Agile estimating and planning

Presented by Simon Baker

think-box Limited

http://www.think-box.co.uk

Produced with permission from Mike Cohn
Author of User Stories Applied for Agile Software Development
This material is based on his forthcoming book Agile Estimating and Planning

Agile Alliance member

Agenda

- ☐ An agile approach
- ☐ Estimating with story points
- ☐ Release planning
- ☐ Estimating with ideal time
- ☐ Iteration Planning
- ☐ Tracking and metrics

An agile approach

plans should be flexible enough to adapt with change

Generic agile process

Let’s define a simple agile process consistent with Scrum and XP

Attributes:

- Software is developed in iterations
- Deliver completed software in each iteration
- Reprioritise often

Develop in iterations

Make progress through successive refinement of functionality

- Take first cut at a piece of functionality knowing it’s incomplete and refine until the functionality is satisfactory
- With each iteration the functionality is improved through the addition of greater detail
- An iteration has a fixed duration
- Requirements, analysis, design, coding, testing all happen concurrently during an iteration

Release in increments

Build and deliver functionality in pieces (increments)

- Each increment represents a complete subset of functionality and is production-quality
- A release comprises one or more iterations that build upon each other to complete a set of functionality
- A release does not need a constant duration like an iteration but it must be an exact multiple of iterations

Deliver completed software

! **We want to always complete the iteration work during the iteration**

? **Complete**

Code that is well written and well factored, checked-in, clean, compiles with coding standards, and passes all tests

- Typical work carried forward includes:
  - » Bug fixing
  - » Exception handling
  - » Code to handle cases that became apparent during iteration

Velocity

! **Measure our rate of progress using velocity**

? **Velocity**

~ Amount of work completed in an iteration

! **Use velocity as our best guess of future progress**

? **Yesterday’s weather**

Today's weather, to a very rough approximation will be the same as yesterday's weather

Reprioritise often

- Use iteration boundaries as an opportunity to adjust priorities
- It is easier to prioritise when all work in an iteration has been completed in that iteration
- Scenario 1:
  - Market research tells us ‘Undo’ support is more valuable than ‘Advanced search’
  - ‘Basic search’ was completed in iteration 1
  - It’s easy to prioritise between ‘Advanced search’ and ‘Undo’ for iteration 2
- Scenario 2:
  - ‘Basic search’ and ‘Advanced search’ are both partially developed in iteration 1
  - Each has bugs to be fixed
  - Should bugs be fixed to make last iteration useful or shall ‘Undo’ be added?

Engineering tasks

Products are built from business-valued functionality

A project planned with engineering tasks rather than business-valued functionality creates a number of problems:

1. We develop a less complete understanding of the system
   - » It's easy to plan an entire project using engineering tasks without understanding the product being built
2. A completed schedule is reviewed by looking for forgotten engineering tasks rather than forgotten / needed business functionality
3. Progress is measured by completion of engineering tasks rather than by completion of business-valued features
   - » Business wants progress in terms of completed and meaningful functionality

User stories

Estimate and plan with business-valued functionality described as user stories

?

User story

A brief description of functionality as viewed by the business

- User stories facilitate face-to-face communication with customer to solidify details
- User stories allow development to start more quickly as details emerge

Planning

Plan at an appropriate level of precision for the distance being considered

1. When far away from something estimate it in one unit

2. When near something estimate it in a different unit

- From far away the team will make mistakes of a certain magnitude

- Looking close up the team will make mistakes of a different magnitude

Release Planning

When planning from a distance, e.g. when looking further out at a release:

- Acknowledge that planning cannot be done with any real precision
- Use **story points** as the estimation unit
- Produce a high-level plan:
  - » That tells us where we expect to be at various checkpoints (the ends of iterations)
  - » Which we can use to tell if the project is ahead or behind where we expected to be

Iteration planning

When planning close up , e.g. when looking at an iteration:

- Plan the engineering tasks to be performed for each user story
- Use **ideal programming hours** as the estimation unit
- Produce a detailed plan at the start of each iteration that adapts to changing requirements or learning

Agenda

- ☑ An agile approach
- ☐ Estimating with story points
- ☐ Release planning
- ☐ Estimating with ideal time
- ☐ Iteration planning
- ☐ Tracking and metrics

Estimating with story points

can I have a large cola please

Magnitude: An everyday concept

In a fast-food joint, you order a large drink as opposed to a small, medium or extra-large one:

- You don't know how many fl.oz's you'll get
- You know large is bigger than small or medium
- You know large is smaller than an extra large
- You know from past experience that when this thirsty, a large drink in other places has been about the right size

Story points are relative

? **Story points**

Are a measure of magnitude, i.e. the 'bigness' based on the time it will take to complete

- From a distance, story points are used to estimate the magnitude of a user story
- The raw values assigned are not important
- What matters is the relative sizes:
  - » A story assigned 2 points should be bigger than a story assigned 1 point
  - » A story assigned 8 points should be smaller than a story assigned 13 points and bigger than a story assigned 5 points

Estimation scale

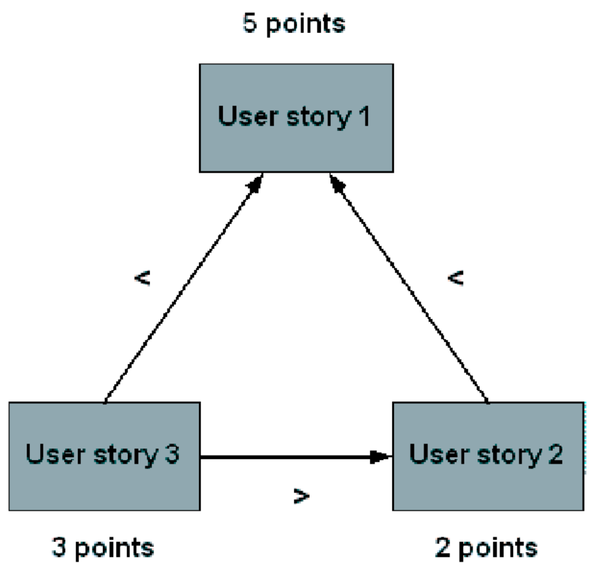
Use the Fibonacci scale to estimate story points: 1,2,3,5,8,13,21, …

- Using such a scale reflects the truth that as estimates get larger, we know less about them
- The spread within the Fibonacci scale avoids discussions about why I assigned 67 story points and you assigned 68 story points
- If I assign a user story 1 story point and you assign it 2 story points it's worth resolving because the difference is 100%
- Focus on pure relative sizing:
  - It's quicker to discuss "this is bigger than that and smaller than the other" than it is to decide "this is twice that"

Triangulation

Triangulate - estimate a story by analogy to two or more other stories

- Compare a user story to one or more other user stories (ideally some will have already been completed)



- Story 3 is bigger than Story 2 and smaller than Story 5. Story 5 is bigger than Story 2

Getting started

- Find what feels like a medium-sized story and assign points somewhere in the middle of the Fibonacci scale

- Each additional story is compared to the first story and assigned points commensurate with story's relative magnitude

Exercise: Dog points [1/3]

Instead of story points, assign ‘dog points’ representing the magnitude of each of these breeds of dog:

- » Labrador retriever
- » Terrier
- » Great Dane
- » Poodle
- » Dachshund
- » German Shepherd
- » St. Bernard
- » Bulldog

▪ What will you decide determines the ‘bigness’ of a dog?

Exercise: Dog points [2/3]

- Based on height:

Breed	Dog points
Labrador Retriever	5
Terrier	3
Great Dane	13
Poodle	3
Daschund	1
German Shepherd	5
St. Bernard	8
Bulldog	3

- Dog points assigned:
  - » Labrador retriever was selected as a medium-sized dog (5)
  - » Great Dane is much bigger (13)
  - » St. Bernard is bigger than a Labrador but smaller than a Great Dane (8)
  - » Dachshund seems about as small as a dog gets (1)
  - » German Shepherd seems about the same size as a Labrador retriever(5)
  - » Bulldog is shorter but bigger that a Dachshund (3)

Exercise: Dog points [3/3]

- Without more detail, it should've been difficult to assign dog points to poodle and terrier
  - There are toy, miniature and standard poodles
  - There are more than 20 breeds of terrier
- When given a loosely-defined user story (or dog):
  - Discuss it with the customer
  - Make some assumptions, take a guess and move on but return when more is known
- Guess for poodle and terrier (3)
  - Even the smallest are smaller than a Labrador Retriever
  - Small poodles and terriers would be 1- or 2-point dogs, but there are bigger breeds too, so on average 3 seems reasonable

Problems with estimating in magnitude

- Values must be distinguishable
  - How do you tell a ‘67’ from a ‘68’? Why bother!
  - This is addressed by the Fibonacci scale
- You have to guess a team’s initial velocity
  - This problem only exists at the start of the first project
  - After one iteration an actual velocity can be used to better predict the duration of the project
- It can feel uncomfortable at first
  - Most developers quickly become accustomed
- Developers may make an implicit conversion
  - “Most 3’s take a day, this seems like a day, I’ll say it will take 3 story points”

Advantages of estimating magnitude

- It’s easier and quicker to estimate
  - It’s a simple comparison – “This is bigger than that and smaller than the other”
  - You no longer need to think about how long a user story will take in elapsed time and about the interruptions that may impact you
- Magnitude is independent of developer speed
  - Developers can agree on magnitude even though it may take them different elapsed times to implement
- You rarely need to re-estimate
  - If a developer estimates in elapsed time and becomes a faster programmer, then his original estimates will be wrong
  - If a developer had estimated in magnitude, his estimates would remain correct – ‘big’ is still ‘big’; even though a developer may be faster

Questions

1. During a release planning meeting three developers are estimating a story. Individually they estimate the story at 2, 4 and 5 story points. Which estimate should they use?

2. What is the purpose of triangulating estimates?

3. Team A finished 43 story points in their last 2-week iteration. Team B is working on a separate project and has twice as many developers. They also completed 43 story points in their last 2-week iteration. How can that be?

Answers

1. They should continue discussing the story until their estimates get closer.

2. Triangulation improves estimates by making sure that each estimate makes sense in relation to multiple other estimates. If a 2-point story seems to be twice a 1-point story, it should also seem to be half of a 4-point story.

3. The story points of one team are not comparable to the story points of any other team. From the information in this questions, we cannot infer that Team A is twice as productive as Team B.

Agenda

- ☑ An agile approach
- ☑ Estimating with story points
- ☐ Release planning
- ☐ Estimating with ideal time
- ☐ Iteration planning
- ☐ Tracking and metrics

Release planning

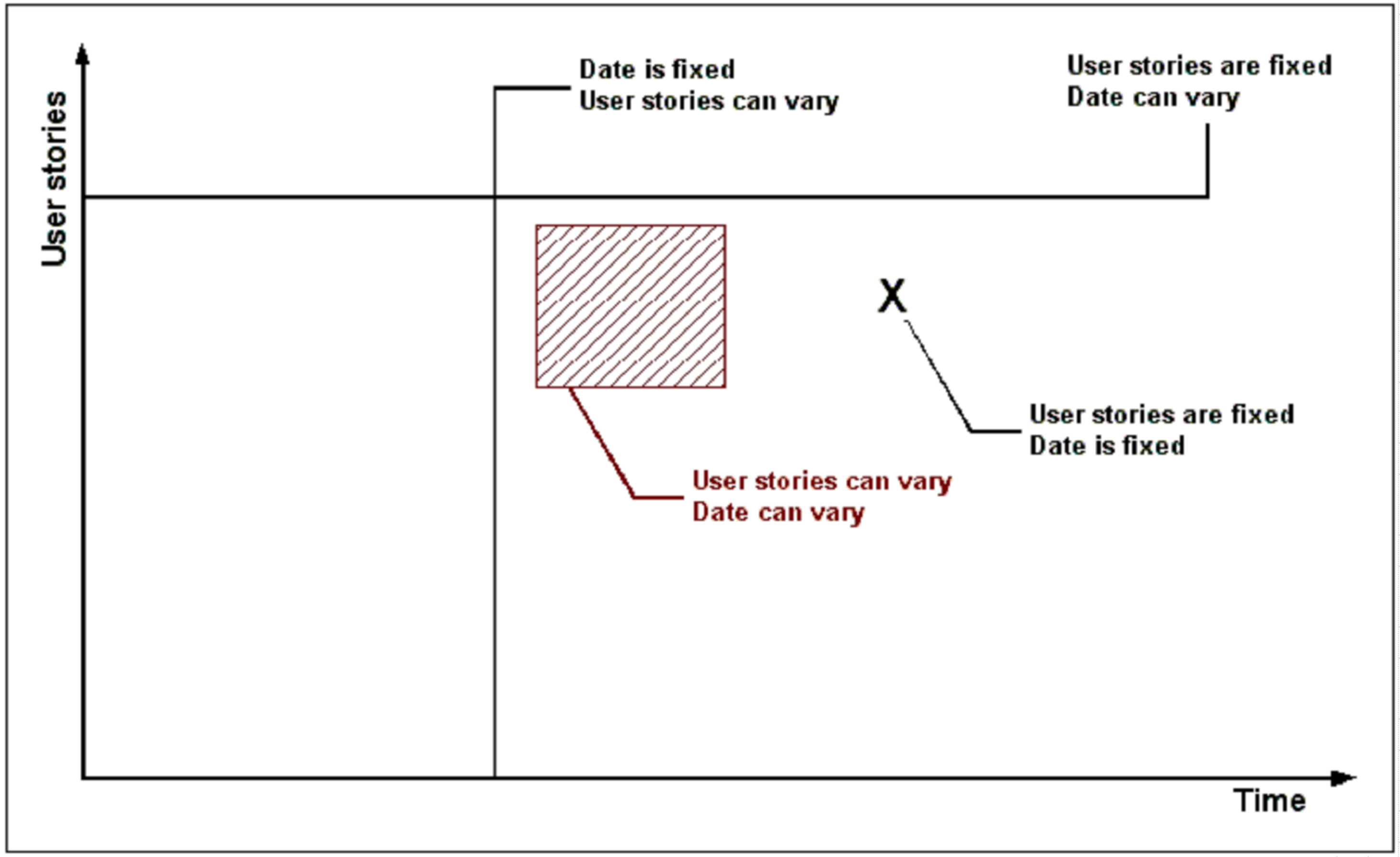
define a roadmap to guide development

Planning far away

Release planning defines a high-level development roadmap that shows the main areas of focus for the next handful of releases

- Aim to balance the competing goals of more features versus early release
- Expect the roadmap to change as we learn more about the product, its market and our ability to develop the product
- Involve the customer and the development team

Functionality and time



Drive projects by business value

Drive the project by business value rather than by fixed dates / features

- For each release, establish a range of dates and combinations of features
- This approach buffers each dimension
  - » User stories are buffered using a Feature buffer
  - » Time is buffered using a Schedule buffer
- For example, starting with a date range in mind you might be able to say:

  "After 6 to 8 iterations we should have these high business-value features and maybe some of those additional medium and low business-value features"

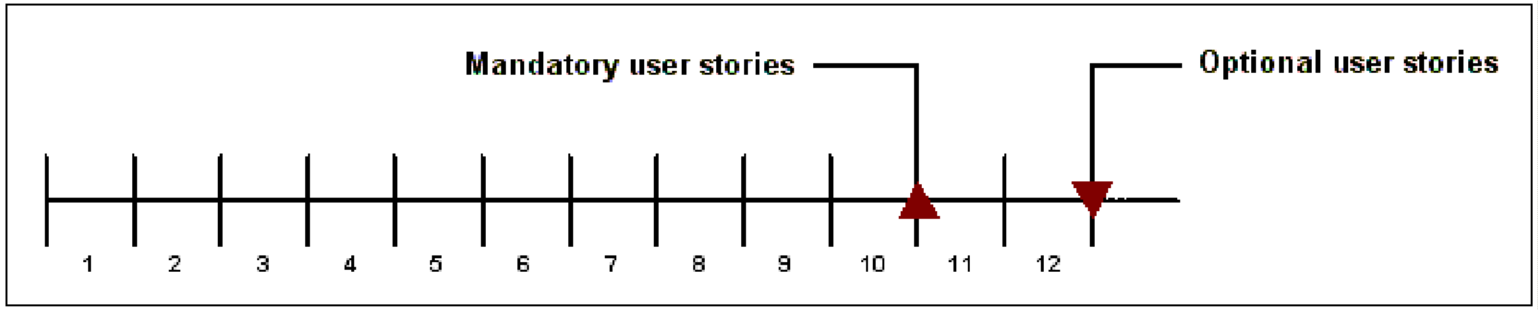
  - » Feature buffer = some of those additional medium and low business-value features
  - » Schedule buffer = 6 to 8 iterations

Creating a feature buffer

- Given the highest business-value user stories the customer identifies those he considers mandatory
- The customer then identifies lower business-value user stories which he considers to be optional
- Plan the release for the complete set of user stories
- If time permits some or all of the optional user stories will be developed, but only after the mandatory user stories have been completed

Example: Using a feature buffer

Mandatory user stories = 100 points and the optional user stories = 20 points
Velocity = 10 story points, therefore we should release after 12 iterations



- If we're on schedule:
  - » The mandatory user stories will be delivered in 10 iterations
  - » The customer can then decide whether to deliver the optional user stories in another 2 iterations

- If we're behind schedule:
  - » We have another 2 iterations to deliver the mandatory user stories

Release plan

! **A release plan describes features to be delivered and not engineering tasks to be performed**

?

Release plan

Comprises user stories ordered by business value and estimated in story points. It provides the development team with guidance about their schedule and tells them where they expect to be at the end of each iteration. It has general uncertainties.

- Release plans should not:
  - » Indicate which developers are assigned to which user stories
  - » Indicate the sequence in which work within an iteration will be performed
  - » Disaggregate user stories into engineering tasks
  - » Creating a release plan with this level of detail would be misleading

Release planning steps

1. Customer writes the user stories
2. Customer selects a target date ranges for the next few releases
3. Developers estimate the user stories as a team using story points
4. Developers calculate the number of iterations in the releases
5. Developers derive their velocity
6. Customer prioritises the user stories in descending order of business value
7. Assign stories to the next few releases in priority order and based on the team’s velocity
8. Assign stories to the iterations in priority order based on the team’s velocity

Step 1: Write user stories

- The customer writes the user stories and the acceptance tests
- The development team should be prepared to collaborate with the customer
- A tester can assist with the acceptance tests especially if they are automated
- In the worst case, the developers write the user stories and acceptance tests from a business / user perspective

Step 2: Select date range for release

- What is the external force driving the need for release?
  - Significant enhancement or new functionality
  - Previously announced public milestone
  - Contractual obligation
- Is the date fixed?
  - The customer should communicate date constraints to the team
  - Use a feature buffer
- In all other situations, think in terms of a date range or window

Step 3: Estimate user stories

- Estimates represent the cost of a project's requirements
- Prior to selecting the contents of release, it's important to:
  - » Estimate every user story
  - » Quantify the technical risk of every user story
- Estimate the user stories as a team
  - » The team doesn't yet know who will work on a story, therefore ownership cannot be more precisely assigned than to the team collectively
- Estimate the user stories using yesterday's weather
  - » If possible, base a story's estimate on a similar story already done
- Valuable discussion to have around a user story estimate is "Why is this so expensive?"
  - » This often uncovers unspoken assumptions
  - » Can lead to a simpler (to implement) user story

Step 4: Calculate number of iterations

- The iteration duration is fixed
- The release should be an exact multiple of iterations
  - » Half- or quarter-iterations will affect the velocity going forward so don't use them

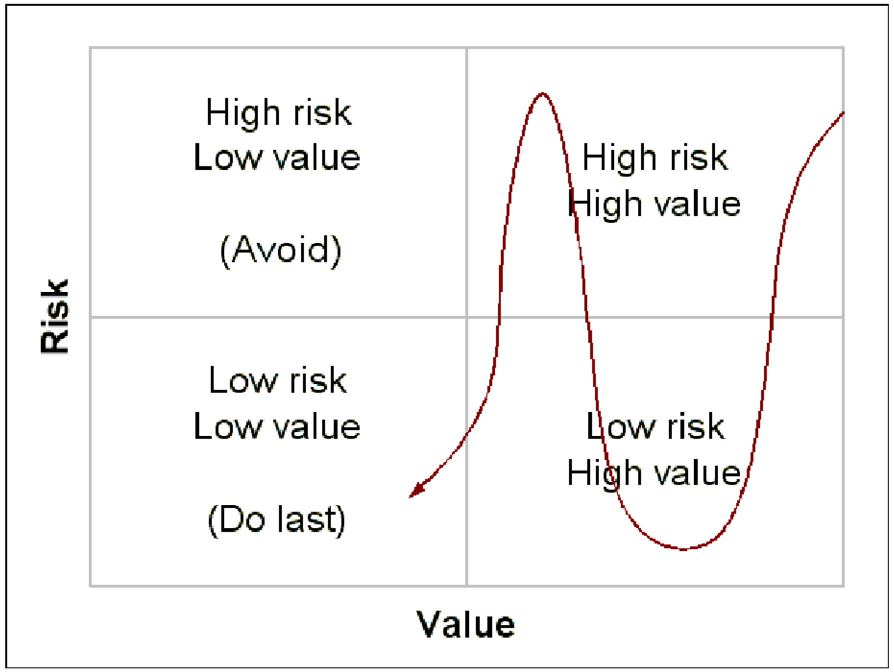
Step 5: Derive velocity

- Use historical values when available
  - » Stability – same team, same technology, same working environment – is vital for a previous velocity to be meaningful
- Or run an iteration and use the observed velocity to plan
  - » In an iteration, you'll deliver features with business value and establish a velocity with which to plan with confidence, rather than have produced a Gantt chart with estimates based on analysis
- Or make a forecast / educated guess
  - » Disaggregate user stories into engineering tasks and estimate in ideal hours
  - » Based on the distribution of ideal hours among stories assign story point estimates from Fibonacci scale
  - » Make a conservative assumption about developer productivities
  - » Count up the story points to give you a forecast velocity

Step 6: Prioritise user stories

Give neither risk nor business value total supremacy when prioritising

- Use risk-biased value order, i.e. within approximately equivalent value-bands develop riskiest user stories first



Step 7: Assign user stories to release

- Velocity x Number of iterations = Total story points in release
- Example:
  - » Actual or estimate of velocity per iteration = 30 story points
  - » Number of iterations in release = 8 iterations
  - » 30 x 8 = 240 story points in the release

Step 8: Assign user stories to iterations

- Assign user stories to iterations in descending order of business value
- There may be a compelling reason (e.g. a dependency) to develop that low-priority user story with this high-priority user story
- Pebbles and sand: Spare capacity in the iteration can be filled with small user stories

Iterations					
1	**2**	**3**	**4**	**5**	**6**
Story 1	Story 5	Story 6	Story 7	Story 12	Story 14
Story 2	Story 8	Story 9	Story 11	Story 13	Story 16
Story 3		Story 10	Story 15		
Story 4			Story 17		

Example: Release planning [1/8]

Company sells PDAs and marketing has determined that customers want basic chequebook management software in next release

Step 1: Customer has written the user stories

User story
A user can enter cheques into her account
A user can enter deposits into her account
A user can enter electronic funds transfers into her account
A user can search for a cheque by payee, amount, cheque #, date
A user may have multiple accounts
A user can manually balance her account
A user can automatically balance her account
A user can print cheques
A user can delete cheques
Transactions can be manually downloaded from bank sites
A user can mark a cheque as void
Various reports on spending history are available

Example: Release planning [2/8]

Step 2: Select a target date range for the release

- For the device to be included in a certain magazine review, it needs to be done in 3-4 months
- We cannot risk being late
- Target a minimum set of features within 3 months
- If progressing well, consider extra feature(s) for outer 4-month target

Example: Release planning [3/8]

Step 3: Team estimates the user stories

- Developers assign story point estimates to each user story

User story	Story points
A user can enter cheques into her account	7
A user can enter deposits into her account	3
A user can enter electronic funds transfers into her account	4
A user can search for a cheque by payee, amount, cheque #, date	10
A user may have multiple accounts	4
A user can manually balance her account	7
A user can automatically balance her account	7
A user can print cheques	5
A user can delete cheques	1
Transactions can be manually downloaded from bank sites	10
A user can mark a cheque as void	1
Various reports on spending history are available	10
Total story points	**69**

Example: Release planning [4/8]

Step 4: Calculate the number of iterations

- Iteration duration is fixed at 2-weeks
- Provide enough checkpoints on progress and opportunities to steer project by reacting to new information/feedback, and adjusting priorities accordingly
- 3 months = 6 iterations
- 4 months = 8 iterations

Example: Release planning [5/8]

Step 5: Derive the team's velocity

- Same team composition, same technologies, same working environment, same iteration duration
- Previous actual velocity (from previous project) = 7 story points

Example: Release planning [6/8]

Step 6: Prioritise the user stories

- Customer prioritises the user stories

User story	Story points
A user can enter cheques into her account	7
A user can enter electronic funds transfers into her account	4
A user can enter deposits into her account	3
A user can search for a cheque by payee, amount, cheque #, date	10
A user may have multiple accounts	4
A user can manually balance her account	7
A user can automatically balance her account	7
A user can print cheques	5
Transactions can be manually downloaded from bank sites	10
A user can delete cheques	1
A user can mark a cheque as void	1
Various reports on spending history are available	10
Total story points	**69**

Example: Release planning [7/8]

Step 7: Assign user stories to the release

- Release in 3 months, i.e. after 6 2-week iterations. Velocity = 7 story points. Release can contain 7 x 6 = 42 story points

User story	Story points
A user can enter cheques into her account	7
A user can enter electronic funds transfers into her account	4
A user can enter deposits into her account	3
A user can search for a cheque by payee, amount, cheque #, date	10
A user may have multiple accounts	4
A user can manually balance her account	7
A user can automatically balance her account	7
Total story points	**42**

- Worst case:
  - » Release in 4 months with only 42 story points
  - » Implied velocity = 42 / 8 = 5 story points

Example: Release planning [8/8]

Step 8: Assign user stories to iterations

- Each iteration must be planned to include 7 story points

Iteration	User story	Story points
1	A user can enter cheques into her account	7
2	A user can enter electronic funds transfers into her account A user can enter deposits into her account	4 3
3	A user can search for a cheque by payee and amount	7
4	A user can search for a cheque by cheque # and date A user may have multiple accounts	3 4
5	A user can manually balance her account	7
6	A user can automatically balance her account	7

- Fourth user story was 10 story points and has been split
  - » A user can search for a cheque by payee and amount – 7 points
  - » A user can search for a cheque by cheque # and date – 3 points

Questions

1. What are the three ways of estimating a team's initial velocity?

2. Assuming 2-week iterations and a team of four developers, how many iterations will it take the team to complete a project with 27 story points if they have a velocity of 4?

Answers

1. You can use historical values, take a guess, or run an initial iteration and use the velocity of that iteration.

2. With a velocity of 4 and 27 story points in the project, it will take the team 7 iterations to finish.

Agenda

- ☑ An agile approach
- ☑ Estimating with story points
- ☑ Release planning
- ☐ Estimating with ideal time
- ☐ Iteration planning
- ☐ Tracking and metrics

Estimating with ideal time

a plan is only as good as the estimates it is based on

Ideal time: An everyday concept

How long is an American football game?

- 4 x 15-minute quarters = 60 minutes or ~ 3 elapsed hours
- Both are correct in their own context
- The difference is the difference between ideal time and elapsed time

Ideal programming time

?

Ideal programming time

Ideal programming time is an estimate of how long an engineering task would take if it's the only thing a developer works on, everything needed is available when the task is started, and there are no interruptions.

Typical everyday interruptions include:

- » Holidays
- » Sickness
- » Meetings, emails and phone calls
- » Demonstrations and code reviews

!

Estimate engineering tasks in ideal programming hours

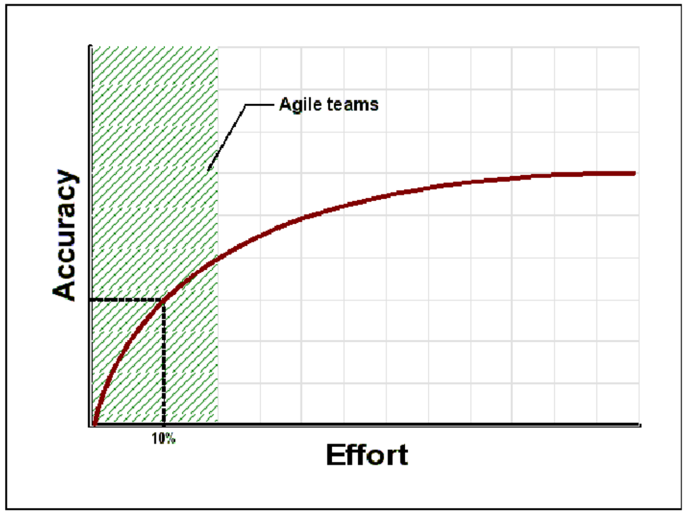
How much estimation effort? [1/2]

Question:	The more effort we put into something, the better the result. Right?
Answer:	Yes, but often we can expend just a fraction of that effort to get results that are 'good enough'

We can spend a little time thinking about an estimate and arrive at a number that is nearly as good as if we had spent a lot of time thinking about it. No matter how much effort is invested, the estimate is still a guess.

How much estimation effort? [2/2]

- Notice how little effort is required to move the accuracy up from the base line – about 10% effort gets 50% potential accuracy



- Inside green zone: We acknowledge inaccuracy cannot be eliminated from estimates but small efforts can often be rewarded with big gains

- Outside green zone: Effort includes extensive requirements, upfront analysis, detailed planning from the start. After this upfront effort, estimates still retain inaccuracy

From ideal time to elapsed time

Deal in ideal time for as long as possible and convert to elapsed time only when necessary

Calculating how many elapsed hours in an ideal hour:

- Using historical values
  - » Use velocity, e.g. in previous the iteration the team completed 100 ideal hours

    Iteration = 2 elapsed weeks

    Iteration = 80 elapsed hours per Developer x 5 developers

    Iteration = 400 elapsed hours

    1 ideal hour = 4 elapsed hours
- Without historical values
  - » Take a guess (it's only a starting estimate)
  - » Use the rule of thumb: 1 ideal hour = 2-3 elapsed hours

Problems with estimating in ideal time

- Some developers will make implicit conversions in their minds as they estimate:
  - A developer considers task and decides it will take 1 elapsed week to complete
  - Knowing that 5 elapsed days = 16 ideal hours, he estimates task at 16 ideal hours
  - Ideal time will later be converted to elapsed time
  - Estimate goes from elapsed to ideal to elapsed time

What is the result when you translate an English paragraph into French and then back into English?

Advantages of estimating ideal time

- Estimating in ideal time is simple
  - A developer need only consider the time to complete the engineering task
  - Estimating in elapsed time is flawed because a developer cannot anticipate all of the interruptions that will occur in the future
- An estimate in ideal time remains useful, e.g. a task = 40 ideal hours
  - It's fair at any time to ask how long it will take to get 40 ideal hours on the task
  - When first estimated the developer said it will take 80 elapsed hours
  - Now the developer is busier so the task will take 100 elapsed hours
- Ideal time is easy to understand
  - Everyone should understand that 8 hours each day are not spent on programming, testing or otherwise making progress on engineering tasks

Agenda

- ☑ An agile approach
- ☑ Estimating with story points
- ☑ Release planning
- ☑ Estimating with ideal time
- ☐ Iteration planning
- ☐ Tracking and metrics

Iteration planning

plan the next iteration in detail

Planning up close

Iteration planning looks no further forward than the current iteration

- Create a detailed, actionable plan for the iteration
- Fine-tune the plan based on new information and knowledge
- It’s an opportunity to reprioritise the release plan
- Involve the customer if reprioritisation necessary

Iteration plan

! **An iteration plan describes the engineering tasks that need to be completed to deliver the user stories**

?

Iteration plan

Comprises user stories disaggregated into engineering tasks and estimated in ideal programming hours. It provides information that the developers will track progress against.

- Iteration plans should <u>not:</u>
  - » Assign developers to engineering tasks at the start if the iteration

Iteration planning steps

1. Identify / confirm the user stories selected for the iteration
2. Disaggregate each user story into engineering tasks and estimate how long each task will take in ideal time
3. Commit to the work

Step 1. Identify the iteration contents

- Plan to complete the same number of story points as the last iteration
- The release planning identified the default user stories scheduled for iteration
  - » With the customer, quickly reconsider whether the release plan still identifies the highest business value user stories
- Confirm or adjust the selected user stories while satisfying the team’s velocity

Step 2. From user stories to tasks

- Developers disaggregate each user story into engineering tasks
- The team estimates the engineering tasks in ideal hours
- Example user story: “*A user can save her preferences*”

Engineering tasks:

» Code user interface
» Create database table(s) to store preferences
» Code to store and retrieve preferences in database
» Code to load preferences when a user logs in
» Refactor parts of application so that they use saved preferences
» Code for new FitNesse fixture (acceptance tests)
» Code FitNesse test page

What to include as tasks?

- Write tasks for the following:
  - » Programming tasks
  - » System / Acceptance testing tasks (not unit testing – this is an implicit activity within test-driven development)
  - » Extraordinary events, e.g. 4-hour full team meeting
  - » Any new activity or an activity that varies significantly in effort
- Do not create tasks for the following:
  - » Planning activities, e.g. estimate new stories, if they are roughly constant across iterations
  - » Activities that were tasks but have now become routine
  - » Repetitive work that is part of daily / weekly routine, e.g. answering emails
- Be consistent

What is the right size for a task?

- Ideally between 8 and 16 ideal hours
- This size allows developers to report significant progress or completion on a daily basis

Difficult to disaggregate

- Some user stories are difficult to disaggregate into tasks

- Example: Need to make a small change to a legacy system
  - » We're not confident that all possible impacts have been identified
  - » We've identified some code sections that would be affected – 4 ideal hours
  - » We're unsure about other code sections – could be as much as 20 ideal hours
  - » We can't be sure without code analysis but don't want to interrupt planning

- Solution: Split user story into 2 smaller user stories:
  - » Let's determine what's affected – 2 ideal hours (could also be a spike)
  - » Let's make the changes – 10 ideal hours
  - » Let's take a guess and re-estimate when we know more and then disaggregate into tasks

Dealing with bugs

- Ideally all bugs should be resolved during the iteration
  - » Try to avoid carrying bugs forward to the next iteration

- If a bug is found with say, 1 elapsed hour remaining in the iteration
  - » Write a task, e.g. Fix bug 439, user cannot reset password
  - » Carry that task forward to next iteration so that it's not forgotten

Step 3. Commit to the work

- The development team review the quantity of work and assess whether it can be achieved within the iteration
- It's possible that one or more user stories is more work than originally thought. Either:
  - » Split the user story and re-estimate the new stories in story points separating the engineering tasks accordingly
  - » Descope the user story from the iteration
- Do not attempt to complete more work than is realistic for team

What is a realistic amount of work?

The more historical information available, the better the planning and predictability you can achieve

- We know what was planned for in the previous iteration
- We know what was achieved in the previous iteration (potentially more or less than the planned level)
- Calculate an estimation accuracy % (which should stabilize over time at the member level and especially at a team level)
- We know what is planned for this iteration
- Calculate a forecasted actual effort based on the estimation accuracy %

Volunteering for tasks

! **Resist the temptation during iteration planning to have developers volunteer for (or be assigned to) engineering tasks**

- A developer should only be signed-up to one task at a time
  - » Volunteer for the next task when the current task is completed
  - » Assess the team's estimate to ensure it is commensurate with the developer's own abilities
  - » This approach avoids problems associated with multitasking
- Foster a "we're in this together" attitude
  - » Developers pick up the slack for each other knowing the favour will be returned
  - » This avoids the problem where one developer is blocked by another developer

Example: Iteration planning [1/5]

Iteration 1:

- 10 story points
- During iteration planning, Iteration 1 resulted in a total task estimate of 80 ideal hours
- At the conclusion of Iteration 1, actual effort was 100 ideal hours
- All user stories were accepted, Velocity = 10 story points

Example: Iteration planning [2/5]

Iteration 2:

- Plan 10 story points based on prior velocity
- Iteration planning results in 90 ideal hours of estimated tasks (therefore there may actually be some task risk in the iteration)
- Iteration results in 108 ideal hours of actual effort
- Likely 1 story did not get completed, let's say 1 story point
  - » Perhaps split from a larger story
- New Velocity = 9 story points

Example: Iteration planning [3/5]

Iteration:

- Plan 9 story points based on prior velocity
- Iteration planning results in 85 ideal hours of estimated tasks
- Iteration results in 100 ideal hours of actual effort
- All user stories accepted, Velocity = 9 story points

Example: Iteration planning [4/5]

Over 10 iterations:

- Team stabilizes on a velocity of 9 story points
- On average:
  - » Task estimate of 85 ideal hours
  - » Actual effort of 103 ideal hours
  - » Estimation accuracy of ~83%
- Take velocity, task estimate and estimation accuracy into account when planning Iteration 11
- We Know that with an estimation accuracy of ~ 83%, task actual effort will likely be in the 103 ideal hours range

Example: Iteration planning [5/5]

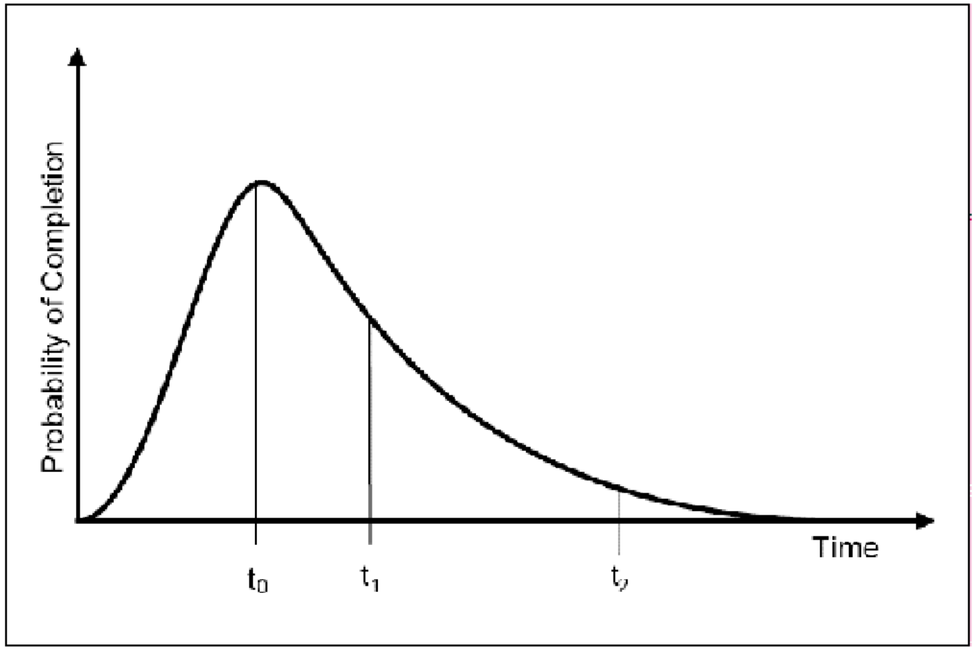
Iteration 11:

- When planning iteration 11, say the task estimates add up to 200 ideal hours
- This means that we really estimated badly at the story level and need to decide what to do
- We could:
  - » Re-estimate the stories based on better information and re-plan the iterations
  - » Remove a story or two from the iteration (therefore reducing the planned velocity)
  - » Proceed with the iteration as planned and very likely remove a user story or two during the iteration

Uncertainty in estimates [1/3]

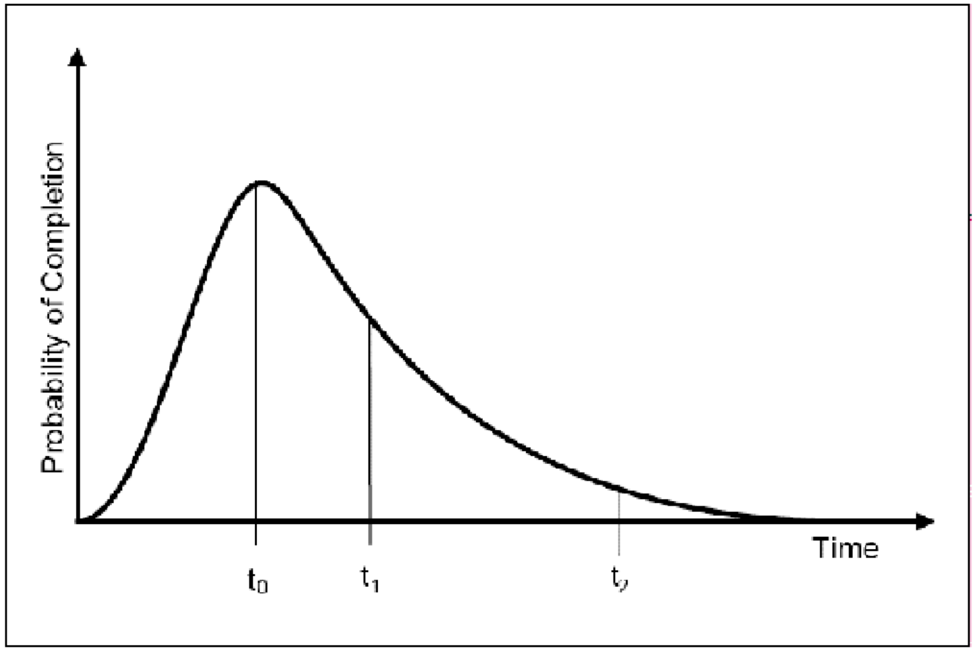
- Imagine a user story has been estimated at 3 story points
- We would not be surprised if it was done in what would've been better estimated at 1 or 5 story points

Uncertainty in estimates [2/3]



- t_0 .. Most likely single duration for the user story
- t_1 .. Duration at which half the time the user story is finished ahead of schedule and half the time the user story is finished behind schedule
- t_2 .. Duration to complete the user story even if many things go wrong

Uncertainty in estimates [3/3]



- Many developers will give t_0 or t_1 as their estimate
- Since <50% of curve to left of t_0 then >50% chance work will not be completed by then

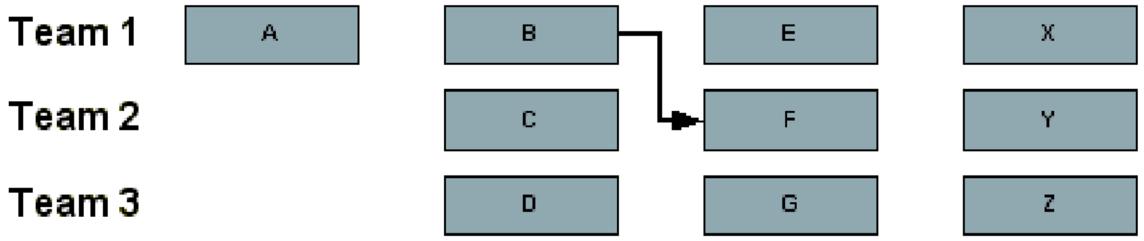
Key points to remember

- Use all pieces of information at your disposal to plan and manage execution
- Break the mental relationship between high-level estimates and elapsed time
- At the release planning level we are looking for consistency to maintain the relative magnitudes
- At the iteration planning level it’s possible that 2 3-point stories may be estimated at 16 and 26 ideal hours. These are stories at opposite ends of the distribution scale
- Over time, consistency between the estimates will very likely be achieved and should provide improved planning and project predictability
- Agile teams have moved away from individual metrics to focus on team-based metrics for planning and predicting, i.e. velocity, total task estimate for the iteration

Inter-team dependencies

Wherever possible teams should develop in parallel using interface-driven design with test-driven development techniques

- But that's not always practical so use a consecutive dependency carefully
- If Team 1 under-delivers in the second iteration then Team 2 is affected and the ripple effect passes down schedule

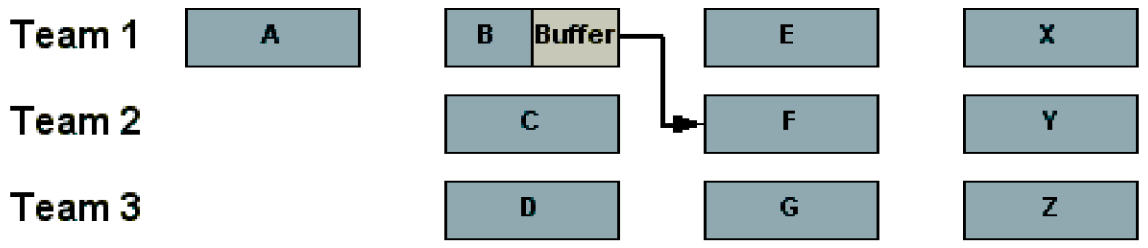


- It's important to protect the deadline and maintain a synchronised iteration schedule across all teams

Using a feeding buffer [1/2]

A feeding buffer is a lower-planned velocity

- Example: Each team has a velocity = 20 story points
  - » Establish a feeding buffer by assigning only 10 story points to the second iteration
  - » Missing 10 story points serve as the feeding buffer
  - » Protect start of F by giving Team 1 a small buffer to ensure the completion of B

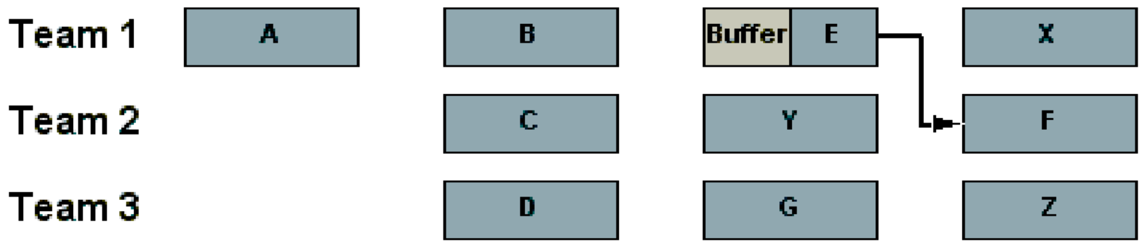


- This scenario shows a feeding buffer can be established by shifting work from B presumably into E.

Using a feeding buffer [2/2]

- What if Team 2 is dependent upon functionality that will take all of the first and second iterations to deliver?

- Then the feeding buffer becomes part of the following iteration
  - » Any work not delivered in the first and seconds iterations is developed at the start of the third iteration
  - » Notice how Team 2 has moved functionality Y forward



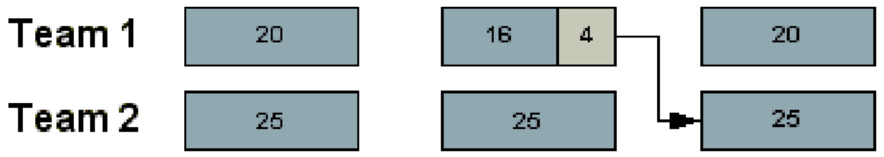
What gets buffered?

Use feeding buffers only when absolutely necessary because they can prolong a schedule

- Add a feeding buffer for critical dependencies between iterations and teams where development has to occur consecutively
  - If a team is unable to do its planned work without the deliverables of another team
- Do not add a feeding buffer:
  - When a team can easily swap in other valuable work
  - If the second team will be able to progress with a partial deliverable from the first team – Interface-driven design, mock objects and tests
  - When dependencies betweens tasks are done during a single iteration by a single team

Sizing a feeding buffer

- Most inter-team dependencies are based on a handful of user stories
- Rule of thumb:
  - Feeding buffer = 50% of the size of the user stories creating the dependency
- Example: Team 2 is dependent on Team 1 for 6 stories with an estimate of 24 story points
  - The feeding buffer = 24 / 2 = 12 story points
  - Team 2 plans for Team 1 to deliver in 24 + 12 = 36 story points
  - Team 1 has velocity of 20 story points



Agenda

- ☑ An agile approach
- ☑ Estimating with story points
- ☑ Release planning
- ☑ Estimating with ideal time
- ☑ Iteration planning
- ☐ Tracking and metrics

Tracking and metrics

keeping score provides regular and rapid feedback

Tracking analogy

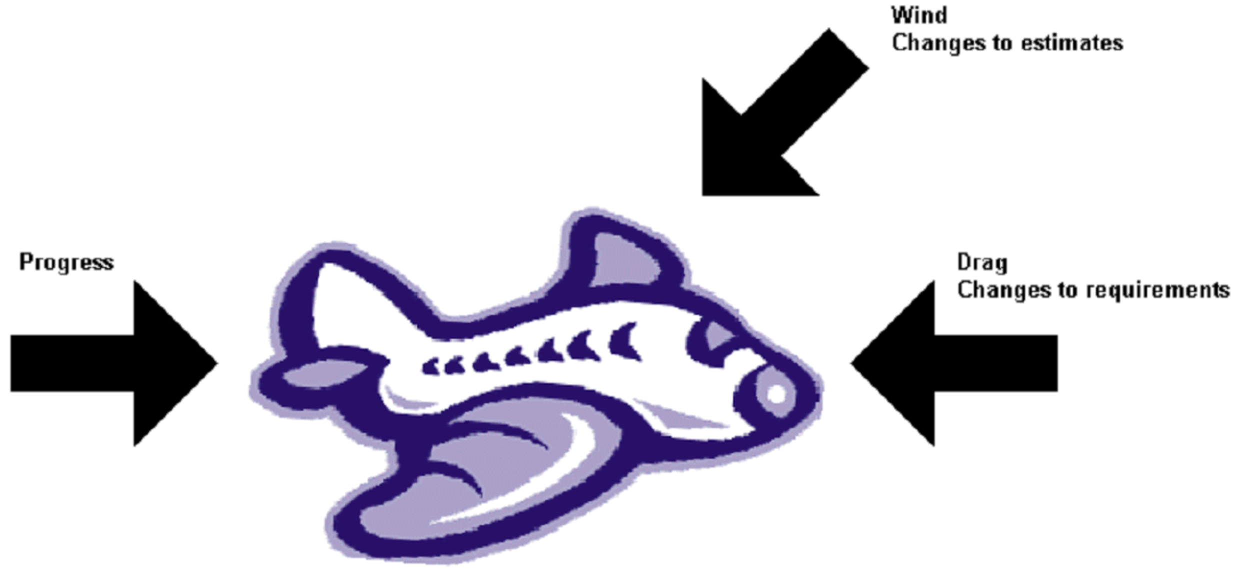
- During development we want to constantly measure:
  - » How much more work we have to do
  - » How fast we are doing work

- Using an aeroplane analogy the measurements become:
  - » Relative distance – how much more work we have to do
  - » Velocity – how fast we are doing work

Tracking a release [1/2]

- At the start of a release, we establish a plan that states something like “Over the next 4 months and 8 2-week iterations we’ll complete approx. 240 story points”
- At the end of each iteration we’d like to assess where we are relative to that goal
- However, there are many forces at work which need to be taken into account
  - » The amount of progress made by team
  - » Changes in requirements
  - » Developers may have learned things that make them want to revise estimates for future work in the release
- Continuing the analogy, these forces are similar to those experienced by an aeroplane

Tracking a release [2/2]



- Drag (changes in requirements) will cause the aeroplane to travel less distance than would be inferred from its speedometer
- The aeroplane's compass points due west but the wind (changes in estimates) will cause it to drift south
- Without course corrections, the aeroplane will take longer to get to its original destination

Continuing the tracking analogy

Adding legs to a flight You can add all the legs you want, but unless you can increase the plane's speed you are going to arrive at your destination later	**Adding features to a release**
Increase the speed of the plane by increasing engine size But it will cost installation time	**Add more team members**
You can increase speed by pushing the engines harder Which decreases their life span	**Working overtime**
Aircraft maintenance Neglecting ongoing maintenance can be expensive with a plane (project) grounded when it's supposed to be in the air or a plane crashing to the ground	**Unit tests and constant refactoring**
Boarding a plane	**Release start-up overhead**

Tracking an iteration

- Release plan states something like:

  "Over the next 4 months and 8 2-week iterations we'll complete approximately 240 story points of work"

- At end of each iteration we'd like to assess where we are relative to that goal

Velocity revisited

! **Velocity is a team's primary measure of progress**

?

Velocity
The number of story points completed per iteration.

Calculating velocity [1/3]

We're good at knowing when something hasn't been started and we're fairly good at knowing when it's done

! **Only count story points toward velocity for the user stories that are complete at the end of the iteration**

? **Complete**

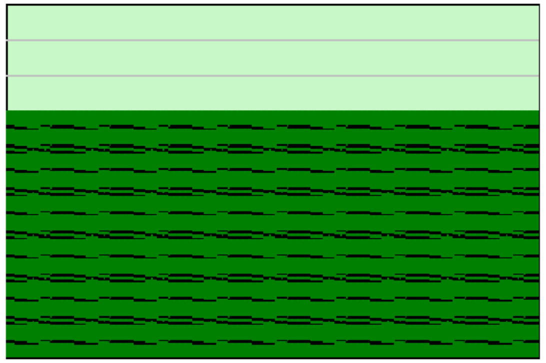
Code that is well written and well factored, checked-in, clean, compiles with coding standards, and passes all tests

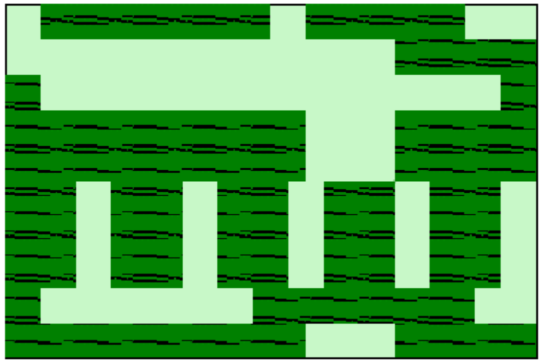
Calculating velocity [2/3]

- The problem with counting incomplete user stories is it's difficult to know how much progress has really been made

- Which is further along?
  - » A story that has been developed but has had not tests run against it, or
  - » A story that has been partially developed and partially tested

- Mowing a lawn analogy:
  - » On the left, it's easy to look at the lawn and say you've mowed roughly a third of it
  - » On the right, it's more difficult to say how much has been mowed without effort





Calculating velocity [3/3]

Minimise the number of times incomplete stories are carried forward

- If incomplete stories are present at the end of an iteration, the customer can prioritise them into the next iteration

Retain the original story point estimate and allow team to earn the full amount in next iteration

- This may cause the velocity to fluctuate more between iterations, so take a rolling average

Velocity

Average number of story points completed over recent iterations

Why velocity?

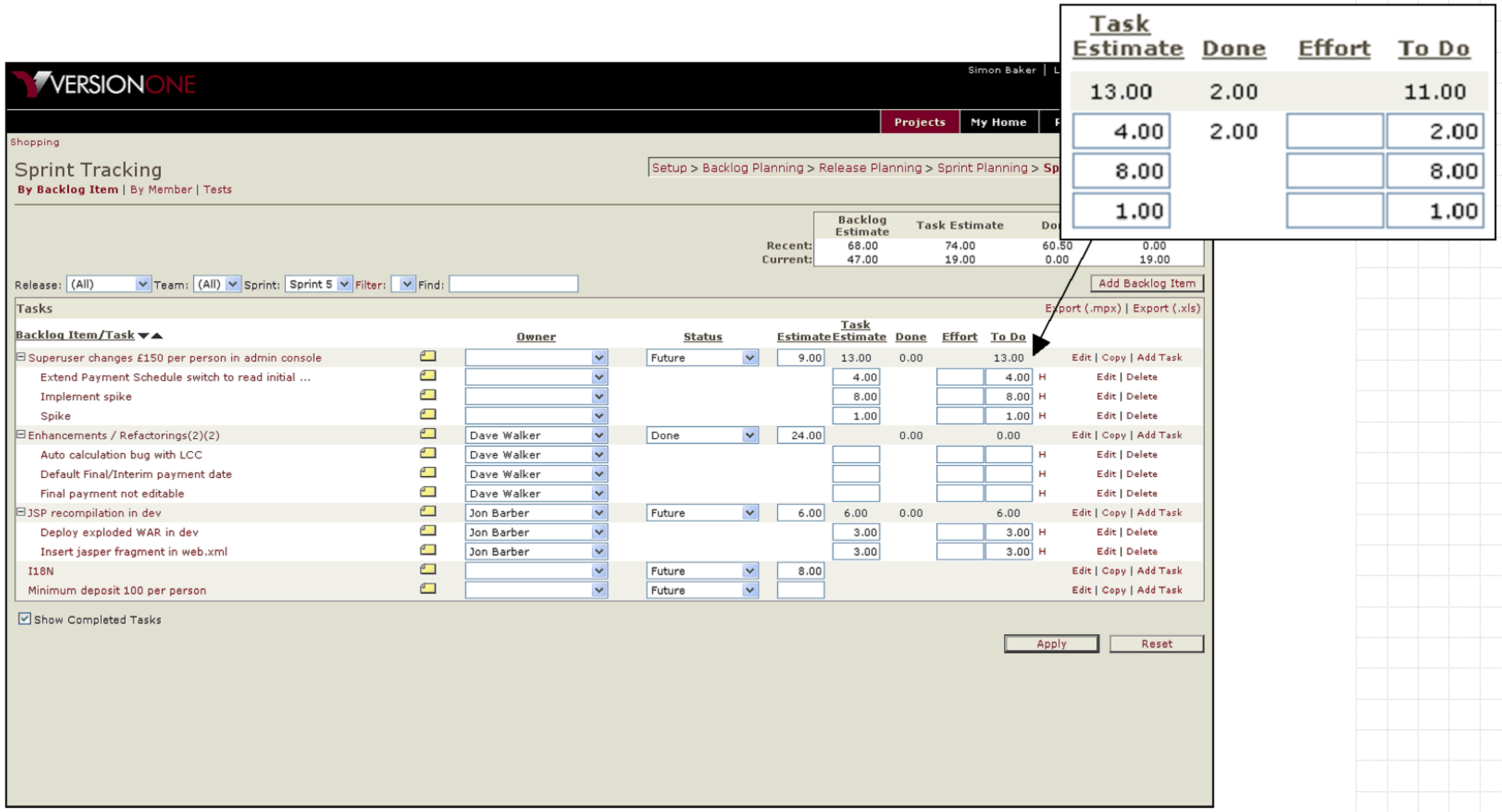
Question:	Why is velocity is better than straight forward estimating?
Answer:	With absolute estimating, the only way to improve is to try and get better at being more accurate. But as software development is unpredictable by nature, this is difficult.

With relative estimating, it doesn't matter how bad your estimates are, as long as you can be consistently bad. Velocity gives you a tool for adjusting the measure of how bad you are.

This means that you don't have to worry about trying to guess how much time you'll be spending doing non-programming things like meetings, etc.

Recording progress

- Use V1 to log your effort (which gets recorded as Done) and constantly assess and adjust the amount left To Do



Sprint Tracking

By Backlog Item | By Member | Tests

Setup > Backlog Planning > Release Planning > Sprint Planning > Sp

	Backlog Estimate	Task Estimate	Do	
Recent:	68.00	74.00	60.50	0.00
Current:	47.00	19.00	0.00	19.00

Release: (All) Team: (All) Sprint: Sprint 5 Filter: Find:

Add Backlog Item

Tasks

Export (.mpx) | Export (.xls)

Backlog Item/Task	Owner	Status	Estimate	Task Estimate	Done	Effort	To Do		
⊟ Superuser changes £150 per person in admin console		Future	9.00	13.00	0.00		13.00		Edit | Copy | Add Task
Extend Payment Schedule switch to read initial ...				4.00			4.00	H	Edit | Delete
Implement spike				8.00			8.00	H	Edit | Delete
Spike				1.00			1.00	H	Edit | Delete
⊟ Enhancements / Refactorings(2)(2)	Dave Walker	Done	24.00		0.00		0.00		Edit | Copy | Add Task
Auto calculation bug with LCC	Dave Walker							H	Edit | Delete
Default Final/Interim payment date	Dave Walker							H	Edit | Delete
Final payment not editable	Dave Walker							H	Edit | Delete
⊟ JSP recompilation in dev	Jon Barber	Future	6.00	6.00	0.00		6.00		Edit | Copy | Add Task
Deploy exploded WAR in dev	Jon Barber			3.00			3.00	H	Edit | Delete
Insert jasper fragment in web.xml	Jon Barber			3.00			3.00	H	Edit | Delete
I18N		Future	8.00						Edit | Copy | Add Task
Minimum deposit 100 per person		Future							Edit | Copy | Add Task

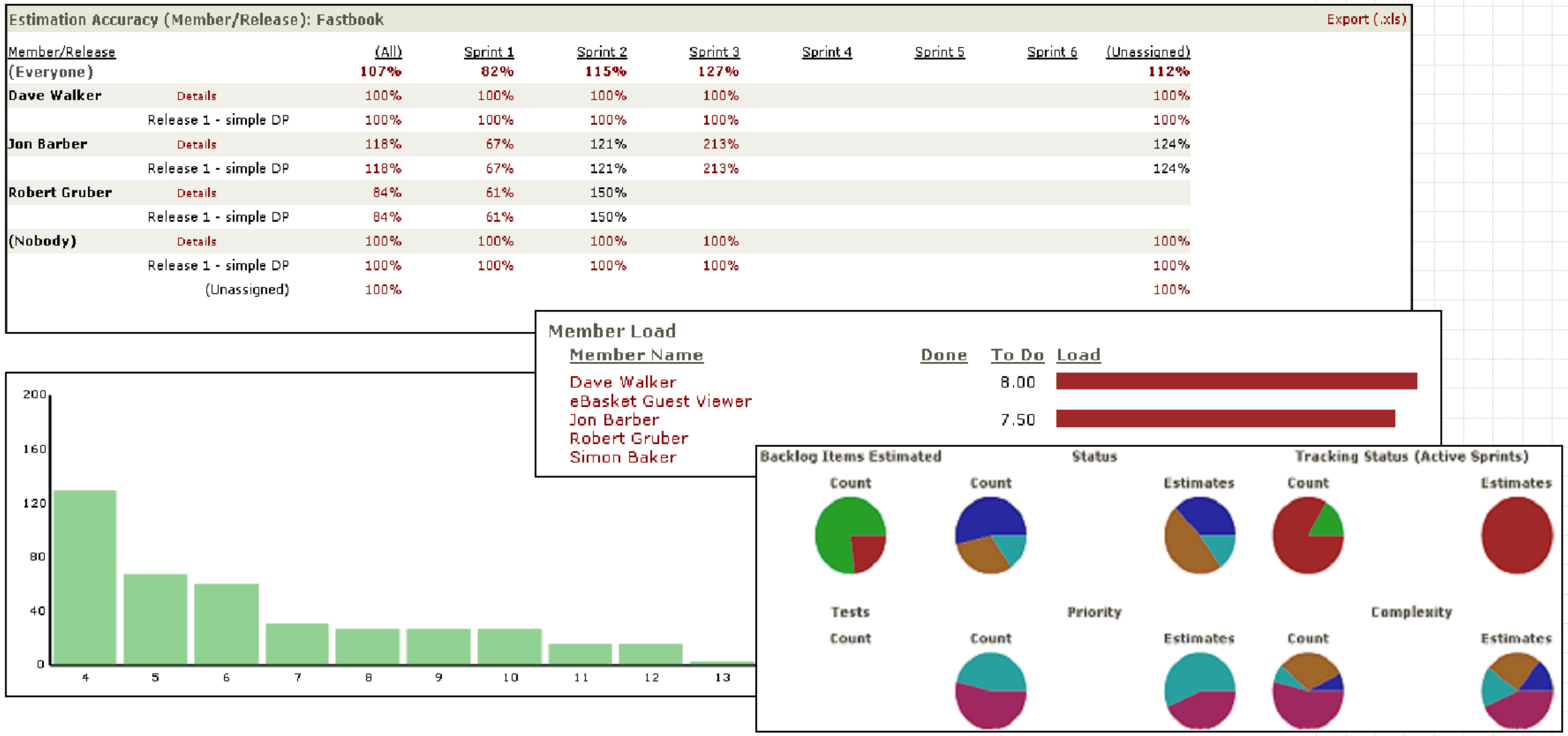
☑ Show Completed Tasks

Apply Reset

Task Estimate	Done	Effort	To Do
13.00	2.00		11.00
4.00	2.00		2.00
8.00			8.00
1.00			1.00

Tracking progress

- V1 is like the monitor in an aeroplane which constantly displays speed, estimated arrival, position relative to destination



Estimation Accuracy (Member/Release): Fastbook — Export (.xls)

Member/Release		(All)	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	(Unassigned)
(Everyone)		107%	82%	115%	127%				112%
Dave Walker	Details	100%	100%	100%	100%				100%
	Release 1 - simple DP	100%	100%	100%	100%				100%
Jon Barber	Details	118%	67%	121%	213%				124%
	Release 1 - simple DP	118%	67%	121%	213%				124%
Robert Gruber	Details	84%	61%	150%					
	Release 1 - simple DP	84%	61%	150%					
(Nobody)	Details	100%	100%	100%	100%				100%
	Release 1 - simple DP	100%	100%	100%	100%				100%
	(Unassigned)	100%							100%

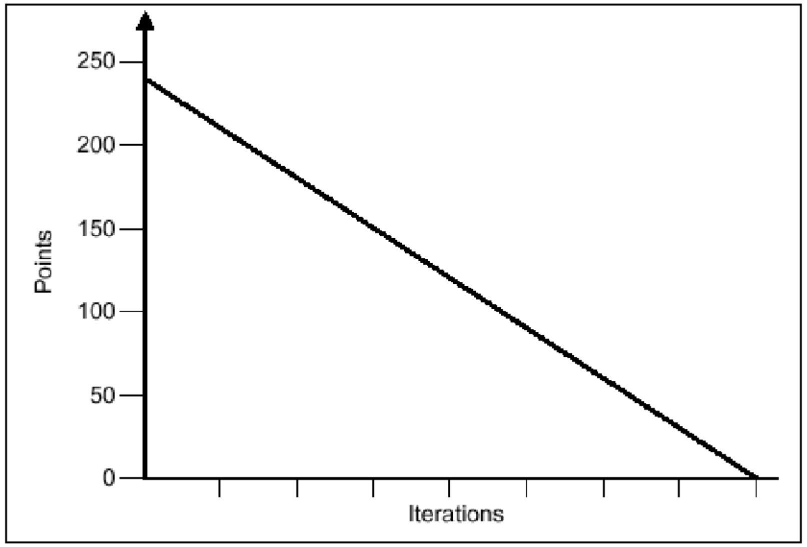
Member Load

Member Name	Done	To Do	Load
Dave Walker		8.00	
eBasket Guest Viewer			
Jon Barber		7.50	
Robert Gruber			
Simon Baker			

Release burndown chart [1/2]

The release burndown chart shows the amount of user stories remaining in a project

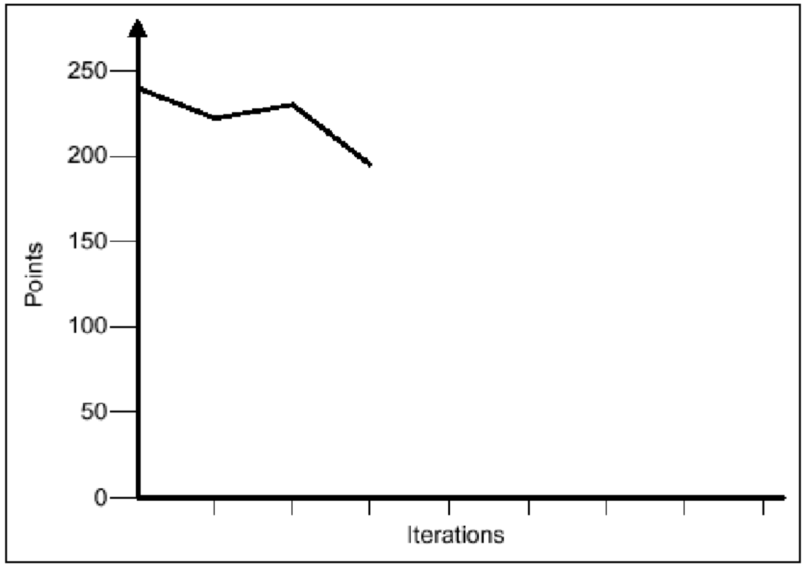
- It is a powerful visual indicator of how quickly a team is moving toward its goal



- Figure shows an idealised burndown for a project with 240 story points to be delivered over 8 iterations

Release burndown chart [2/2]

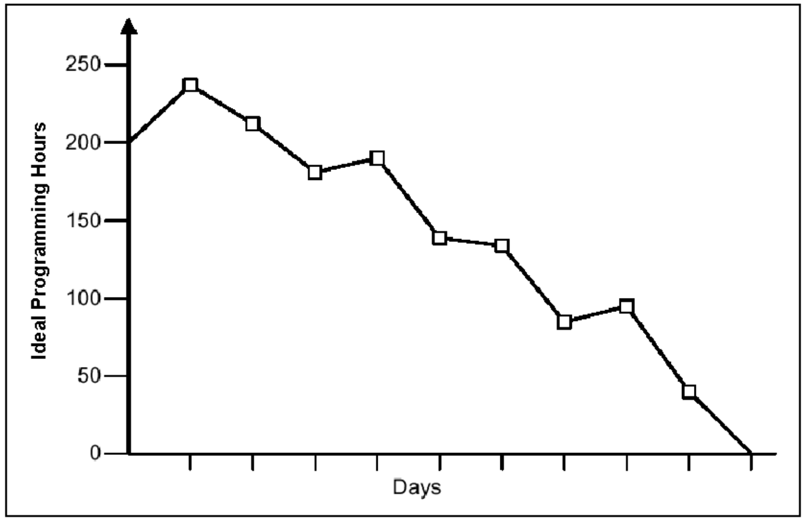
- It's unlikely that a team with a velocity of 30 will maintain it over iterations



- Iteration 1: The team completed around the right amount of work
- Iteration 2: There's more work than when the team started the iteration
  - » Burn up – work was added to the release plan
  - » The net result is that the project is going backwards and away from its goal

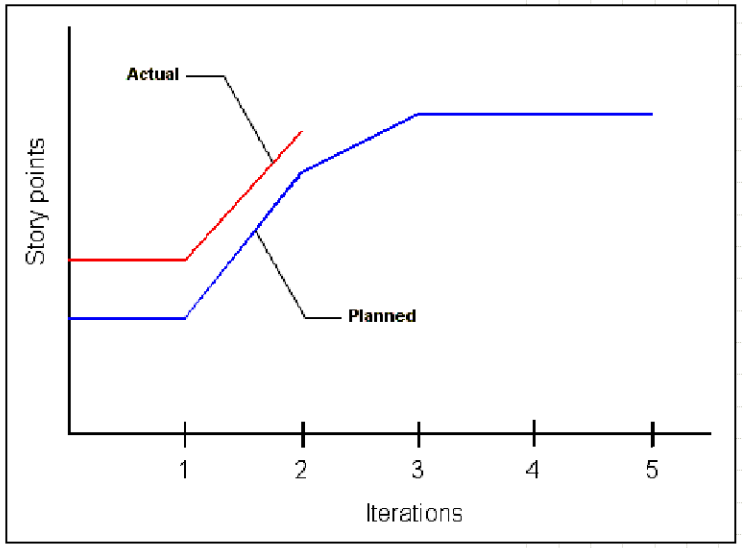
Iteration burndown chart

The iteration burndown chart shows the amount of ideal programming hours remaining by day



Questions [1/3]

1. Define velocity.
2. A story estimated at 1 story point actually took 2 days to complete. How much does it contribute to velocity when calculated at the end of the iteration?
3. What conclusions should you draw from the figure below?
   Does the project look like it will finish ahead, behind or on schedule?



Questions [2/3]

4. What is the velocity of the team that finished the iteration shown below?

Story	Story points	Status
1	4	Finished
2	3	Finished
3	5	Finished
4	3	Half finished
5	2	Finished
6	4	Not started
7	2	Finished

5. What circumstances would cause an iteration burndown chart to burn-up?

6. What are the missing values in the table below?

Questions [3/3]

6. What are the missing values in the table below?

	Iteration 1	Iteration 2	Iteration 3
Story points at start of iteration	100	?	?
Completed during iteration	35	40	36
Changed estimates	5	-5	0
Story points from new stories	6	3	?
Story points at end of iteration	**76**	?	**0**

Answers [1/2]

1. Velocity is the average number of story points completed over recent iterations.

2. It contribute 1 story point to the velocity.

3. This team started out a little better than anticipated in the first iteration. They expected velocity to improve in the second and third iterations and then stabilise. After two iterations they have already achieved the velocity they expected after three iterations. At this point they are ahead of schedule but you should be reluctant to draw too many firm conclusions after only two iterations.

4. The velocity is 16. Partially completed user stories do not contribute to the velocity.

Answers [2/2]

5. An iteration burndown chart would burn-up if new work is added faster than known work being completed, or if the team decides that a significant amount of future work has been underestimated.

6. The completed table looks like:

	Iteration 1	Iteration 2	Iteration 3
Story points at start of iteration	100	76	34
Completed during iteration	35	40	36
Changed estimates	5	-5	0
Story points from new stories	6	3	2
Story points at end of iteration	**76**	**34**	**0**

Agenda

- ☑ An agile approach
- ☑ Estimating with story points
- ☑ Release planning
- ☑ Estimating with ideal time
- ☑ Iteration planning
- ☑ Tracking and metrics

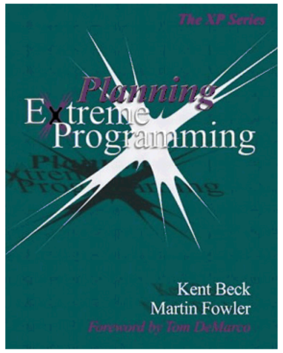
References

- *Agile Estimating and Planning*

  Cohn M (Not yet published), Addison Wesley

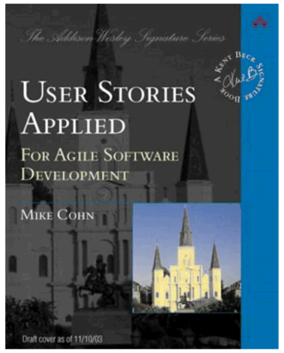
- *Planning Extreme Programming*

  Beck K, Fowler M (2000) , Addison Wesley



- *User Stories Applied For Agile Software Development*

  Cohn M (2004), Addison Wesley


The Addison-Wesley Signature Series
A Kent Beck Signature Book
User Stories Applied
For Agile Software Development
Mike Cohn
Draft cover as of 11/10/03