Agile Metrics

Agile 2009, August 26, 2009

Dan Rawsthorne
Certified Scrum Trainer
Senior Coach
Danube Technologies, Inc.
dan@danube.com

Tim Gallagher
Certified Scrum Master
Agile Coach
Intel Corporation
tim.gallagher@intel.com

© 2009 Danube Technologies, Inc. All Rights Reserved.

Agenda

- The Basics
  - How Scrum Works
  - Backlog and Stories
  - Discussion of Story Points
  - Good and Bad Behaviors
- Progress within a Sprint
  - Task Hour Burndown
  - Checklist Item Burnup
  - Story Point Burnup
  - Graduated Story Point Burnup
- Production across Sprints
  - Velocities
  - Burndowns
  - Running Tested Features (RTF)
- Monitoring a Sample Release
  - The Release Plan
  - Release Burnup
  - Earned Value Metrics (CPI and SPI)
  - Earned Business Value

© 2009 Danube Technologies, Inc. All Rights Reserved.
Agile Metrics

The Basics

How Scrum Works
Backlog and Stories
Discussion of Story Points
Good and Bad Behaviors

Scrum Flow Each Sprint

Sprint Planning
Sprint Backlog
Product Backlog
Sprint Review Retrospective

Daily Standup
Spring/Iteration
Reports/Metrics/Impediments

Demonstrably Done, Reviewable, Increment of Product

Product
Agile Metrics

Backlog Prioritization

Iron Triangle and “Doneness”

- Agile Planning is actually about balancing effort, scope, and technical debt
- The expected scope and debt is seldom documented in detail, but there is a virtual list (which I often like to make explicit)
- DoneList (acceptance criteria) has two parts
  - Scope Side, usually defined by tests, time boxes, etc
  - Debt Side, usually defined by inspections, process steps, etc
- The DoneLists have different qualities for different types of stories (storyotypes), and lead to specific tasks to get the DoneList completed
Sample Story 1

Get List of Flights from CUTLASS
Size: 8 SPs  
Type: [backbone]

As a flyer I want to have a list of flights that matches my itinerary so that I can choose one that works for me

Agreement:
- Joe is the expert on CUTLASS
- Simplifying Assumptions: One Way, Single Leg, No Seat Selection, Single Passenger, Full Fare, No Luggage ...

Doneness Criteria:
- Review Architectural Decisions with Team
- Design Review
- Review Functional Test Strategy
- Review Unit Tests
- Verify Tests passing on Development Machine
- Code Review
- Functional Tests Written
- Verify Tests (including regressions) passing on Integration Box
- Add Tests to Regression Test Suite

Sample Story 2

Install Copy of CUTLASS in Lab
Size: 8 SPs  
Type: [enviro]

As a developer I want to have my own copy of CUTLASS to play with so that I can figure out how it works

Agreement:
- SirJeff is expert on system - he wrote it!

Doneness Criteria:
- Get CUTLASS Install from SirJeff
- Set up clean machine
- Install CUTLASS
- Do Smoke Test to see if it works
Two Types of Stories

• Features/Functional Stories
  • Have moving parts in the code
  • Often a scenario of a Use Case
  • Have a sequence diagram associated with them – at least in theory

• All the Rest (non-Functional Stories)
  • Analysis
  • CleanUp stories
  • Chores
  • Most Bugs (as only involve part of a scenario, or fix an existing scenario – not new work
  • Etc

• We want sizes (in Story Points) for all of these types

Why Do We Want a Story’s Size?

• Typically, people say they want Sizes so that:
  1. We can know how much time it will take to do the Story
  2. We can know how many Stories (Story Points) we can commit to in a Sprint
  3. We can do long-range planning based on this (velocity) information

• The first two are BAD reasons...
  • Can’t be accurate enough for the first two (see next slide)
  • Unless we’re constantly updating our estimates...
  • Which negates the third reason

• So, we’re left with the third one...
  • Instead of, NOT in addition to, the first two...
Agile Metrics

Why is it Hard to Estimate Effort for a Story?

- Effort Estimation is difficult, because (for functional stories, at least) it depends on:
  - Our understanding of the problem (def’n of “done”)
  - The complexity of the logical design for implementation
  - The Technical Debt present to impede us (20:1 range in difficulty)
  - Who will work on it (10:1 range in productivity)
- The only time we can possibly know all these things is during Sprint Planning, as we’re committing to doing the work
  - We’ve finalized “done”
  - We can take a peek at the code
  - We know who’s available
- This is why we determine effort at the task effort, when we are committing to the stories during planning
- Not before...

So, What is a Story’s Size?

- Let’s listen to folks that have thought about it a lot, we don’t need to re-invent the wheel here
  - For Functional Stories, we could try to measure Functional Size...
  - People have been thinking about FSM (Functional Size Measurement) for 30+ years, and have some real good ideas
  - Basically, FSM methods try to measure the “size” of the logical design, without taking into consideration technical or quality requirements. That is, they try to count the “moving parts” in the logical design
  - Various methods include Function Point Analysis, Use Case Points, Feature Points, and so on.
- These are a pain to use (mostly), and usually require too much design up front to be able to calculate
  - So we give up?
  - No, we use our estimation game with the right question...
The Rules of the Estimation Game (my way...)

- Give Each Estimator a set of Cards
- Ask an appropriate Question
  - How Big is This Dog?
  - Compared to ABC, how big is XYZ?
  - How much does this Risk scare you?
  - How much Effort is this Task?
- Have every Estimator show their cards, all at once
- If everybody agrees, we’re done
- If there is disagreement, have an interesting conversation
  - You may limit the time given for conversation
  - Conversation about "why do we disagree", rather than the original question. This is very useful...
- If necessary, vote again, and repeat until done

Summary of Story Size Estimation

- Pick a "typical" M-Sized Story – possibly "adding a simple secondary scenario for an existing use case"
- For Functional Stories, compare the story to the "basic M-Sized" Story
  - Use Estimation game with question: "How big is this one, in terms of moving parts, compared to our 'typical M-Sized one', given that the codebase is the same, the same people work on it, and so on"
  - Add some extra points if it is "architecturally significant"
- For non-Functional stories with well-defined definitions of "done" compare the story to the "basic M-Sized" Story
  - Use Estimation game with question: "How hard is this one, in terms of effort, compared to our 'typical M-Sized one', given that ..."
- For Stories with ill-defined definitions of "done", timebox them
  - “Do 8 hours of Exploratory testing on page ABC”
  - “Do a Small Story's worth of work cleaning up the code in module XYZ”
Agile Metrics

Discussion

• Talk amongst ourselves...

• Topic: Story Points, Effort or Size?

• 5 minutes

Purpose of Metrics

• Provide meaningful information about important stuff
  • Who uses the metric?
  • Why do they use it?
  • What does it tell them?

• Provide force for good
  • “you get what you measure”
  • People will try to “game” the metric

• Be Simple and Understandable
  • Don’t do too many things at once
  • Focus on 1-2 things
  • Eliminate Noise

• Remember: metrics don’t manage, people do...
Agile Metrics

Good Behavior – Small Bites

- A team moves quickest when its work is in small, easily understood, chunks
  - A typical story should be 2-3 people for 1-4 days. Basically, up to 10 person days
  - A typical task should take a person (or pair) from 1/2 to 2 days to do. We like to see each person (or pair) complete at least one task a day
- Other coaches and consultants have different ideas about this – but this is in the right ballpark

Good Behavior – Quality Product

- Use the XP practices for quality code
  - Extensible code
  - Maintainable Code
  - Testable (Tested) Code
  - Understandable code
  - Consistent practices
  - Pairing, Peering, Coaching, etc.
- We would like our metrics to force, not fight against, these behaviors
  - We want metrics that fight against Technical Debt, not those that force it
Agile Metrics

Bad Behavior – Technical Debt

- 22 SP
- Dead Legacy, Velocity = 0
- 18 SP/Sprint
- 16 SP/Sprint
- Time
- StoryPoints
- no

Good Behavior - Team Swarm

- Stakeholders
- PO/Analysts
- Testers
- Coders
- To The Backlog
- To The Product
Agile Metrics

Progress Within a Sprint

Task Hour Burndown
Checklist Item Burnup
Story Point Burnup
Graduated Story Point Burnup

Purpose

• These metrics measure the progress within a Sprint
• the purpose is to help the team predict whether they will complete the work they signed up for within the Sprint
  • Shows need for replanning to make work to fit into the Sprint
  • If too much work in Sprint, force is to let quality slip in order to “make a number” – BAD THING
  • If too little work in Sprint, gives ProductOwner warning to start thinking about what else needs to be done
What do we Measure?

- We can measure anything that is estimatable and supposed to drop to 0 at the end of the Sprint.
- If the total is fixed, we can do a burnup instead.

Sprint (task hour) Burndown
Sprint (Task Hour Remaining) Burndown

- What we normally see in the books and in our tools
  - To calculate, we must estimate the time remaining on a task every day (note: the time actually spent on the task is irrelevant here)
  - We graph the total estimated Task Hours remaining each day
    - This should burn down to 0
    - This usually trends to 0 after the first 1/3 of the Sprint or so (empirical observation)
    - It almost always goes up at the beginning
      - Finding new tasks
      - Finding that existing tasks are harder than we thought

Sample Data for Task Hour Burndown

- This shows “good” behavior, with the team swarming on each Story until it’s done
Discussion of the Task Hour Burndown

- These are useful for measuring work remaining, they don’t measure actual progress
  - It would be possible to have 10 hours remaining but no stories completed – BAD
  - Can extrapolate against a Task Hour velocity, which is GOOD
  - If we do “most risky” tasks first, it settles down fairly quickly
- Hard to explain that the shape is normally like this
- Doesn’t force any good behavior at all on the team
  - Force is to do fewer hours per task, which could cause quality slippage
Agile Metrics

Checklist Item Burnup

Sample Data for Checklist Item Burnup
Discussion of the Checklist Item Burnup

- These are useful for measuring progress remaining, but not work
  - It would be possible to have only one Item left, but lots of work left – BAD
  - Measures progress against things that actually provide value – GOOD
- It always goes up, and has an upper bound
- Doesn’t force any good behavior at all on the team
  - Force is to do fewer hours to get an CheckList Item done, which could cause quality slippage
Story Point Burnup(s)

- It is a common theme in agility to count or observe stories that are completed
  - They meet their exit criteria
  - If stories are about product, this means that they are demonstrable or potentially shippable or some reasonable definition of “done”

- We have two graphs here...
  - # Story Points completed in Sprint
  - Proportional # Story Points completed
    - Get partial credit when Checklist Item is done
    - Could also get partial credit when Task is done

- Could, of course, be Burndowns
Discussion of the Story Point Burnups

- I like the Story Point Burnup
  - Its measures actual progress - **GOOD**
  - It provides a force to do the Team Swarm on stories - **GOOD**
  - It requires relatively small stories, so the graph is granular enough, so is a force to do smaller stories – **GOOD**
  - Can do the graduated one to get partial credit - ???
  - These are useful for measuring progress remaining, but not work - **BAD**
- So, I like to use this one along with the Task Hour Burndown, so that I get a balance between actual progress and effort...
Agile Metrics

Discussion

• Talk amongst ourselves...

• Topic: Measuring Progress with a Sprint

• 5 minutes

Production Across Sprints

Velocity
Burndowns
Running Tested Features (RTF)
Purpose of Velocity

- To understand how fast the team does work
- To estimate team capacity, so we can predict duration and extrapolate schedule
- Definitions
  - Velocity – measure of the team’s *proven* rate of progress per iteration
  - Capacity – the team’s *assumed* rate of progress per iteration
- Yesterdays Weather says that Capacity = Velocity until *proven* otherwise
- Law of Large Numbers (statistics) says that after 30-40 stories velocity is a good estimate of capacity
Calculating Velocity

- Count the number of completed SPs in each sprint (no partial credit)
  - Call these values $SP_1$, $SP_2$, $SP_3$, ..., $SP_n$
- Straight-line average calculation:
  - $Vel_n = \frac{SP_1 + SP_2 + SP_3 + ... + SP_n}{n}$

Discussion (good and bad)

- All work must be estimated for predictions to be valid
- Capacity changes for many reasons
  - Team makeup changes
  - Improvements in process, knowledge
  - Etc.
- Velocity will be a trailing indicator of these changes
  - But it’s the best we’ve got
  - Teams have tried to modify their velocity calculations to account for other factors, but it is difficult and risky
Warnings

- If it is a "good thing" that Velocity goes up, then
  - Could force Technical debt because there is a force to make each story "easier", or
  - It could force the team to estimate more StoryPoints for same work
    - Quality remains good, but
    - Velocity is no longer consistent, thus invalid for prediction

- If the ProductOwner wants more features for the same SPs (wants more done with same budget), then
  - Force on team to underestimate SPs for stories, and either
    - Calculated velocity may go down, thus invalidating predictions, or
    - Quality ma go down to achieve same calculated velocity

*Using Velocity badly can cause Technical Debt*

Product Burndown

---

© 2009 Danube Technologies, Inc. All Rights Reserved.
Product Burndown

- The Product Burndown shows the amount of known work remaining at the end of each Sprint
  - Typically, it’s the known work in the current release, so it is often called the Release Burndown
  - In our formulation (with stories), the amount of remaining work is given in StoryPoints. As usual, partial credit is not given, a story is either done or not done
  - Takes into account: work completed, added, removed, and resized
  - Putting a large SP value on an epic is actually supplying a budget, not an estimate – be aware of this...
- The basic idea is that by looking at the Release Burndown the Product Owner and Stakeholders can determine if the project or release is “on schedule”
  - Do we need to de-scope?
  - Can we actually add some more stuff?

A Well-Behaved Product Burndown

- From our friends at Conchango, www.Conchango.com
What we Usually Get

- A Pattern called “get enough and go”
- “Scope Creep” is an expected part of agile projects
- Makes this graph hard to understand

Enhanced Product Burndown
So, We get the Enhanced Burndown

- In order to determine whether a Project/Release Burndown’s shape is due to progress or scope creep, Mike Cohn introduced the Enhanced Burndown chart*. We’ll look at a couple on the next two pages...
  - This chart shows a series of bars, with the height of the bar representing the StoryPoints remaining
  - The heights above the baseline indicates work that’s been done
  - The height of the bar below the baseline represents work that has been added or removed

* [http://mountaingoatsoftware.com/alt_releaseburndown](http://mountaingoatsoftware.com/alt_releaseburndown)

Enhanced Burndown (converging)
**Discussion**

- I like the enhanced burndown graphs
  - The top line is equivalent to the velocity line I showed previously, so we can see the velocity in action
  - We can also see scope creep in action, by looking at the bottom line
- The only problem is when it diverges
  - This indicates that we need to de-scope the release, or
  - Base our decision about releasability on some other factor than “all the work is done” – see Earned Business Value later...
Providing Value

- Agile Teams provide externally-visible value – it is, by definition, the only thing of value we produce (and I mean business value, this time 😊)
- So, we need a way to measure how much externally-visible value we are producing. This is hard, and we discuss two attempts to do this with metrics
  - Measuring Acceptance Tests
  - Running Tested Features
Value in Software

• It is a truism in software that if you don’t have:
  • A test to prove something works, and
  • A regression test to prove it continues to work, then
  • You don’t really have the feature
• So, the simplest measure of value is running tests. You can prove you have something, and if you keep running the tests, you can prove you still have it

Acceptance Test Metrics

• One of the early (and fairly easy) metrics to collect about value is in this graph
• It measures
  • How many tests you have defined for your system
  • How many are running in the regression test suite
  • How many are passing each sprint
Discussion of Acceptance Tests

• This is a great set of metrics that is easily produced...
  • Measuring what counts
  • Provides force to do more testing
• But
  • There is no intrinsic way to know the coverage of the Acceptance Tests given in this graph
    • Positive tests – the system does what it’s supposed to
    • Negative tests – the system doesn’t do what it’s not supposed to
  • Can’t tell them apart
  • You can make the graph look good by writing lots of little, trivial acceptance tests that “just pass”

Running Tested Features

• As we have been saying all along, stories are units of work that provide value to a project
  • We don’t get credit for them until they are done
• However, external value of a project is only provided by stories that add features to the product. That is, they are feature stories, which are a subset of all stories
• Ron Jeffries introduced Running Tested Features as a metric that forces agility*
  • This extends the previous notion by focusing on stories, not just tests
  • It is tightly coupled with the burndown/burnup metrics, but focuses only on feature stories

Calculating RTF

- Calculating RTF is relatively simple:
  - A story is said to “pass” if all of its acceptance tests pass
  - Otherwise, it fails
- Only count stories that provide business value;
  that is, only count “user” stories
- Run the regression test, and count them up –
  that’s the RTF
- Do this at the end of the Sprint and you
  get a snapshot of value

Discussion of RTF

The RTF graph should look like →, as
the team should be adding features
early and often and consistently

If it looks like →, there might have
been too much waterfallish behavior
going on – merits a conversation

This → behavior indicates that things
broke, and tests were failing.
Questions must be asked
More RTF Comments

- RTF is a nice metric
  - It provides a force to make smaller and small stories, which is good (if not great)
  - Ultimately, it gives one story per “positive” acceptance test
- However:
  - It requires our regression test suite to be divided up by story, so that we know which stories are broken when tests don’t pass
  - It treats all stories as “equal”; that is, no story is worth more than another – they all provide equally to the count

Discussion

- Talk amongst ourselves...
- Topic:
  - How do you measure progress from Sprint to Sprint?
  - Are simple velocities good enough for you?
  - And your management?
- 5 minutes
Agile Metrics

Monitoring a Release

The Release Plan
Release Burnup
Earned Value Metrics (CPI and SPI)
Earned Business Value

Releases are Different...

- There are targets
  - Target Dates
  - Target Budgets
  - Target Features
- So, our metrics are different
  - They don’t just measure progress
  - They measure comparisons to targets
- We are answering questions like
  - “How done are we?”
  - “How much further?”
- Not
  - “Where are we?”
  - "How fast are we moving?"
Agile Metrics

Types of Release Planning

- There are three kinds of Releases
  - When Can we Release?
    - Known scope, existing team
  - What Can we Release?
    - Known time, existing team
  - How much will it cost?
    - Known time, known scope, new team
    - This is the tough one...
  - Release Planning is out of scope for this talk, but once we’ve planned, we’ve got
    - A Date
    - Some Scope to produce
    - And a Team
  - Our metrics measure progress against our plan

<table>
<thead>
<tr>
<th>Time (No of Sprints)</th>
<th>Scope (Total SPs)</th>
<th>Capacity (SPs/Sprint)</th>
</tr>
</thead>
<tbody>
<tr>
<td>??</td>
<td>XX</td>
<td>XX</td>
</tr>
<tr>
<td>XX</td>
<td>??</td>
<td>XX</td>
</tr>
<tr>
<td>XX</td>
<td>XX</td>
<td>??</td>
</tr>
</tbody>
</table>

Time x Capacity = Scope
What’s Going On Now

• Have a team
  • Transitioning from SouvSite development
    • Half the team for the first 2 sprints
    • Whole team from then on
  • Velocity of 60 SPs/Sprint (including chores)
  • Expense is 480 Hours/Sprint for the team
• Want to release something to SirJeff in 3 months (7 sprints)
  • What he can tell his marketing and sales folks will be there?
• SirJeff’s priorities are:
  1. Buy an e-Ticket
  2. Check Status of Flights
  3. Pilot Timesheets
  4. The rest…

Capacity Calculations

• Team of 8, half of them for first 2 sprints, Joe getting married Sprint 5-6
Baseline Budget for Epics

- After working with SirJeff and our Team for 2-3 days, we come up with the basics of our Release Plan, as follows...

<table>
<thead>
<tr>
<th>Capability/Item</th>
<th>BV</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy an e-Ticket</td>
<td>80%</td>
<td>108SPs</td>
</tr>
<tr>
<td>Investigate CUTLASS interface/capabilities</td>
<td></td>
<td>10 SPs</td>
</tr>
<tr>
<td>Investigate the basics of Pilot Timesheets</td>
<td>10%</td>
<td>20 SPs</td>
</tr>
<tr>
<td>Check Status of Flights</td>
<td>10%</td>
<td>40 SPs</td>
</tr>
<tr>
<td>SouvSite Maintenance</td>
<td></td>
<td>27 SPs</td>
</tr>
<tr>
<td>Release Activities</td>
<td></td>
<td>38 SPs</td>
</tr>
<tr>
<td>Chores</td>
<td></td>
<td>92 SPs</td>
</tr>
<tr>
<td>Total</td>
<td>100%</td>
<td>335 SPs</td>
</tr>
</tbody>
</table>

Release Game Plan

- And here’s our baseline for “spending” our SPs and hours

<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>
<tr>
<td>SPs</td>
<td>30</td>
<td>30</td>
<td>45</td>
<td>60</td>
<td>58</td>
<td>52</td>
<td>60</td>
<td>335</td>
</tr>
</tbody>
</table>

- Notice how this looks like a “real” plan
Agile Metrics

Monitoring the Release

Release BurnUp
Earned Value Metrics
Earned Business Value

May Need to Produce and Present Metrics

- Sprint/Product Burndowns

- Velocity Calculations

- Earned Value (EV)

- Earned Business Value (EBV)
How the Release Actually Played Out

### How Release Played Out (SPs)

<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>253</td>
<td>233</td>
<td>495</td>
<td>530</td>
<td>467</td>
<td>445</td>
<td>498</td>
<td>2921</td>
</tr>
<tr>
<td>SPs</td>
<td>26</td>
<td>32</td>
<td>42</td>
<td>54</td>
<td>66</td>
<td>52</td>
<td>54</td>
<td>326</td>
</tr>
</tbody>
</table>

### How Release Played Out (PersonHours)

<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>253</td>
<td>233</td>
<td>495</td>
<td>530</td>
<td>467</td>
<td>445</td>
<td>498</td>
<td>2921</td>
</tr>
<tr>
<td>SPs</td>
<td>26</td>
<td>32</td>
<td>42</td>
<td>54</td>
<td>66</td>
<td>52</td>
<td>54</td>
<td>326</td>
</tr>
</tbody>
</table>

**Stories for “Buy an e-Ticket”**

<table>
<thead>
<tr>
<th>Work (found)</th>
<th>Done</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>[backbone] Get List of Flights from CUTLASS (ugly interface)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>[backbone] Capture Itinerary Information</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>[backbone] Reserve Flight in CUTLASS</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>[alt] Modify CUTLASS to Understand When Flight is Full (note was awful)</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>[analysis] Analysis Meeting with SirJeff</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>[backbone] Pick One Flight and Pay for it (note: stubbed out actual payment)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>[alt] Handle Round Trip Flights (ugly Interface)</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>[beefup] Hook Up Actual Visa/StandardCard Processing Widget (note: was a PITA)</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>[backbone] Issue email Confirmation to Customer</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>[interface] Improve Interface for buying e-ticket</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>[beefup] Close Reservations when Plane is Full (note: turned out to be easy)</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>[alt] Add Payment with PayPal (note: really straightforward)</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>[alt] Reserve Flight to Pay upon arrival at Airport</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>[alt] Handle multiple-Passenger Parties</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>[interface] Web Interface for Adding/Modifying Flight Info</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>[beefup] Get Luggage Info, including Scuba Tanks</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>[analysis] Exploratory Testing to “See What’s Left” for Buy an e-Ticket</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>[spt] Work with SirJeff’s Marketing folks to make sure their materials are correct</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>[bug] Fix Bug in Luggage Weight Calculations</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>[bug] Fix Small List of Bugs found inExploratory Testing</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>[alt] Pay with Coupon</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>[alt] Pay with AMEX</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>[beefup] Seat Belt Extender Needed for “large” Passenger</td>
<td>3</td>
<td>S</td>
</tr>
<tr>
<td>[alt] Bring Pet on Board</td>
<td>5</td>
<td>M</td>
</tr>
<tr>
<td>[alt] Comfort Seat for “really large” Passenger</td>
<td>5</td>
<td>S</td>
</tr>
<tr>
<td>[alt] Pay with PayPal</td>
<td>3</td>
<td>M</td>
</tr>
<tr>
<td>[beefup] Special Meats</td>
<td>5</td>
<td>S</td>
</tr>
<tr>
<td>[beefup] Special Needs (wheelchairs, etc)</td>
<td>5</td>
<td>S</td>
</tr>
</tbody>
</table>
Agile Metrics

Product/Release BurnUp

Product/Release Burnup Graph

- Measuring against a budget
- Measuring what is done
- Measuring what is “ready” to do – inventory

- This takes both agility and lean into account, to some degree
Agile Earned Value Management (AgileEVM)

• First of all, it’s not about value as it “goodness” or “business value” – it’s value as in “actually done” – and the emphasis is on “earned”
• EVM is about measuring project performance, comparing budgets to actuals in scope, schedule, and resource
• The genius of AgileEVM is the realization that stories are an appropriate thing to measure and count for EVM purposes *
  • In fact, stories are better than the activities we normally measure in software... done/earned is better defined
  • The unit of value that we are measuring is the “earned StoryPoint”; that is, we are measuring stories that are “done”

Agile Metrics

The Metrics CPI and SPI

- CPI answers the question “are we paying what we expected for each SP?”
  - The ratio CPI = (budgeted $/SP) / (actual $/SP)
- SPI answers the question “are we getting the SPs at the rate we expected?”
  - The ratio SPI = (actual SP/Sprint) / (budgeted SP/Sprint)
- We can use these metrics to monitor SirJeff’s project, as seen on the next slide

EVM Metrics (CPI and SPI)

- Measures both kinds of velocity
  - SPI measures SPs/sprint
  - CPI measures SPs/hour
- “boxes” in behaviors
- These two metrics are very nice
  - But they require a baseline
However...

- In an agile project, we are not paid to produce StoryPoints, we are paid to produce Business Value
- Business Value is subjective, and based on our Stakeholder’s needs for this Release
- Here are the features with BV in this release

<table>
<thead>
<tr>
<th>Goal/Feature/Capability</th>
<th>BV</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buy an e-ticket</td>
<td>80%</td>
<td>108 SP</td>
</tr>
<tr>
<td>Investigate the basics of Pilot Timesheets</td>
<td>10%</td>
<td>20 SP</td>
</tr>
<tr>
<td>Check Status of Flights</td>
<td>10%</td>
<td>40 SP</td>
</tr>
</tbody>
</table>

- As we deliver Stories within a feature, the Earned Business Value ($EBV(feature)$) increases, and is a percentage of the Feature’s Business Value ($BV(feature)$)
Calculating Earned Business Value

- Each capability has its own Business Value
  - “S-shaped” curve if there’s an architectural component
  - “80/50” curve if there isn’t
- It is our job to be smart, and agile, enough to stay on this curve
  - But the curve is very forgiving
  - Is based on baseline SP budget for Capability
  - Minimal releasability is 2/3 of the way down, so have a checkpoint...
- And total EBV just uses the BVs of the capabilities themselves to calculate a weighted total as we go along

Here’s our Philosophy

- For each capability, our Release Plan has a Baseline SP budget
- Our goal is to use agility to provide business value the best way we can
  - 80/20 rule is valid in hindsight
  - 80/50 rule is reasonable to expect as we move forward
    - Actually two different curves
    - We’ll see them next
- We expect the last third (EBV > 90%) of our Baseline budget for each capability to be our buffer
  - Using agility, we manage each buffer
  - And tradeoff SPs between capabilities
  - When in a pinch, must be calm and focus on agility and “doneness”
- This is kindof like the Critical Chain Analysis from TOC...
For New Features

- That have an architectural element to them
- In our Case, we use this curve for
  - Buy an e-ticket
  - Check Status of Flights

For New/Existing Features

- Without an architectural element to add this Release
- In our Case, we use this curve for
  - Investigate the Basics of Pilot Timesheets
Discussion of Earned Business Value

- EBV (story) is determined by a calculation
  - Doesn’t require evaluation at story level, it is at feature, or epic, level – GOOD
  - Relies on having a “correct” budget for a feature – BAD
- Calculation is only as good as your original SP budgets
  - Relying on initial working with Stakeholders - BAD
  - Gives checkpoints to determine if overall budget is “correct” for feature – GOOD
- What we are measuring makes sense to Stakeholders
  - If they’ve worked in high-ceremony organizations, they are used to, and expect, graphs like these - GOOD
  - Can use agility to succeed in spite of ourselves - GOOD

Earned Business Value Graphs

- Total Earned Business Value
- EBV for “Get e-Ticket”
- EBV for “Investigate Pilot Timesheets”
- EBV for “Status of Flights”

Correction In Sprint 3
Summary

• OKAY, that’s enough!
• To summarize
  • Don’t measure something just because you can
  • Measure things that either:
    • Tell you something about your team
    • Tell you something about your product
    • Tell you something about your progress
    • Pay attention to the forces that will be unleashed when you measure something

• Good Luck!

Any Final Questions?
Thank You Very Much!