Alternative Sizing Units for Agile Estimation

Function Points for Agile Estimation

Function points have been used as a measure of size for estimation for many years. There are benefits and challenges to using function points as the unit of measure for agile projects. Note that function points share with story points the characteristic that there's no such thing as "one of them." While you assign function points to scope items and add those up to get a total, there's no formal definition of a single function point.

There are several reasons function point counting has been successful for sizing projects for estimation. This includes the specific way the scope is decomposed for function point counting that helps account for the degree of shared code. Perhaps the biggest benefit is that function point counting has an internationally agreed to standard, giving it the consistency needed for estimation. Of course, that's the reason the standard was developed and why many companies invest in training function point counters. Actually, there is more than one type of function point counting, and more than one standard, but for the purposes of this paper, they are similar.

One challenge several of the Agile Round Table members found when using function points for estimating agile projects was gaining acceptance from the team. Some agile teams considered function points "old fashioned" and felt that, because of that, it was not applicable to agile projects. Related to this is that, unlike story points, the agile team probably won't use the function point counts for any purpose beyond the release estimate, so it's not considered part of the team's agile methods. In particular, epics and stories do not correspond directly to the decomposition of a project's scope that's needed for function point counting. You can't count the function points in each story and add them up, since the scope elements to which function points are assigned overlap multiple stories. This can be mitigated by doing the function point count in parallel to story points or other techniques for iteration planning, limiting the use of function points to release estimation and tracking overall progress. Although several of the members of the Agile Round Table did use function point counting extensively, there was less prevalence of recounting actual function points at the end of a project.

The biggest challenge, though, to using function point counting on agile projects is the agile technique of emergent requirements and the lack of scope detail at the time when we need to estimate the release or project. This is the issue addressed for story points by "Big Rock Sizing," and it creates some specific issues that must be overcome when using function points.

The standards for function point counting require a level of detail about the requirements not usually available early on in an agile project. For example, when you count function points, you count the data stores you expect to use, and assign function points based on the approximate number of data elements. In an agile project, you will not have this detail early. In some cases, you can take a smart guess. If you're building an airline reservation site, you know you'll have a data store for airports, for flights, for available seats, for reservations, and several others. But the complete list of data stores only emerges as the project progresses and the big rocks are refined iteration by iteration. And even for the ones you can guess at, you won't know the number of elements, even approximately, to use when assigning function points. Forcing the team to think about these details early goes against the main agile technique of emergent requirements. Even "taking a good guess" biases the team for later, when detailed decisions are needed.

The members of the Agile Round Table suggested two mitigations for this problem. The first was the use of "function points lite," as one member of the Round Table called it. Instead of formal function point counting, including assigning function points based on the detailed complexity of each of the scope elements, you assume

each element has average complexity. You can also count the obvious elements without biasing future emergent requirements and buffer the count to deal with the less obvious elements.

The second was "counting on the side." This mitigates the team's resistance to using function points for their day to day activities, and is particularly useful for tracking progress in a project and forecasting completion as the project progresses. An initial function point count is done based on the high level scope (perhaps using "Function Point Lite"), and as the scope is refined by breaking epics down into stories, a separate tally of function points is maintained, modified as the backlog is groomed.

Source Lines of Code for Agile Estimation

Agile projects, like all other software development, produces code as an end product, and counting that code gives a measure of size. Counting code is perhaps the most concrete measure of size of a project. Unlike function points and story points, there is a standard definition of a single source line of code, even though it's surprisingly complex.

Counting source lines of code fell out of favor with many development teams long before agile methods became popular. There are many reasons for this, some good, some more emotional than logical. A fuller discussion of this is outside the scope of this paper. But counting code still remains a viable way to compare the size of projects, and perhaps the best way to get standardized data from many different sources, so it's particularly valuable to compare your organization's projects to industry data. And since it's independent of the way the scope of the project is expressed (e.g. as epics and stories, vs. the breakout used by function point counting), counting finished code allows you to compare projects developed with agile methods to projects developed in other ways.

The main disadvantage for using code counts for early estimation is not unique to agile methods. Since it's not widely used for any purposes anymore except estimation, the project team members are particularly inexperienced at counting total code. Ask most developers, "how many lines of code do you think that will take?" and you'll be lucky if you just get a blank stare.

For counting code of finished projects, you can use automated code counters that work with the code files themselves, or sometimes with the version control systems that your team may use. Thus you can get code counts "after the fact" to use to calibrate your estimates using other size measures, as we have described earlier in this paper.

Summary

As with projects built by other methods, function points can be applied to agile projects consistently because of international standards. Agile teams may be resistant, however, because of the perception that function points are old fashioned. More importantly, the lack of detailed requirements early in a project presents a challenge to rigorous function point counting.

Source lines of code is concrete measure that can be used consistently, and can help compare your projects to industry wide data. You can automate counting the size of completed projects, thus getting a good historical base of data for future estimates. However, the lack of experience in this measure makes it difficult to use in advance for release estimation when estimating a new project.

In the next article, we will look at the issue of assigning a gearing factor for story points to use with the SLIM Suite of Tools.