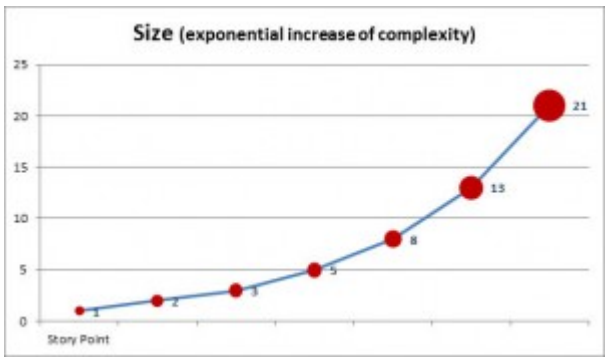
people10.com

» Story point estimation simplified: and here is how it works…

11-14 minutes



There are many reasons why task breakdown does not work for pure agile delivery. The estimation practice has to work in different ways in traditional (waterfall) and agile deliveries.

One of the main differences between traditional and Agile is that in traditional, you know the details of all your requirements up-front. Whereas in Agile, you only have the high-level requirements, and you elaborate as you move on.

Elaborate and upfront requirements definition is not possible in agile projects

Traditional method for estimation (WBS)

Requirements → Modules → Task breakdown → Effort estimation → Project cost

When you don't know your requirement details up-front, how would you do a task-level breakdown? Apart from that, when your requirements can change and evolve (the agile way), how would you manage your WBS?

There are many **disadvantages to using a traditional estimation method** in an agile context. Some of them being:

– Task breakdown is a technical view of the work in hand

– Task breakdown is difficult to understand for the product owner

– They don't know what they are paying for and how much value is realized for finished tasks. i.e the product owners do not realize the cost of each feature

– Tasks breakdown gets into thousands of line items and difficult to maintain

– Change in one task affects related tasks; change in requirement affects too many tasks

– Difficult to allocate, manage task interdependencies, and assign ownership

– Cannot measure productivity and improvements as you move along in a project

– Cannot measure scope change ; only able to tract effort change

– Cannot link effort change to scope change where increase in effort is caused due to increase or change in scope

In order to overcome these challenges, we use the ‘story point sizing’ technique.

Agile method for estimation (Story point sizing)

Requirements → Stories or Features → Story Point Sizing → Effort estimation → Project cost

In agile, we avoid doing a task-breakdown to derive at the the project cost. This method eliminates all the disadvantages stated above. We will discover how.

What is size?

While estimation is measuring the cost of the work, Sizing is nothing but ‘measuring the amount of work’ in producing the feature. The trick is to derive the size first, and then the effort. The effort needed by two different programmers to produce a feature may be different. (Due to factors like skill, experience, domain familiarity, etc..). Nevertheless, the size of the feature remains the same for both.

There are many methods to sizing like function point, story point etc. For agile projects, the features are in the form of stories, and story point is the best method for sizing.

What is Story Point?

Story Point is a unit of measuring ‘size’, same as ‘kilo’ or ‘pound’ is a measure of weight. Story point is nothing but the measure of complexity of a story or a feature. Take an example of a typical story: “The user wants to enter his user name and password, click the login button and login to the application”

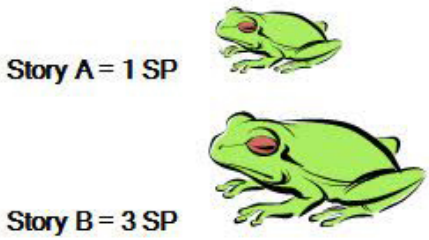
In a traditional approach, If we were to break this down into tasks, we would end up with tasks like database creation, table creation, creation of the UI layers, writing test cases, testing, making fixes etc.

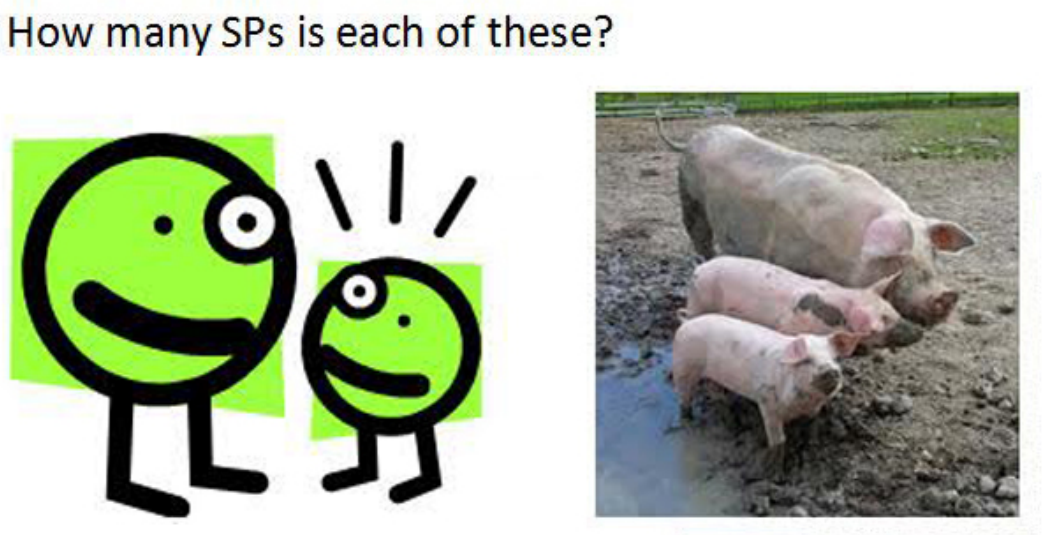
Developer A will say: I need 2 days to do this” while Developer B says “I need 5 days to do this”

These numbers are not a true measure of the complexity of the job. They are debatable and they indicate the ‘effort’ as perceived by developers of varied skill and experience levels. How would you really measure how complex or how big the story is? How would you compare it with the ‘size’ of another story?

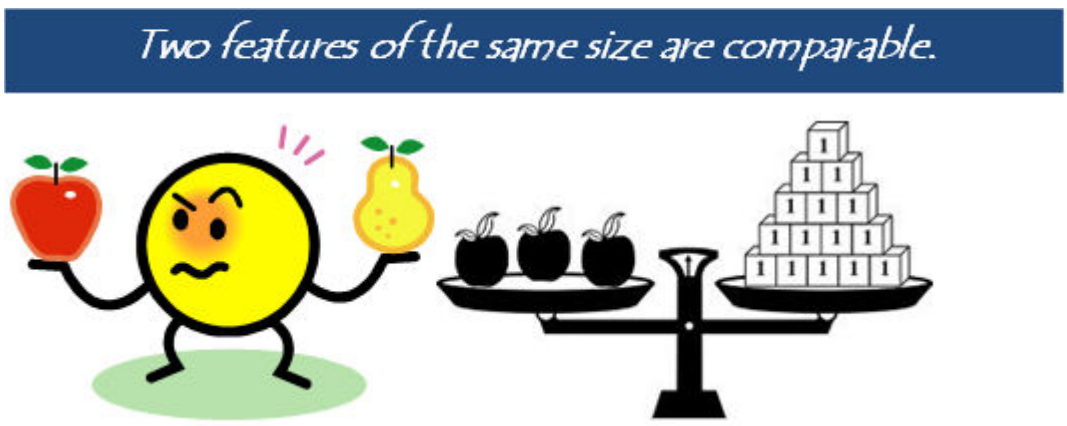
Typically in a product backlog, we have many stories. What story point sizing is intended to do is to ‘weight’ each story

and decide how big (complex) each one of them is. (Mind you, we are not talking about the effort involved in making the story). If we say a story A is 1 story point (1 SP), and story B is 3 SP, it just means that Story B is 3 times as complex as compared to story A





In short, the size indicates how complex a feature(story) is, how much is the amount of work (scope) required to make the feature, and how big the feature is.



An apple = A pear = 3SP means, you can replace feature apple with the feature pear and it should have no impact to the overall scope of the project and the overall cost of the project (provided there are no other dependencies that will generate more work)

Size (of a feature)	= complexity of the feature
	= baseline scope of the feature
	= how big is the feature

Note that size is not effort. Example: A story/feature to build a login screen is sized as 5 SPs. Person A has estimated 2 days to do it, and Person B has estimated 5 days to do it. The size remains the same, but effort may vary.

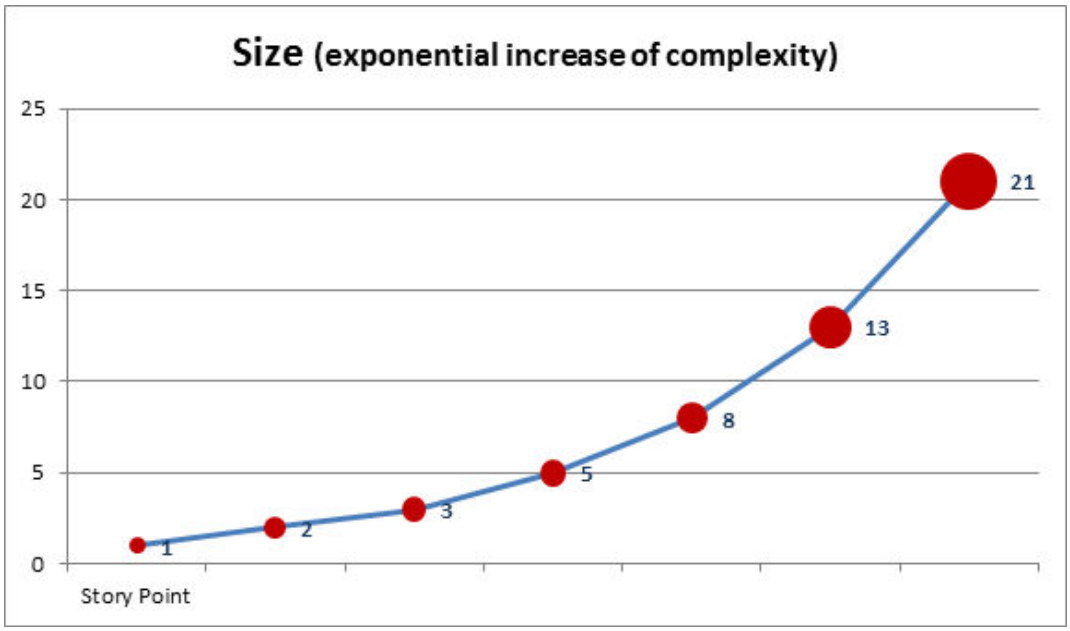
Size ≠ effort

How to measure the size?

There are many ways to measure size, one of the simplest (not necessarily the best) being T-Shirt sizing. We can go “Simple”, “medium”, “large” etc.. Most of the time, it is based on assumptions and hunch. Again, the question would be how many story points constitute ‘simple’, how many story points would mean ‘large’? I would call this the **‘lazy sizing’** method.

How big is it really???

One of the best ways to measure size is to go by the **Fibonacci series**. The idea being, the complexity of a feature hardly increases in a linear fashion. Normally, the complexity tends to increase exponentially.

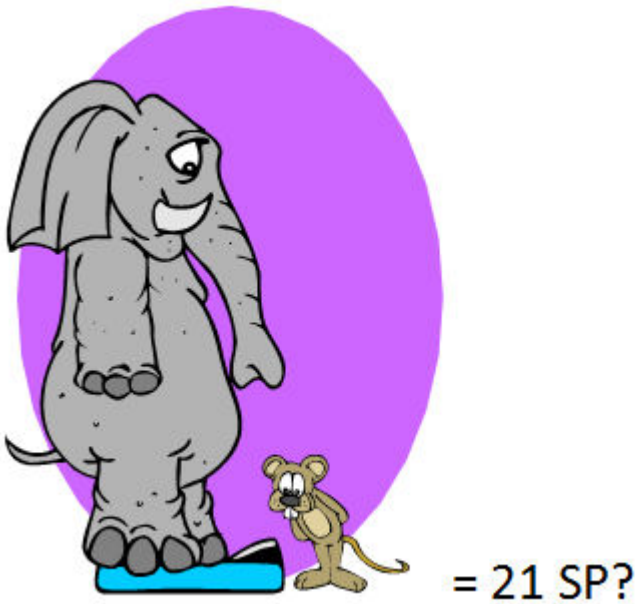


Story point sizing – Fibonacci series

If the stories are very clear, we should be able to break them into a reasonable level of granularity. A size of 21 certainly implies hidden complexity or an extremely large feature. Very large features are not desirable in an agile team. Large features are risky and there is a possibility of discovering hidden complexities during the sprint and the team may not be able to complete them in a sprint.

The team should re-look at each of these very large features and see how they can break them down into smaller features of lesser size. Note that as the story gets larger, the confidence level of the estimate gets lower.

How do you derive at the Size?



You can use many techniques for sizing. One of the most effective ways to measure the size is by using a good ‘sizing framework’. There is no ‘one-size-fits-all’ framework that is off-the-shelf. Mature agile teams can easily build their own framework using the parameters applicable to them. It is possible to derive sizing frameworks that can work across technologies. We have been successful in building frameworks that could size features across projects of open source technologies, Microsoft technologies, as well as legacy technologies like COBOL. Framework building is a

big topic in itself (that is for later J) . But generally, the idea is, against each feature, you input the relevant parameters and the value against each, you can easily derive how many SPs are each feature.

	Parameter 1	Parameter 2	Parameter 3	...	...	Parameter N	SIZE (Story points)
Story 1	value	value	value	value	value	value	3
Story 2	value	value	value	value	value	value	1
Story 3	value	value	value	value	value	value	5
Story 4	value	value	value	value	value	value	2
Story 5	value	value	value	value	value	value	8
Story 6	value	value	value	value	value	value	3
Story 7	value	value	value	value	value	value	1
Story 8	value	value	value	value	value	value	5
TOTAL (SP)							**28**

Story point sizing framework – simple example

Here you can see that eight stories are sized to be a total of **28 SPs**.

Baselining the scope through sizing

The amazing thing here is that, by using sizing, you are able to measure and baseline your initial scope. This is very important for an agile team to baseline the scope as the requirement may change and there are not tight change management processes.

In traditional method, scope baseline is the WBS. This is our ‘agile’ version of the scope baseline. Further down, I will explain how to re-scope and re-baseline.

Deriving effort from the size

Now the question is: How do you convey to your business

stakeholder about the ‘cost’ of the project? Its pretty obvious they would not want to understand or hear about story points. I have experiences where some of my very junior project managers got trashed by the clients for showing them the story point estimation.

Once you are able to measure the size, you can derive at the effort easily. There are three main methods to do this:

Situation 1: Historical information exists and team is stable

The team is not a new team and they have been doing sizing and project execution for a reasonable amount of time. In this case, the team has to look back at their last 3 sprints sprints (the most recent ones) and calculate how many SPs are delivered and how much effort is spent to deliver them.

Eg: If the team has delivered 100 SPS with an effort of 500 combined hours, 500/100 = 5 hours / SP.

In this case, they would take 28*5 = 140 hours to complete the 8 features.

If cost of an hour = $100, the 8 features would cost $14,000

Situation 2: No historical information, new team

In this case, the team can pick few features that are indicative of all the features. They can do a task-breakdown estimation of these features and extrapolate it to the whole lot.

A word of caution here that as similar to any sampling

exercise, the features you pick need to be good enough that they are exhaustive of the scope of the project.

In this case, if stories 1,3,6,and 8 are similar, you can choose any one. For instance, story 1

Stories 2,4,7 are similar, so you pick story 2

Story 5 is unique, so you also pick story 5.

Do a task breakdown of stories 1,2 and 5 (combined size of 12 SP) and you find that the effort comes to 60 hours.

i.e. the effort for 1 SP = 60/12 = 5 hours.

There you go ! As in the previous case, you can compute the effort of the project as 140 hours and the cost as $14,000.

Situation 3: No historical information, new team, research type project

Here is the most difficult situation. In this case, what I suggest is to go ahead and do a task-breakdown of all features at least for the first few sprints. When you do it for few sprints, you will see that you are able to measure your velocity. It may go up and down first few times, but will definitely stabilize few sprints into the project. Once you build up historical information, you can switch your sizing technique to method (1) mentioned above.

Changes in the scope and re-sizing

During the course of the project, changes are normal (hey, we are Agile!). But how do we keep track of these changes

and if we produce more than we originally planned for, how do we justify that?

Another main dilemma of a project team is how they justify the effort over-run caused due to changing requirements?

The fact is, a bug produced by the team will not change the size of the story. But, a change requested by the client may result in change in size. The idea is to keep track of the change in scope by re-sizing the story every time there is a change. This re-sizing is also exposed to the product owner to avoid disputes later.

	Parameter 1	Parameter 2	Parameter 3	...	...	Parameter N	SIZE (Story points)	REVISED SIZE (Story points)
Story 1	value	value	value	value	value	value	3	3
Story 2	value	value	value	value	value	value	1	1
Story 3	value	value	value	value	value	value	5	8
Story 4	value	value	value	value	value	value	2	2
Story 5	value	value	value	value	value	value	8	8
Story 6	value	value	value	value	value	value	3	1
Story 7	value	value	value	value	value	value	1	1
Story 8	value	value	value	value	value	value	5	8
Story 9	value	value	value	value	value	value	0	2
TOTAL (SP)							28	34

Note that in this example, two stories have changed (in yellow) the scope and a new story has been added. The original scope baseline was 28SP and the revised scope is 34SP.

It does not require a genius to figure out that 6 additional SPS would take 6*5= 30 more hours to complete with an additional cost of 30*$100= $3000

The team has to have good discipline to be able to re-size every change. This is how change management is done in Agile.

Measuring the team's productivity through sizing

Few sprints into the project, you will see that the velocity (number of hours / SP) of the team stabilizes and reaches a plateau. This would be your optimum productivity (or perhaps not!)

There is always room for improvement, and by looking at the hours spent/ SP, you can clearly see how your team's productivity is improving. If there is any productivity dip in any of the sprints, you can retrospect and find out the reason as well. It could be due to a new team member, attrition, unexpected change, etc. Nevertheless, without implementing sizing, it is impossible to measure real productivity.

Just by observing the effort variation is not an indicator of productivity. If scope is changed and effort is more, productivity could still be good. If scope remains the same and effort is increasing, the productivity is declining.

Too long for a blog already. Maybe one day soon we will get into the intricacies of the sizing framework  **Adios!**