Summary (from Proposal)

We know that StoryPoints are "a relative measure of size that can be applied to Stories and Epics." Beyond this simple statement there is not much about StoryPoints that we can all agree on – teams and organizations are free to estimate and use StoryPoints as they see fit.

I want to use them to aid in Project (not Sprint) Management, and in this talk I present a way to define StoryPoints for this purpose.
Agenda

▲ Purpose of Talk
▲ StoryPoint Definition
▲ Release Planning
  ▲ Release Plan
  ▲ Monitoring
▲ Problem Statement
  ▲ Size Epics for making Release Plan
  ▲ Size Stories to consume SPs
▲ Size versus Effort
  ▲ Mathy
  ▲ Effort = Size x Productivity x Variability
▲ Sizing Stories
▲ Sizing Epics
▲ Summary

Setting the Stage

Release Planning
Effort versus Size
What we Need to Do
Problem for Release Manager

How does the Release Manager (RM) develop and manage/update the Release Plan, including the Release Backlog?

Release Plan Summary: “the CatAir Project is building an airline website. The first Release will enable the User to ‘Buy an e-Ticket’ and ‘Check Status of Flights’. We will also continue to maintain the SouvSite website. This will take three months (7 Sprints), and cost approximately 2800 hours.”

This summarizes Cost, Scope (Release Backlog), and Schedule

Release Management is “owned” by the Release Manager, and is done by the Team the Release Manager belongs to.

Initial Planning Assumptions
(Initial Baselines for Release)

Size in SPs: 340 SPs
- Buy an E-ticket -125 SPs, Check Status of Flights - 75 SPs, Maintain SouvSite - 30 SPs, Chores - 70 SPs, Release Activities - 40SPs

Prod_Rate in Hours/SP: 8.25 Hrs/SP

Burn_Rate in Hours/Sprint

<table>
<thead>
<tr>
<th>Sprint</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>240</td>
<td>240</td>
<td>480</td>
<td>480</td>
<td>468</td>
<td>432</td>
<td>480</td>
<td>2820</td>
</tr>
</tbody>
</table>

These baselines justify the Release Plan, and are ‘owned’ by the Release Manager
Sprint to Sprint...

Planning Assumptions

Re-Baseline (Variance Analysis)

Release Plan - Cost - Scope - Schedule

Actuals (hours, SPs, proposed scope changes)

Definition of Done

Product Ownership

Scrum Mastering

Chores

The RM must be able to justify the current Release Plan with data – and the Team saying it can do it is not data

Leadership

Development

good decisions

good info

Variance Analysis: Do the actuals support my baseline? If not, I must re-Baseline.

Variance Analysis using TCPI

Baselines versus Actuals (BuildUp)

Baseline = 50 SPs/Sprint

Required future Velocity to catch up

Here is a simple way to “see” Variance Analysis
So...

- In order for this to work – in order for the data to be useful – the StoryPoints we use must have some properties...
  - We must be able to estimate SP(Epic/Feature), which will be the number of SPs needed to release the Feature,
  - We must be able to estimate SP(Story), which will be number of SPs the Story consumes when the Story is done
  - We must be able to combine SPs across Teams
  - We want SPs to be stable through time

I will show you how to do this in this talk...

Size versus Effort

- Basic formula the Release Manager uses to approximate Effort:
  \[ \text{Effort (in hours)} = \text{Size (StoryPoints)} \times \text{Prod\_Rate (Hours/StoryPoint)} \]
- Effort(Item) is actually Effort(Item, Team, Imp, Const)
  - Impediments (Imp) are things that can change from Item to Item – like Technical Debt, SME Availability and Organizational Noise…
  - Constraints (Const) are those things that will be consistent from Item to Item – like the Coding Language, Architectural Concept, the type of Product, Development Style, Definition of Done, and the required Regulatory Regimen
- Want a Size (SP) measure that makes the following equation true:
  \[ \text{Effort(Item,Team,Imp,Const)} = \frac{\text{Size(Item,Const)}}{\text{StoryPoints}} \times \frac{\text{Prod\_Rate(Team, Imp)}}{\text{Hours/StoryPoint}} \times \text{Variability} \]

This is what the Release Manager needs…
Example, for installing FencePosts

▲ Imps
▲ Rain, rocks, mud, vegetation, slope, etc

▲ Consts
▲ Type of FencePost

▲ Tools we use

Bottom Line

▲ We need to be able to estimate StoryPoints:
   ▲ For Epics/Features, so that we can make the plan
   ▲ For Stories, so we will know how to update the plan

▲ With the following Properties:
   ▲ Size(Item,Consts) is independent of Team and EnvVars
   ▲ Size(Item,Consts) ‘works’ in the Effort equation we just saw
   ▲ Size(Item,Consts) is stable through time
   ▲ Size(Item,Consts) is normalized across Teams

▲ So, let’s get on with it…
Estimating StoryPoints for Stories

Acceptance-Based Stories and Ideal Effort
Functional Stories and COSMIC FPs
Time-Boxed Stories and THs
Estimation Game Script

My Basic Assumptions

Constraint: I assume that the Development is being done in a ‘proper’ agile manner
- A Story is not ‘done’ until it’s verified as being done – verification testing is part of the Story
- Code Design emerges (through refactoring) as a result of doing Functional Stories
- Functional Stories are one Acceptance Test ‘thick’ (sashimi development)
- Architectural Decisions are usually made as the result of doing Architecturally-Significant Functional Stories – there could also be Spikes…
Acceptance-Based Stories

*StoryPoint*(Story) is a relative measure of Ideal Effort, which is the effort it should take to develop the Story (meet the Doneness Criteria) if everything were as it should be (given your Constraints); there are no Impediments, and you don’t require any magic or miracles.

- This is a logical extension of XP’s Ideal Time
- All SPs should take about the same amount of Ideal Effort
- The size of a Story is separated from the variations in effort it takes
Ideal Effort vs Actual Effort

Actual Effort = Ideal Effort + Drag caused by Impediments

It’s the ScrumMaster’s Job to try to move the Team To this...

Velocity curves will “look like” the dark parts of these graphs...

Advantages of “Relative Measure of Ideal Effort”

Once the Impediments settle down to steady-state, this method provides productivity metrics that allow accurate long-term predictions

These StoryPoints provide a relative measure of “how much” effort should be spent on the Story; how much the Story is worth; how much the Story should cost.

If Velocity metrics don’t settle down, this indicates that the Impediments haven’t settled down, so there will be things for the ScrumMaster to retrospect with the Team about
It would be nice if we could find a natural measure to help size functional Stories. A natural measure would something we find in the problem (or logical solution) that:

- Can be easily estimated
- We believe is linearly correlated with Ideal Effort
- (optional) can be calculated when the Story is ‘done’

Examples:
- Lines of Code (LOC) – for a non-OO development
- total number of Class Members – for an OO development
- the number of widgets on a page – for web development
- the number of ‘passing’ acceptance tests we need
- Etc…
One common natural measure...

This idea says that the size of a Functional Story/Scenario is determined by the complexity of the associated sequence diagram...

The COSMIC Function Point (CFP)

There is a standard (ISO/IEC 19761), that uses this particular natural measure.

It is called the COSMIC (Common Software Measurement International Consortium) Functional Size Measurement Method.

This measure is an extension of Use Case Points from the 1990s, and is a useful (for our purposes) Function Point Method.
More Specifically

- The size of the Story is the number of COSMIC Function Points (CFPs) the Story created or changed...
- Where a CFP represents Data crossing Your System’s boundary

For a given scenario, all we do is count the number of times information moves across the boundary, and the total is our COSMIC FP count.

When you change code, you get the CFPs that were effected by the changes...

- This count measures the complexity of the ‘conversation’ these entities are having in this scenario...

Note on Complexity

- I often get the comment: “It can’t just be about data movements! Sometimes the Code is just plain complicated!”
- Not in most systems...
- Most complications are because there are many states, or conditions, to deal with. Each of them gets its own flow and its own CFPs.
- Changing a piece of code to make a different Acceptance Test pass guarantees at least 1 CFP gets created... and it’s usually at least 2 CFPs
Estimation Games

► Of course, we're not going to count CFPs, we're going to estimate them
► So, we'll need some exemplar Stories to estimate against
► I like to use Stories with T-shirt Sizes of
  ► Small = 2 SPs
  ► Medium = 4 SPs
  ► Large = 8 SPs

► Let me show you why…

Small Story: Present ‘Add Traveler’ Screen

► This is a Small Story with CFP = SP = 2
Medium Story:  
Get List of Flights from CUTLASS

Get List of Flights from CUTLASS  
Present List of Flights from CUTLASS that matches the User's Itinerary  

Preconditions:  
- Traveler has prepared an itinerary  

Postconditions:  
- CUTLASS has returned a list of flights that matches the itinerary (up to 10)  
- The list is presented to the traveler

This is a Medium-Sized Story, CFP = SP = 4

Large Story:  
Pick One and Pay with Credit Card

Pick Flight and Pay with Credit Card  
Choose a Flight from list, and Pay for it with a Credit Card Using the Credit Card Widget  

Preconditions:  
- a list of flights is on the screen  

Postconditions:  
- the chosen flight has been paid for  
- the itinerary is updated as being paid for  
- the payment is confirmed to the traveler

This is a Large Story, CFP = SP = 8
Normalizing and Stabilizing Stories

- If each team has examplars for their estimation games that have the same number of CFPs, then
- Doing an estimation game with the question:
  “which examplar is Story X most like, with respect to new or changed data movements?”
Leads to Stories that are both Normalized across Teams and Stabilized across Time…

- Because either the data moves, or it doesn’t

Sizing Time-Boxed Stories

- Exploratory testing of Buy an e-Ticket
- Do 2 days of Cleanup in Module ABC
Time-Boxed Stories

- Some Stories are not Acceptance-Based – their Done is not well-defined.
  - CleanUp Stories
  - Analysis Stories
  - Research Stories / Spikes
  - Etc…

- ‘Done’ for Stories like these is defined by a Time-Box. I like to time-box in terms of Team Hours (THs) – the total number of hours we expect Team Members to work on it.

- THs can be converted to StoryPoints (for the purpose of metrics) by using Yesterday’s Weather or the current Prod_Rate baseline.

Sizing Architecturally-Significant Functional Stories

- Backlog
- Item
- Epic
- Story
- Container
- Feature
- ‘big’ Story
- Acceptance-Based
- Time-Boxed
- Atomic Use Case
- Plus-Up
- Functional
- Spike
- Architecturally-Significant

‘Get List of Flights from CUTLASS’
Architecturally Significant Stories

- Architecturally Significant Stories are functional Stories that cause the Team to make architectural decisions.
- Basically, they are Functional Stories with imbedded Spikes – stories that figure out answers to tough problems.

Therefore, we size them by sizing them as Functional Stories first, and then add some SPs in order to account for the imbedded (time-boxed) Spike.

- Basically, for each new data movement, add a point to account for new interface you need to ‘invent’
- So, for a totally new ArchSig Story, you double the SPs… for less than that you ask the Team how many to add…

‘Complete’ Estimation Game

1. If <Story ABC> is a Functional Story, ask the question: ‘Which exemplar Story is <Story ABC> most like, in terms of moving parts that need to be added or changed?’
   - If <Story ABC> is architecturally-significant, add some StoryPoints (up to double the size) in order to account for making the architectural decisions
   - If <Story ABC> requires significant Cleanup, add an additional (time-boxed) Cleanup Story to be done concurrently

2. If <Story ABC> is a non-Functional Acceptance-Based Story (it has a well-defined Acceptance Criteria), then ask the question:
   ‘Which exemplar Story is <Story ABC> most like, in terms of Ideal Effort; that is, the effort each would take if everything were as it should be?’

3. If <Story ABC> has ill-defined Acceptance Criteria, you should Time-Box it in terms of Team Hours (THs); and then convert to SPs if necessary.
Estimating StoryPoints for Epics

Use Cases and PlusUps
Other Epics

Sizing Atomic Use Cases

'Buy an e-ticket' or 'Rent a Hotel Room'
Atomic Use Cases

An Atomic Use Case is the accomplishment of a single goal in a single session, like 'Buy an e-Ticket' or 'Rent a Hotel'.

A Use Case is a collection of scenarios.

So, the Size of the Use Case is (basically) the number of StoryPoints it will take to implement enough Scenarios to make the Use Case Releasable.

How many is that??

We have the S-Shaped Curve

S-Shaped Curve

- Minimally Releasable
- Architecture Infrastructure
- Must-Haves
- Buffer, Nice-to-Haves, Rework

% Value Achieved

% Budgeted Effort/Cost Expended
**Aside on Product Owner**

- **80/20** – the perfect, see into the future, Product Owner
- **80/45-50** – the average Product Owner
- **80/80** – the totally random, doesn’t have a clue, Product Owner

---

**We have the S-Shaped Curve**

**S-Shaped Curve**

- Minimally Releasable
- Buffer, Nice-to-Haves, Rework
- Architecture Infrastructure
- Must-Haves
Let’s convert this to CFPs and SPs

Atomic Use Case (ideal)

% Must-Have FPs

% Budgeted StoryPoints Expended

Minimally Releasable

Architecturally Significant Must-Have FPs

Must-Have FPs

Buffer, Nice-to-Have FPs, Rework, Analysis, Etc…

What Really Happens...

Atomic Use Case (typical)

% Must-Have FPs

% Budgeted StoryPoints Expended

bug fix

analysis

nice-to-have functionality

Interface clean-up
So, now that we know the story…

▶ Let’s Estimate Atomic Use Cases
▶ Case 1: you “know” all the must-have scenarios
   ▶ Then, simply estimate them in StoryPoints (ignoring architecture), and multiply by 2
   ▶ Note: if your developers like to overbuild their architecture, multiply by 3, just sayin…
▶ Case 2: you haven’t got a clue…
   ▶ Then use Domain-Specific Estimates for Small, Medium, and Large Atomic Use Cases…
   ▶ Next Page…

Standard Sizing…

▶ This is based on my experience with web-based applications – Your Mileage May Vary
▶ Typical Scenario has 5 SPs (average of 2,4,8)

<table>
<thead>
<tr>
<th>Size of the Atomic Use Case</th>
<th># of must-have Stories*</th>
<th>StoryPoints(UseCase)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small (S)</td>
<td>5</td>
<td>75 SPs</td>
</tr>
<tr>
<td>Medium (M)</td>
<td>10</td>
<td>150 SPs</td>
</tr>
<tr>
<td>Large (L)</td>
<td>20</td>
<td>300 SPs</td>
</tr>
</tbody>
</table>

*Beyond the architecture/backbone
Example: Buy an e-Ticket

<table>
<thead>
<tr>
<th>usecase</th>
<th>Buy an e-Ticket (132 SPs used, 150 SPs estimated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[backbone]</td>
<td>Get List of Flights from CUTLASS (ugly interface)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Capture Itinerary Information</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Capture Passenger Information</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Reserve Flight in CUTLASS</td>
</tr>
<tr>
<td>[alt]</td>
<td>Modify CUTLASS to understand when Flight is Full (note: was awful!)</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Analysis Meeting with SirJeff</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Pick One Flight and Pay for it (note: stubbed out actual payment)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle Round Trip Flights (ugly Interface)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Issue email Confirmation to Customer</td>
</tr>
<tr>
<td>[interface]</td>
<td>Improve Interface for buying e-ticket</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Hook Up Actual Visa/MasterCard Processing Widget (note: was a PITA)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Reserve Flight to Pay upon arrival at Airport</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle multiple-Passenger Parties</td>
</tr>
<tr>
<td>[interface]</td>
<td>Web Interface for Adding/Modifying Flight Info</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Get Luggage Info, including Scuba Tanks</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Exploratory Testing to &quot;See What's Left&quot; for Buy an e-Ticket</td>
</tr>
<tr>
<td>[spt]</td>
<td>Fix Bug in Luggage Weight Calculations</td>
</tr>
<tr>
<td>[bug]</td>
<td>Fix Small List of Bugs found in Exploratory Testing</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with AMEX</td>
</tr>
<tr>
<td>[alt]</td>
<td>Bring Pet on Board</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Seat Belt Extender Needed for 'large' Passenger</td>
</tr>
<tr>
<td>[alt]</td>
<td>Select Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Needs (wheelchair, etc)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Change Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Meals</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle multiple-Passenger Parties</td>
</tr>
<tr>
<td>[interface]</td>
<td>Improve Interface for buying e-ticket</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Close Reservations when Flight is Full (note: turned out to be easy)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle Round Trip Flights (ugly Interface)</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Hook Up Actual Visa/MasterCard Processing Widget (note: was a PITA)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Reserve Flight in CUTLASS</td>
</tr>
<tr>
<td>[alt]</td>
<td>Modify CUTLASS to understand when Flight is Full (note: was awful!)</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Analysis Meeting with SirJeff</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Pick One Flight and Pay for it (note: stubbed out actual payment)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle Round Trip Flights (ugly Interface)</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Hook Up Actual Visa/MasterCard Processing Widget (note: was a PITA)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Issue email Confirmation to Customer</td>
</tr>
<tr>
<td>[interface]</td>
<td>Improve Interface for buying e-ticket</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Close Reservations when Flight is Full (note: turned out to be easy)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle multiple-Passenger Parties</td>
</tr>
<tr>
<td>[interface]</td>
<td>Web Interface for Adding/Modifying Flight Info</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Get Luggage Info, including Scuba Tanks</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Exploratory Testing to &quot;See What's Left&quot; for Buy an e-Ticket</td>
</tr>
<tr>
<td>[spt]</td>
<td>Fix Bug in Luggage Weight Calculations</td>
</tr>
<tr>
<td>[bug]</td>
<td>Fix Small List of Bugs found in Exploratory Testing</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with AMEX</td>
</tr>
<tr>
<td>[alt]</td>
<td>Bring Pet on Board</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Seat Belt Extender Needed for 'large' Passenger</td>
</tr>
<tr>
<td>[alt]</td>
<td>Select Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Needs (wheelchair, etc)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Change Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Meals</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle multiple-Passenger Parties</td>
</tr>
<tr>
<td>[interface]</td>
<td>Improve Interface for buying e-ticket</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Close Reservations when Flight is Full (note: turned out to be easy)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle Round Trip Flights (ugly Interface)</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Hook Up Actual Visa/MasterCard Processing Widget (note: was a PITA)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Reserve Flight in CUTLASS</td>
</tr>
<tr>
<td>[alt]</td>
<td>Modify CUTLASS to understand when Flight is Full (note: was awful!)</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Analysis Meeting with SirJeff</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Pick One Flight and Pay for it (note: stubbed out actual payment)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle Round Trip Flights (ugly Interface)</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Hook Up Actual Visa/MasterCard Processing Widget (note: was a PITA)</td>
</tr>
<tr>
<td>[backbone]</td>
<td>Issue email Confirmation to Customer</td>
</tr>
<tr>
<td>[interface]</td>
<td>Improve Interface for buying e-ticket</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Close Reservations when Flight is Full (note: turned out to be easy)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Add Payment with PayPal (note: really straightforward)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Handle multiple-Passenger Parties</td>
</tr>
<tr>
<td>[interface]</td>
<td>Web Interface for Adding/Modifying Flight Info</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Get Luggage Info, including Scuba Tanks</td>
</tr>
<tr>
<td>[analysis]</td>
<td>Exploratory Testing to &quot;See What's Left&quot; for Buy an e-Ticket</td>
</tr>
<tr>
<td>[spt]</td>
<td>Fix Bug in Luggage Weight Calculations</td>
</tr>
<tr>
<td>[bug]</td>
<td>Fix Small List of Bugs found in Exploratory Testing</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with AMEX</td>
</tr>
<tr>
<td>[alt]</td>
<td>Bring Pet on Board</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Seat Belt Extender Needed for 'large' Passenger</td>
</tr>
<tr>
<td>[alt]</td>
<td>Select Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Needs (wheelchair, etc)</td>
</tr>
<tr>
<td>[alt]</td>
<td>Change Seat online</td>
</tr>
<tr>
<td>[alt]</td>
<td>Pay with Coupon</td>
</tr>
<tr>
<td>[beefup]</td>
<td>Special Meals</td>
</tr>
</tbody>
</table>

Sizing PlusUps

'Add ability to choose Meals'
**PlusUps**

PlusUps are functional Epics that “add” functionality to an existing Use Case.

**80/50 Curve**

- Minimally Releasable
- Must-Haves
- Buffer, Nice-to-Haves, Rework

---

**Do the Same Sort of Analysis**

- If we know the must-haves, size them and multiply by 1.7
- If you have no clue, the table looks like this (values are rounded to nearest 5 SPs)

<table>
<thead>
<tr>
<th>Size of the PlusUp</th>
<th># of must-have Stories</th>
<th>StoryPoints(PlusUp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small (S)</td>
<td>5</td>
<td>45 SPs</td>
</tr>
<tr>
<td>Medium (M)</td>
<td>10</td>
<td>85 SPs</td>
</tr>
<tr>
<td>Large (L)</td>
<td>20</td>
<td>170 SPs</td>
</tr>
</tbody>
</table>
Containers

▲ Are purely for holding a budget of StoryPoints. There is no specific doneness criteria involved except that you have “enough” SPs in there.

▲ Therefore, it’s purely a judgement call on your part.

▲ Good luck
A Sample of Sprint Planning

- Long Story short – this table represents the Budget Estimates of various Epics
- These are the result of estimations as we just discussed, and discussions and negotiation between the Team, Product Owner, and Stakeholders...

<table>
<thead>
<tr>
<th>Capability/Item</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy an e-Ticket</td>
<td>113SP</td>
</tr>
<tr>
<td>Investigate CUTLASS interface/capabilities</td>
<td>10SP</td>
</tr>
<tr>
<td>Investigate the basics of Pilot Timesheets</td>
<td>20SP</td>
</tr>
<tr>
<td>Check Status of Flights</td>
<td>40SP</td>
</tr>
<tr>
<td>SouvSite Maintenance (before Release Sprint)</td>
<td>21SP</td>
</tr>
<tr>
<td>Chores (before Release Sprint)</td>
<td>76SP</td>
</tr>
<tr>
<td>Release Sprint (includes SouvSite Maint and Chores)</td>
<td>60SP</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>340SP</strong></td>
</tr>
</tbody>
</table>
That’s Enough!!

Any Final Questions?

Thank You Very Much!
Join Our Scrum Community!

@scrum-coach

facebook.com/3Back

3back.com/linkedin