of time. Or does it mean that a team can get a lot done without expending a lot of hours of effort? And what constitutes "getting a lot done"? Writing code? Testing the code? Efficiently defining requirements and building in quality so software is developed right the first time? With waterfall methods, producing an elegant, robust design may be very productive even though no actual code is running, but that's unlikely in an agile environment. And once you decide on what productivity means, how do you measure it?

Agile methods simplify some of these issues. Since working software is the measure of progress, and since the project progresses in time-boxed iterations, the amount of stories developed per iteration, known as the Team Velocity on a project, provides some measure of productivity. Many agile teams use Team Velocity for iteration planning, where "amount of stories developed" is the total Story Points for the stories completely developed in an iteration. The best guess of the Team Velocity you can achieve in the next iteration is that it will be about the same as the Team Velocity of the previous iteration, and that helps the team plan which stories

they can tackle in an upcoming iteration.

However, when you switch from Iteration Planning on a specific project to Release Planning for a portfolio of projects, Team Velocity becomes quite problematic as a measure of productivity. There are many reasons for this including:

1. **Velocity is not constant:** The Team Velocity from iteration to iteration will change fairly slowly, especially if iterations are short, which is why it's useful for iteration planning. But Team Velocity does change through the course of projects in a systematic way. The fundamental research in project estimation, pioneered by Larry Putnam Sr., the founder of QSM, showed that software development projects follow a basic shape, called the Putnam-Norden-Rayleigh curve, and there is an ebb and flow (or more precisely a "flow and ebb") to projects. Research by QSM verifies that this is true of agile projects as well as other methods. Most agile methodologists are implicitly aware of this: they suggest waiting a few iterations before starting to compute velocity because velocity typically starts low and grows for a while, then starts changing slowly (growing and then ebbing) in the middle of projects, and then, usually, velocity starts going down towards the end of a project as a release is "hardened". It's in that middle stage where the velocity changes slowly that it's useful for iteration planning. This "grow, change slowly for a while, reduce" pattern is typical of the Rayleigh shape of projects. The cumulative effect is a familiar "S-shaped" curve:

Agile Effort

Image not found or type unknown

In particular, many companies are using “agile at scale” methods, for example SAFe or other large-scale agile methods. Several examples from members of the Agile Round Table showed the distinctive Rayleigh pattern when you look at development by teams of teams.

2. **Velocity depends on team size:** While most agile methodologists and the results of QSM’s research recommend small team sizes, there’s no single “ideal” size, and team sizes will vary from project to project, in part depending on whether time pressure or cost is paramount. So the Team Velocity attained by a 5 person team on a project cannot be used to predict the Team Velocity of a project that may be

under more time pressure, so an 8 person team is used.

3. **Velocity depends on the iteration length:** If one team is working in two week iterations, and another team is working in three week iterations, their Team Velocities will be different, even though there may not be much difference in how long or how much effort comparable releases take.
4. **Velocity is not linear:** Remember "The Mythical Man Month"? The relationship between software size, duration of a project, and team size or effort expended, is not a simple linear relationship. A 10 person team, all else being equal, will NOT have twice the team velocity of a 5 person team. And the amount of software a fixed team can develop in 6 months is not twice what the same team can develop in 3 months (it's likely to be more than twice!), so the Team Velocity (even assuming the same length iterations) is different.

For these and other reasons, Team Velocity is not very successful as a predictive measure for Release Level estimation.

Instead of Team Velocity, the SLIM tools and methods use a more abstract measure of productivity called "Productivity Index". This is statistically derived from size, effort, and duration of completed projects, to produce a trend that compares Productivity Index to size of a project. This, when used with the Software Production Equation compensates for the non-linear nature of software development and allows the SLIM tools and methods to adjust the Rayleigh shape of a project based on tradeoffs among size, effort and schedule.

There are two ways to get a Productivity Index trend to use for Release Estimation:

a. You can use industry data. The SLIM tools come with a trend based on projects from multiple customers doing agile development.
b. You can use your own data.

Most agile development shops recognize the importance of keeping metrics. The key metrics for computing Productivity Index trends are size, effort, and duration of projects. In the articles in this series about size, we discussed that size measures in agile projects are often different for different companies. The computation of the trend for Productivity Index adjusts for that (as does Team Velocity, for that matter). So if you use your own data, the size and Productivity Index form a matched pair.

Measuring Effort and Productivity Are Related

The Team Velocity of a project is derived from size and duration, but the Productivity Index of a completed project is derived from size, duration, and effort through the SLIM Software Production Equation. That's why this measure of productivity can be used for release estimation, including alternative estimates that understand "the mythical man-month" and the non-linear tradeoffs between effort and duration. Therefore, the effort tracking discussed in the first part of this article is important for measuring productivity, including the split of effort between Story Writing and Story Development.

By measuring effort for completed projects as well as time and size, and using those metrics to derive a trend for the Productivity Index, you can meet business constraints with release estimates that can trade off cost and size of future projects, letting you plan how to best apply your resources to agile teams.