Agile Estimation – What's the Point?

Agile is at the forefront of many software development discussions these days, so I thought this would be a good topic to dig into a little bit. There is a hotbed of activity in systems delivery around "Agile" – everyone is asking for it, everyone wants to be Agile, especially in the realm of federal contracting. But at the same time there seems to be quite a bit of misunderstanding and misinformation about how Agile works in practice, especially when it comes to estimation. I've seen situations where customers ask for organizational velocity, average velocity, or similar metrics for contractual reasons. This is nothing more or less than problematic.

I've experienced other situations where we have to estimate software development effort and cost at the macro level to create a project budget or a proposal bid, which is very difficult to do with most Agile estimation techniques. So the purpose of this discussion is to dig into some Agile estimation concepts, talk about where it works, and note where some challenges arise. Then we can discuss about some other estimation approaches that may nicely complement or supplement these and can help overcome some of these challenges.

Two Maxims

In my 20+ years of experience in estimating software development projects, I have come to understand that two maxims are critical. Now when I use the term "maxim" I mean an expression of general principal or truth.

1. Any approach to estimation MUST enable communication when changes to a project can possibly impact cost and schedule of delivery.

You need to be able to explain the consequences to cost and schedule when requirements and/or development conditions change. Communication, expectation management, and customer buy-in are critical to project success – and these things are critical. Unless you've got unlimited budget and no time constraints, you need to be able to explain the consequences to cost and schedule when (not if) requirements and/or development conditions change.

2. Size matters.

Software should be estimated based on some sort of sizing unit. It's simply an industry best practice that too often gets ignored or overlooked. Estimating in level of effort (hours), although widely practiced, does not enable effective communication. Hours is NOT a unit of size. Let me give you an example to illustrate this. Let's say we need to estimate how far Baltimore, Maryland is from Washington, DC. I come up with an estimate of an hour and a half, but you generate an estimate that says it should only take an hour. How do we reconcile that? Honestly we can't. Depending on the route we take, how fast we drive, the amount of traffic, weather conditions, etc., we really cannot have an informed discussion. But if I say it's 45 miles, then "miles" is a common unit of measure that we can have an informed discussion about. Want to take the Baltimore Washington Parkway instead of I-95? Well that may give us a different number of miles. But that "size" unit enables that informed discussion.

Another example: let's say we're building a house. Would you ever ask a builder to give you a 10,000 hour house? No! Of course you wouldn't. You negotiate in terms of square feet, which is a common size unit that enables effective communication and common understanding.

“Typical” Agile Estimation

First let’s agree that “typical” Agile estimation is probably a unicorn – it simply doesn’t exist. This makes sense given what Agile development tries to accomplish and how Agile estimation techniques are implemented. The Agile manifesto values individuals and interactions over processes and tools. Agile teams are encouraged through the twelve principals to reflect on how to become more effective, then to tune and adjust its behavior accordingly – which applies directly to estimation practices. So it’s no surprise that Agile estimation cannot be defined by a specific approach, technique, or tool.

In practice, methods used for estimation on Agile software development projects vary widely. Agile teams adopt their own approach and adapt it based on what works for them. Here is a list (certainly not all-inclusive) of some methods that are widely known:

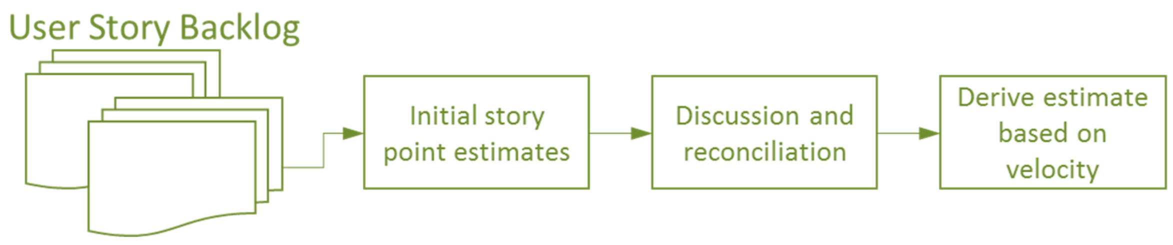
- Story points
- Bucket system
- Affinity mapping
- T-shirt sizing
- Ideal days
- Dot voting
- Ordering protocol
- Big/small/uncertain
- Planning poker

In spite of this variability, two key concepts are consistent across these methods. First, estimates are generated in a collaborative manner. Teams participate in estimation and planning activities together so many viewpoints, perspectives, and experience can be considered and included. Second, estimates are typically reviewed and revised in an iterative manner. Each additional iteration is informed by experience, lessons learned, and information gleaned from the previous sprint. So *collaboration* and *iteration* are cornerstones of effective Agile estimation.

We can’t cover all of these Agile estimation techniques in a single article, so let’s focus on one approach that is fairly widely used: ***story points.***

Agile Estimation with Story Points

Story point estimation starts with the generation of *user stories*. User stories are short descriptions of a desired function or feature written from an end-user perspective. They are often expressed in a way that can be easily understood, such as, “As a user of this system, I want X feature so that I can accomplish Y.” AS user stories are developed, they serve as the basis (requirements) for what software functionality will be built. Product owners capture requirements from the business/customer. The Agile team develops user stories from the users perspective and work with product owner to prioritize them (user-centric collaboration in action). Agile teams assign story points to user stories as a size measure that helps to plan out and deliver these priorities.


User Story Backlog
Initial story point estimates
Discussion and reconciliation
Derive estimate based on velocity

The Agile team selects what is called a “reference story” and determine the point value of that story. Then all other stories are evaluated relatively against that reference. No “standards” exist for story points. They are determined on a team-by-team basis. As an example, let's say we want to estimate the size of a set of vehicles. The table below demonstrates the relative sizing that two different teams might come up with.

Vehicle	Point Size
Honda (Reference)	5
Cadillac	8
Smart Car	2
Ford Pickup	13
Motorcycle	1
Semi Truck	21

Vehicle	Point Size
Toyota (Reference)	7
School Bus	30
Minivan	10
Chevy Pickup	14
Fiat	3
Hummer Stretch Limo	20

Which one of these is correct? Both of them, assuming that the team collaboratively developed the weighting. The key point of this illustration is that story point values are team-specific and will likely evolve as the team works together over time. No two teams are going to define story points in the same way, so tying to compare size estimates between two teams is an egregious error.

Velocity

In order to translate story points into effort, Agile teams need to understand their typical *velocity*. Velocity basically measures the team output in story points per sprint. So once you have story point estimates for a user story backlog, you can estimate how many sprints the Agile team will likely need to complete the development. Likewise, velocity metrics help determine the scope of stories included in each sprint. Velocity metrics are most reliable when certain conditions are met.

- Team members remain consistent across sprints
- Each sprint encompasses about the same number of work days
- Technology/development tools/programming languages remain consistent over the course of the development life cycle
- Team members are fully dedicated to the project

Planning Poker

Planning poker provides opportunity for everyone on the product team to have input to size estimates based on their roles, perspectives, and experiences. Teams often apply Fibonacci or other numerical sequences. Discussion and iteration helps the team consider potential risk areas and develop a collective agreement about what stories to include in a sprint. Estimating story points and applying velocity metrics can reduce biases and natural tendencies that typically occur when estimating level of effort in hours

Collaboration and iteration are essential to ensure all viewpoints and experiences are taken into account in estimation and planning. Someone may have additional insight into a particular story that the rest of the team may not, so without group discussion this insight may be completely overlooked.

Hypothesis: In General, Story Point Estimation Works Well for Agile Teams for Planning Sprints

I wanted to start from the perspective that this type of estimation approach works well for a lot of Agile teams. When a team has had some run time they can become very good at predicting how much they can accomplish in a sprint. The immediacy of feedback of data from the previous sprint during retrospectives provides the opportunity for lessons learned to be applied right away. The relative nature of story points allows teams to tailor and calibrate the size unit to their own situation. Collaboration and iteration with all stakeholders, including the customer/product owner encourages communication and expectation management. Plenty of Agile teams have had success with this approach.

In some situations, however, story points and velocity do not work well and are insufficient to meet stakeholder needs.

- ***Generation of estimates to establish initial project budgets***. Limited IT budgets mean that organizations must make choices when deciding what development projects to undertake in a given timeframe. Projects compete for scarce funding and staffing resources. Other environments (such as government contracting) require competitive bids to determine what organization will deliver the work. Bidding organizations need to develop estimates that are well-documented and defendable. It is virtually impossible to effectively apply story point sizing to effective estimate a budget that early in the project life cycle. So for overall project estimating for big picture budgeting and portfolio planning, Agile techniques may not be the best approach.

- ***Formation of a new development team with no history together***. With a new team, actual velocity is unknown. Not only that, but an Agile team usually takes a few cycles to normalize their estimation techniques. In fact, Agile estimation has to reset any time something changes on a project, even if it is just the loss of a single team member.

- ***Establishment of organizational portfolio management with consistent metrics across projects.*** Consistent IT productivity metrics require a consistent, standardized measure of software size. As we've discussed, comparing story point sizing across different teams or organizations basically amounts to estimation malpractice. This applied to competitive bidding for development projects as well. Trying to effectively compare apples-to-apples across bids that have completely different sizing metrics is simply not possible.

Other situations and circumstances can lead to less-than-optimal application of story point estimation methods.

- **Using points as a proxy for hours.** It becomes very easy to slip into the habit of estimating how many hours something will take, then calculating the number of story points that equals. For example, 1 story point equates to 4 hours of work, and I estimate a piece of work will take 12 hours, which translates into 3 story points. This approach is no different than estimating in hours and eliminates the benefits of using relative sizing.

- **Treating velocity as productivity.** Organizations often set improvement goals based on productivity and quality metrics. For example, if a team produces a velocity of 20 story points per sprint, management may challenge the team to improve performance by 25% and deliver 25 story points per sprint. Or if one team is delivering an average of 30 story points per sprint and another is only completing 20, management may ask the second team why they are so much less productive. These examples are inappropriate applications of metrics, and doing so will result in unintended consequences and "bad behavior." Don't you think if you were on a team delivering 20 story points per sprint, and some Pointy-Haired Boss said you needed to be yielding 30, that you would find a way to update your story point weighting pretty quickly? Story point inflation can be pretty easy with the right motivation.

- **Customers can have an alternate idea of what a story point is or should be.** Without an established "standard" for story point, this disconnect can be a real possibility and can be a source of serious miscommunication and misunderstanding. If your business customer and/or product owner defines story points or their weighting differently from the development team, collaboration can turn into combat. Let's go back to our house analogy. What if house builders each had their own sizing metric – instead of using a standard foot measure (12 inches), they went back to the old days of a foot being equal to the size of their shoe. As the future home owner with a size 12 shoe, you ask your builder for a 2,000-square foot home. Unfortunately, your builder is closer in size to Herve Villachaize (who played Tattoo on Fantasy Island) with a size 3 shoe. His measurements on that same sized house would be a serious source of contention between the two of you.

A Supplemental (Alternative?) Approach

Given this range of challenges, I'd like to explore some ideas for estimating software size that can either supplement or replace the use of story points on Agile development projects. If you've got Agile estimation techniques in place that work for your team, great! I would never recommend disrupting that. The idea of sizing user stories keeps the focus customer-centric, so keep doing what works. The supplemental aspect of my proposal would help fill in some of the shortcomings of story points if your team needs a more effective way to develop defendable initial project estimates or a set of consistent metrics. However, if you find that your team has difficulty applying story point estimation consistently or is frequently subjected to some of the other pitfalls, I would recommend an alternative approach to sizing altogether.

If only there was an industry standard software sizing measure that is based on functionality described from the users' perspective…well, what do you know – there actually is! The function point sizing standard established and maintained by the International Function Point Users Group (IFPUG) is exactly that. Function points measure software size based on the functional requirements requested by and provided to the user. Counting rules are documented and maintained in the Counting Practices Manual. Function points are accepted as a standard size measure by ISO(20926:2009). IFPUG also offer a certification program to recognize experts in the field – certified function point practitioner (CFPP) and specialist (CFPS).

A Quick Function Point Overview

I don't want to get too far into details, but it's important to understand the basic function point counting methodology. There are two types of functions: data functions and transaction functions. Data functions represent the "things," or pieces of information, the system needs to meet the users' needs. Transaction functions are the business processes, reports, etc. – that data in motion – that users need to manage and manipulate the data to do their jobs.

The methodology essentially involves identifying data and transaction functions, assessing the complexity, and applying the standard function point weighting matrix. Data function complexity is determined by the number of data elements and logical data subgroupings. Transaction function complexity determined by the number of data elements and files referenced during the transaction. The IFPUG weighting matrix identifies how many function points each identified function receives.

	Low	Average	High
Internal Logical File	7	10	15
External Interface File	5	7	10
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6

This matrix is the foundation of the standard, and ensures that counts are consistent, repeatable, fully documentable, and auditable. The IFPUG FP matrix identifies how many function points each type of function receives based on its complexity. By nature, it also has an aspect of relativity to it, similar to that of story point sizing, but much less subjective. A function point is by definition a standardized metric that describes a unit of work product suitable for quantifying software that is based on what the end user requests and receives. Translating that into more common terms: all things being equal, an average internal logical file at 10 FP should take about twice the effort as an average external output. One of the key differences from story points here is that with the rules and standards, sizing becomes more objective.

The Problem with Function Points

This all sounds great, right? Customer-centric sizing, standardized and ready to go! So why aren't function points more widely applied to Agile development projects? Well, it turns out that function points have a real problem that is prevalent: ***perception***. The term "function points" can evoke a visceral negative reaction from many people, especially Agile enthusiasts. Try it sometime. Just mention the term "function points" to an Agile developer, and you will probably get one of two reactions: either, "What's that?" or, "No way, function points don't work in Agile." Function points take too much time and effort. Requirements must be fully defined to effectively apply function points. Function points don't offer the flexibility we need to estimate in an Agile environment. Function points are not granular enough to apply in an Agile environment. Usually these arguments are made without actual experience with function points, which is why I claim that this is really a problem of perception. But perception can become reality, so the end result is the same.

Revisiting the Agile manifesto and principles, you can understand why the initial reaction to something as structured and "archaic" as function points would be resistance and rejection.

- Values individuals and interactions over processes and tools.
- At regular intervals, team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Point taken! Maybe there are just too many obstacles to overcome to ever effectively and consistently apply function points in an Agile development environment. So how do we deal with this bias against function points and still address the challenges of Agile estimation? This is something a group of estimation experts have thought long and hard about, with many late night meetings and contentious discussions, maybe even a few broken chairs.

And we came up with a concept called Agilons

Agilons

The Agilon sizing method is similar to function points, only they are applied specifically to Agile software development projects. There are five types of Agilons.

1. Internal data – managed by the application
2. External data – referenced by the application but managed by some other application
3. Inputs – add, change, delete internal data
4. Outputs – reports, calculations based on internal or external data
5. Inquiries – search and retrieval of internal or external data

Agilon complexity generally can be determined by the number of data elements involved. However, this detailed information is not always available when an estimate needs to be completed. If this is the case, analysts should simply make an assumption about Agilon complexity and document it for future review and discussion. One common technique applied is to assume that all functions are average complexity. Here is the standard Agilon weighting matrix:

	Low	Average	High
Internal Data	7	10	15
External Data	5	7	10
Inputs	3	4	6
Outputs	4	5	7
Inquiries	3	4	6

This might look familiar to you. It's not quite a Fibonacci sequence, but yeah, it's pretty close.

Analyzing user stories in the Agilon framework can provide a good litmus test for those user stories. If a user story can be applied to more than one type of Agilon, it probably should be broken down into simpler stories (concept of elementary process or minimum viable product). Velocity can be measured in Agilons per sprint

Let's take a look at a real user story and apply the Agilons framework.

As a customer I would like to have the ability to search for and reserve a hotel room in order to spend the night in another city.

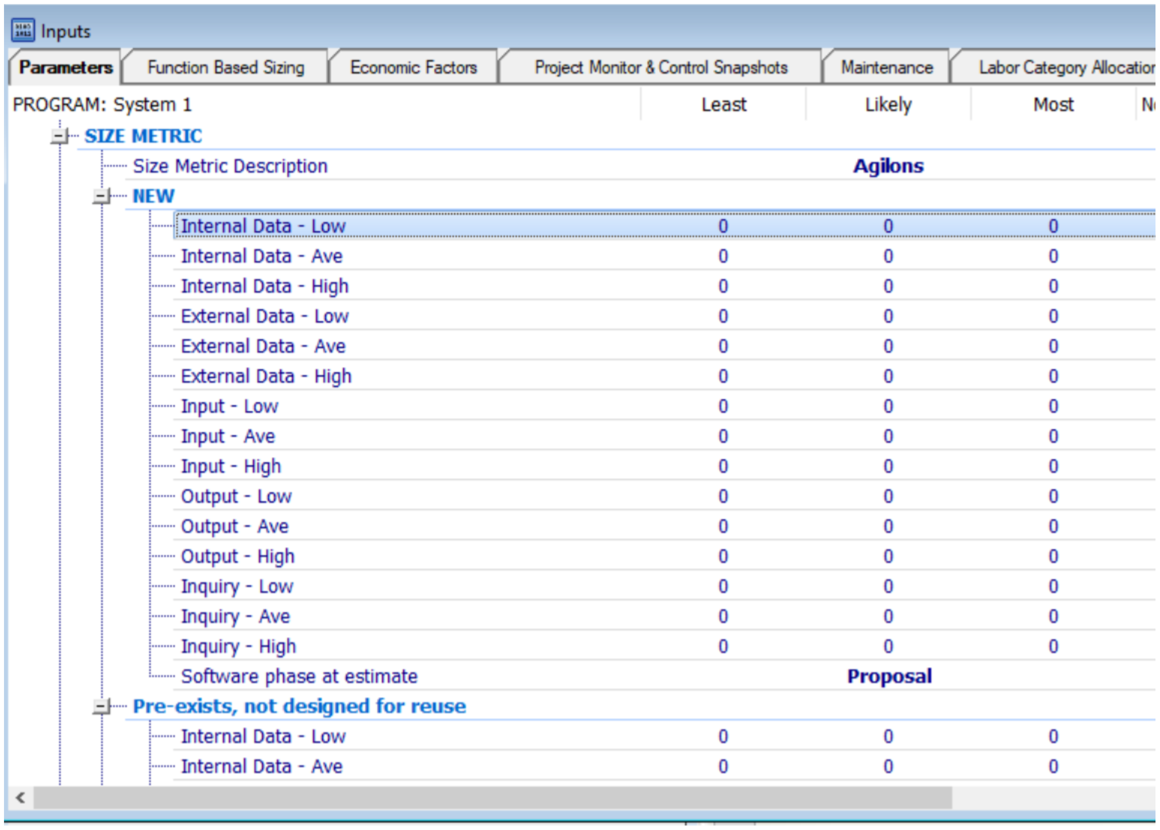
First off we should recognizing that details on data elements are missing, so we can assume "average" complexity for any identified Agilons. This is just an assumption, but it's one I can document and revisit later. If I had a different understanding about the simplicity or complexity, then perhaps I would make a difference assumption. For example, perhaps I have previously seen similar functionality in a different

application, and know that there are usually a low number of data elements involved. Then I could in good conscience assume low complexity and move on from there.
Analyzing this user story reveals multiple Agilon types that need to be decomposed.

Description	**Agilon Type**	**Agilon Size**
Hotel data	Internal data	10
Search for hotel room	Inquiry	4
Reserve hotel room	Input	4
	Total	18

Size is really only part of the estimation equation….so how do we translate that into effort, cost, and schedule? When converting size to cost, historical data is ideal. With a consistent sizing metric like Agilons, data can be collected from across an organization and used as the basis of estimate. Data can be stratified and applied appropriately to new teams, mixed teams, experience teams, etc. In this situation for sprint planning purposes I would simply review my team’s velocity. If it is around 18 to 20 Agilons per sprint, we’re good to go. But what happens if we don’t have that data, or if we’re tackling a new type of application with a newly formed team?

In this scenario , an estimation tool like SEER for Software can be an effective way to generate and document estimates. SEER for Software leverages large historical data sets and flexible input parameters. Outputs from tools are based on assumptions (size, personnel skills and experience, development environment, etc. which can be fully documented and discussed. SEER for Software is parametric in nature, meaning calculations are based on complex statistical algorithms. The customizable Sizing Metric function in SEER for Software enables users to apply the Agilon framework to develop an estimate.

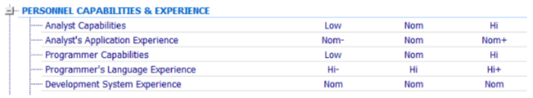


PROGRAM: System 1	Least	Likely	Most	N
SIZE METRIC				
Size Metric Description		Agilons		
NEW				
Internal Data - Low	0	0	0	
Internal Data - Ave	0	0	0	
Internal Data - High	0	0	0	
External Data - Low	0	0	0	
External Data - Ave	0	0	0	
External Data - High	0	0	0	
Input - Low	0	0	0	
Input - Ave	0	0	0	
Input - High	0	0	0	
Output - Low	0	0	0	
Output - Ave	0	0	0	
Output - High	0	0	0	
Inquiry - Low	0	0	0	
Inquiry - Ave	0	0	0	
Inquiry - High	0	0	0	
Software phase at estimate		Proposal		
Pre-exists, not designed for reuse				
Internal Data - Low	0	0	0	
Internal Data - Ave	0	0	0	

<u>Coming Full Circle</u>

So how does this approach really address the challenges of story point estimation? Let's revisit some of the most significant pitfalls of estimating with story points.

- ***Generation of estimates to establish initial project budgets.*** With Agilons, estimates can be fully documented and explained, even in the absence of requirements, and then can be used facilitate communication. You can establish a project baseline, but when conditions turn out differently than anticipated, the estimation methodology becomes a mechanism for communicating what has changed and why, as well as what can be accomplished.
- ***Formation of a new development team with no history together.*** Applying Agilons as a standardized sizing metric, combined with good historical data or a parametric model, you can provide estimates that stakeholders can understand. You can tailor the estimate to the combined experience of the development team.



PERSONNEL CAPABILITIES & EXPERIENCE			
Analyst Capabilities	Low	Nom	Hi
Analyst's Application Experience	Nom-	Nom	Nom+
Programmer Capabilities	Low	Nom	Hi
Programmer's Language Experience	Hi-	Hi	Hi+
Development System Experience	Nom	Nom	Nom

- ***Establishment of organizational portfolio management with consistent metrics across projects.*** Leveraging Agilons as a standardized size measure across an IT portfolio empowers consistent productivity and quality metrics across an organization, offering real possibilities for improvement.

<u>Sources</u>

Berteig, Mishkin, "9 Agile Estimation Techniques," Agile Advice Blog, October 13, 2015.

http://www.agileadvice.com/2015/10/13/agilemanagement/9-agile-estimation-techniques/

Clifford, John, “Agile Estimation: Key Principles and Practices for Successful Agile Projects,” 2012.
http://www.construx.com/uploadedFiles/Construx/Construx_Content/Resources/Presentation/AgileEstimation_KeyPrinciplesAndPracticesforSuccessfulAgileProjects.pdf

Cohn, Mike, Agile *Estimating and Planning*, Prentice Hall Professional Technical Reference, 2006.

Cohn, Mike, “The Main Benefit of Story Points,” September 9, 2014.
https://www.mountaingoatsoftware.com/blog/the-main-benefit-of-story-points

Computer Associates, “Top 10 Mistakes Made by New Agile Teams,” 2016.
https://help.rallydev.com/top-10-mistakes-teams

David Consulting Group, “Can Function Points be Counted/Estimated from User Stories?” *Trusted Advisor*, April 2016.

Donavan, John, Agile Estimation *Practices – Demystifying Story Points*, John Donovan, 2013.

Green, M. David, “Do You Make These 7 Agile Estimation Mistakes?” July 9, 2014.
https://www.sitepoint.com/make-7-mistakes-agile-estimation/

IFPUG, “Applying Function Point Analysis to Scrum Agile Software Development Projects,” V.1.0, October 25, 2013

James, Michael, “Scrum Effort Estimation and Story Points,” Scrum Methodology Blog, November 21, 2009.
http://scrummethodology.com/scrum-effort-estimation-and-story-points/

Kerievsky, Joshua “Stop Using Story Points,” October 12, 2012,
https://www.industriallogic.com/blog/stop-using-story-points/

Radigan, Dan, “Collaboration, Abstraction, and Other Secrets of Agile Estimation,” Undated.
https://www.atlassian.com/agile/estimation