CSC 436, Fall 2017

Agile Processes

Ravi Sethi

THE UNIVERSITY OF ARIZONA

Agile Processes

Agile Manifesto

Feb 2001: self-described "independent thinkers" signed …

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

"That is, while there is value in the items on the right, we value the items on the left more."

Agile Working Definition

THE UNIVERSITY OF ARIZONA
Computer Science

- A process is *agile* if the values of the Agile Manifesto apply
    - Individuals and interactions over processes and tools
    - Working software over comprehensive documentation
    - Customer collaboration over contract negotiation
    - Responding to change over following a plan

Agile Processes/Methods emphasize

Based on the Agile Manifesto and accompanying Principles

- Satisfying customers through collaboration
- Delivering working software frequently (in weeks, not months)
- Accommodating changes during development
- Valuing simplicity and technical excellence

Principles behind the Agile Manifesto

Principles included here for completeness

– Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
– Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
– Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
– Business people and developers must work together daily throughout the project.
– Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
– The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
– Working software is the primary measure of progress.
– Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
– Continuous attention to technical excellence and good design enhances agility.
– Simplicity--the art of maximizing the amount of work not done--is essential.
– The best architectures, requirements, and designs emerge from self-organizing teams.
– At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://www.agilemanifesto.org/principles.html

Unix Design Philosophy: Is it Agile?

Summer 1978: Excerpts from a foreword to papers on Unix

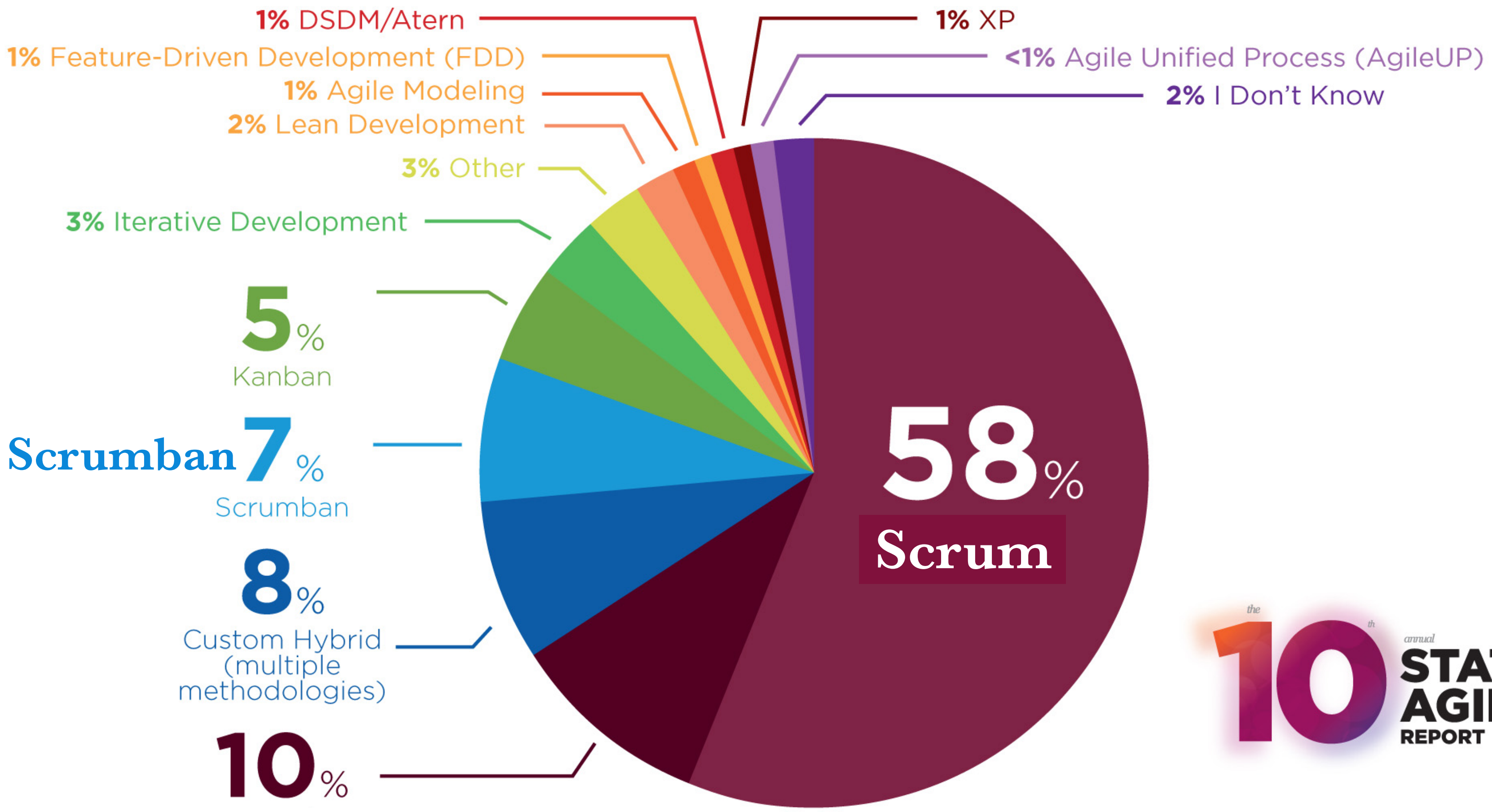
- **"Make each program do one thing well**
  - **To do a new job, build fresh rather than complicate old programs …"**
- **"Design and build software, even operating systems, to be tried early, ideally within weeks.**
  - **Don't hesitate to throw away the clumsy parts and rebuild them."**
- **Software utilities "were continually improved by much trial, error, discussion, and redesign."**

McIlroy, Pinson, and Tague [1978]

2015 Survey: Agile Methods Used

THE UNIVERSITY OF ARIZONA
Computer Science

Over 70% use some form of Scrum

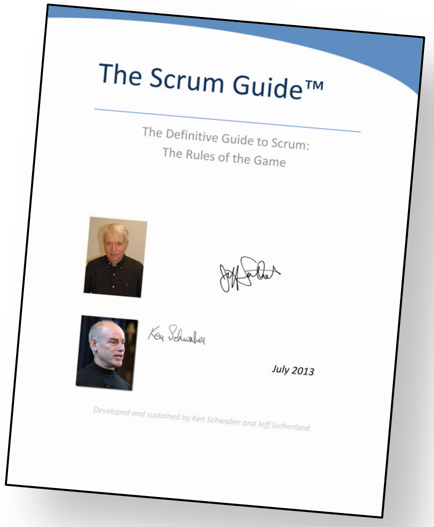


Scrumban

Scrum

Scrum/XP Hybrid

Scrum


The Scrum Guide™

The Definitive Guide to Scrum:
The Rules of the Game

[illegible]

Ken Schwaber

July 2013

Developed and sustained by Ken Schwaber and Jeff Sutherland

Reference: "The Definitive Guide to Scrum" by Jeff Sutherland and Ken Schwaber [2013]

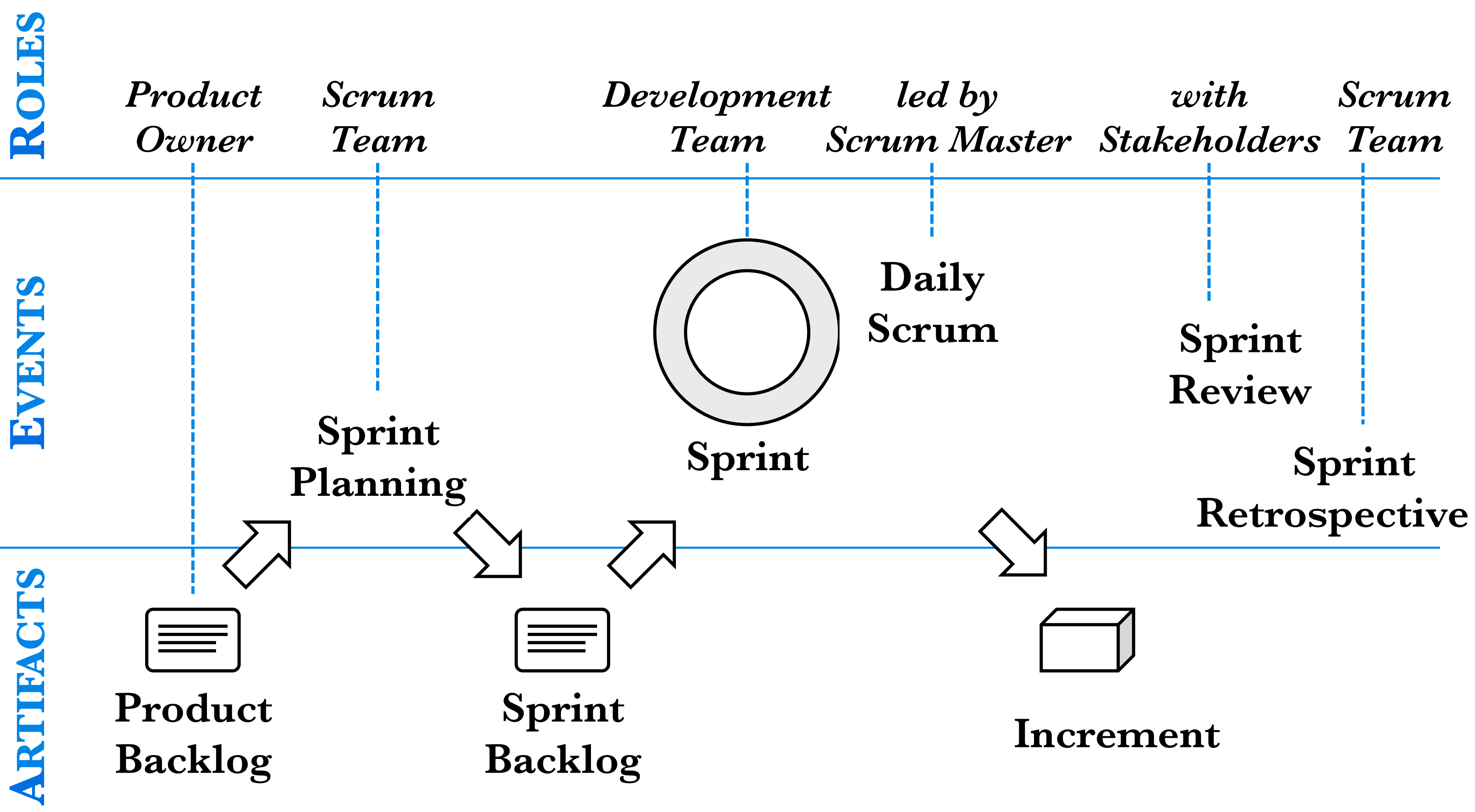
The Scrum Framework

Sprints and Scrum Meetings are now used more broadly

- **Scrum is not a specific process, it is a framework**
  - Within it you can apply various processes and techniques
- **The Scrum framework consists of rules for**
  - Scrum Teams and their roles
  - Scrum Events
  - Scrum Artifacts
- **Started with Jeff Sutherland and Ken Schwaber in 1995**

Elements of Scrum

Roles, Events, and Artifacts



Scrum Teams

Teams are cross functional and self organizing

- **Product Owner is a person, not a team**
  - Represents the voice of the customer
  - Responsible for managing and prioritizing the product backlog
- **The Development Team**
  - Deliver potentially shippable product increments
  - No one tells the development team how to implement backlog items
- **Scrum Master is a coach (servant-leader)**
  - Removes external impediments and facilitates the internal process

http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

Scrum Events

Events are time boxed

- **Sprints are iterations of one-month or less**
  - No changes to Sprint Goal, but scope may be clarified/re-negotiated
- **Sprint Planning: at most 8 hrs for 1-month sprint**
  - Set Goal, select backlog items, informed by development forecasts
- **Daily Scrum: 15 min, to synchronize Development**
  - What did I do yesterday? What will I do today? Any impediments?
- **Sprint Review: at most 4 hrs for 1-month Sprint**
  - Inspect increment and adapt the Product Backlog if needed
- **Sprint Retrospective**
  - To plan improvements for the next Sprint

http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

Scrum Artifacts

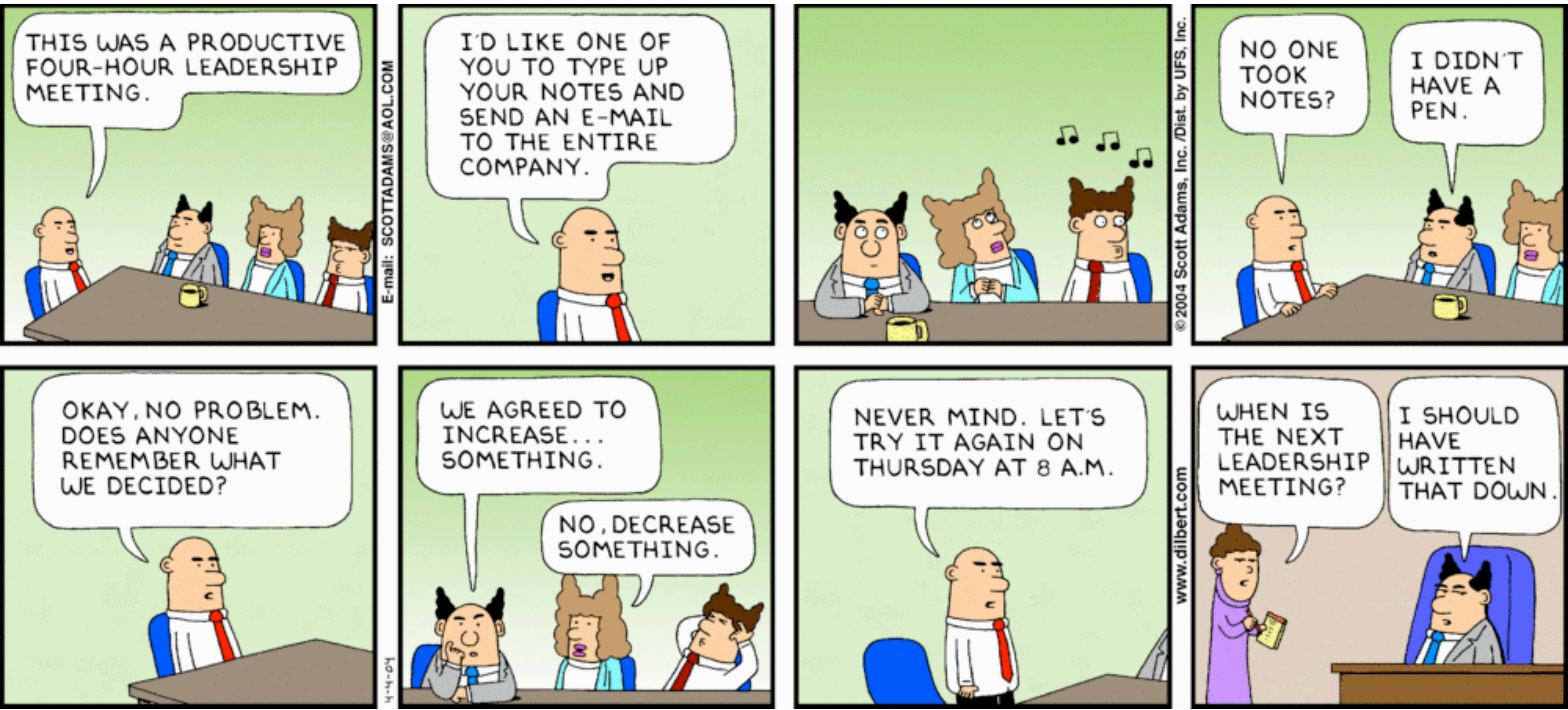
Artifacts are to maximize transparency of key information

- **Product Backlog**
  - Ordered list of everything that might be needed for a product
- **Sprint Backlog**
  - All Product Backlog items selected for a Sprint
- **Increment**
  - Sum of all Product Backlog items that are completed during a Sprint

Meetings

Have an organizer for each meeting – rotate the role

- Send out an agenda in advance, with what members need to prepare
- Meet face-to-face
- Take notes; record "who will do what by when"


THIS WAS A PRODUCTIVE FOUR-HOUR LEADERSHIP MEETING.
E-mail: SCOTTADAMS@AOL.COM
I'D LIKE ONE OF YOU TO TYPE UP YOUR NOTES AND SEND AN E-MAIL TO THE ENTIRE COMPANY.
© 2004 Scott Adams, Inc./Dist. by UFS, Inc.
NO ONE TOOK NOTES?
I DIDN'T HAVE A PEN.
OKAY, NO PROBLEM. DOES ANYONE REMEMBER WHAT WE DECIDED?
4-4-04
WE AGREED TO INCREASE... SOMETHING.
NO, DECREASE SOMETHING.
NEVER MIND. LET'S TRY IT AGAIN ON THURSDAY AT 8 A.M.
www.dilbert.com
WHEN IS THE NEXT LEADERSHIP MEETING?
I SHOULD HAVE WRITTEN THAT DOWN.

Extreme Programming (XP)

“an always-deployable system to which features, chosen by the customer, are added and automatically tested on a fixed heartbeat.”

— Kent Beck

Agile Values and XP

- Customer Collaboration: User Stories
  - A user story is a brief description of a feature or piece of functionality
- Responding to Change: Iteration Planning
  - "Time boxed" iterations; customers set priorities
- Working Software: Testing and Refactoring
  - Testing is essential for maintaining a state of clean working software
- Individuals and Interactions: Philosophy
  - "values of communication, feedback, simplicity, courage and respect"
  - Pair programming: claims that the added cost is 15% not 100%

From Extreme Programming Explained

THE UNIVERSITY OF ARIZONA
Computer Science

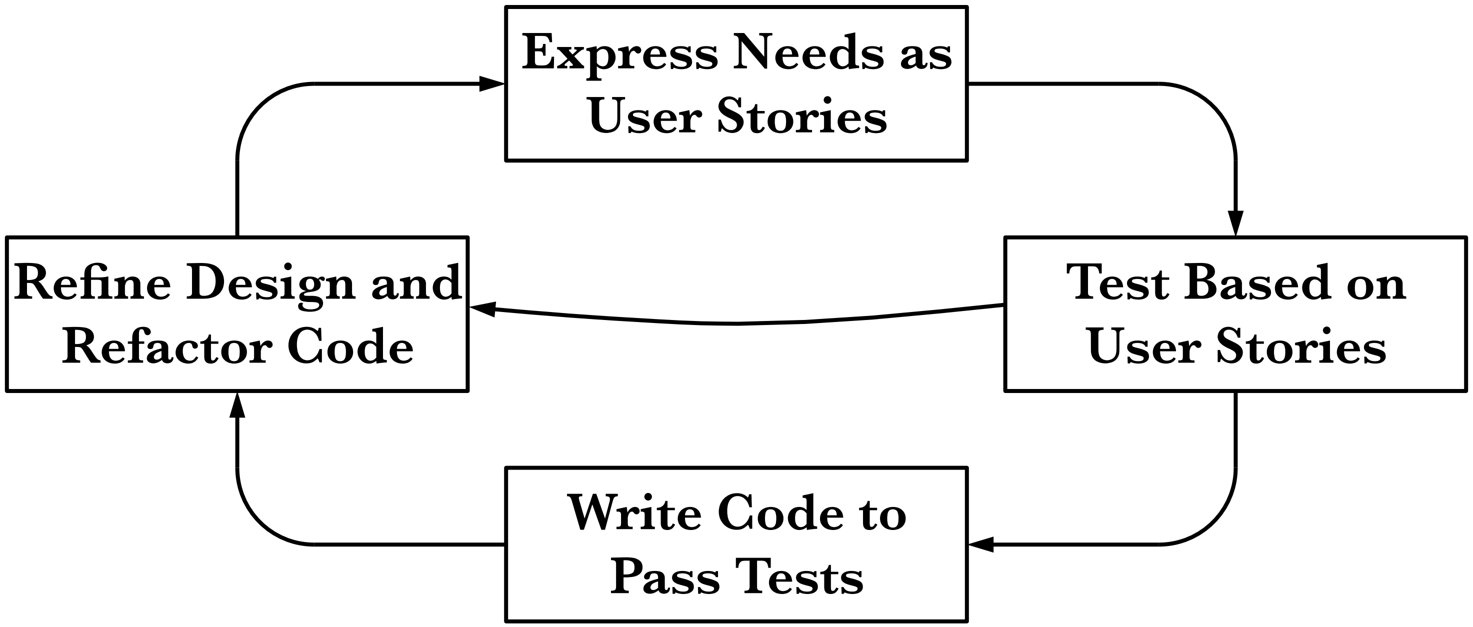
Take commonsense principles and practices to extreme levels

- If code reviews are good, we'll review code all the time (pair prog.).
- If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
- If design is good, we'll make it part of everybody's daily business (refactoring).
- If simplicity is good, we'll always leave the system with the simplest design … (the simplest thing that could possibly work).
- If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
- If integration testing is important, then we'll integrate and test several times a day (continuous integration).
- If short iterations are good, we'll make the iterations really, really short—seconds and minutes and hours, not weeks and months and years (the Planning Game).

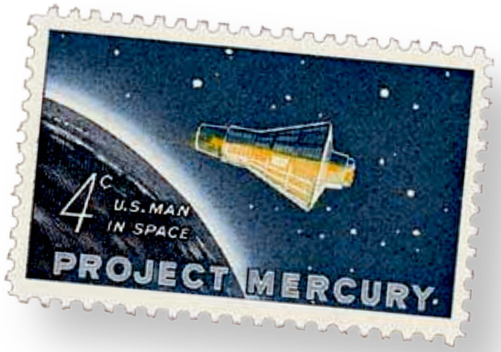
Beck [2000]: Extreme Programming Explained

Software Development Using XP

Key Practices


Express Needs as User Stories
Test Based on User Stories
Write Code to Pass Tests
Refine Design and Refactor Code

Test-Driven Development



The "practice of test-first development, planning and writing tests before each micro-increment" was used as early as NASA's Project Mercury, in the early 1960s

Source: Larman and Basili [2003[

Test Driven Development

Red-Green-Refactor: red for fail tests, green for pass

- **Each new feature begins with writing a test**
  - Base the test on requirements; e.g., user stories, use cases, exceptions
- **Run all tests and see if the new one fails**
  - Does the new test fail for the expected reason?
  - If the test succeeds, either the feature exists, or the test is defective
- **Write just enough code to pass the test**
  - The code can be inelegant; it will get cleaned up in a later step
- **Run all tests**
  - If all tests now pass, the code meets the tested requirements
- **Refactor code as necessary**
  - See next slide

http://en.wikipedia.org/wiki/Test-driven_development

Test-Driven Development, continued

- **Refactor code as necessary**
  - Remove any duplication
  - Move code from where it was added to pass a test to where it belongs
  - Does the code reflect the developer's intent?
  - Do the variable and method names reflect their current usage?
  - Minimize the number of classes and methods
  - Refactor tests as well; tests are part of the maintenance overhead
- **Repeat**
  - Add another test that adds functionality
  - Keep steps small, a few edits at a time
  - If the new code does not rapidly satisfy the new test or if other tests fail unexpectedly, undo and revert – avoid excessive debugging

http://en.wikipedia.org/wiki/Test-driven_development

User Stories

User Stories: Template and Example

User Stories Capture Customer Requirements

- **User Story Template**
  - ***Feature***: [Name]

    As a [kind of stakeholder] ***I want to*** [do some task],

    so that [I can achieve some benefit]

- **ATM Example**
  - ***Feature***: Account holder withdraws cash

    As a customer

    I want to withdraw cash from an ATM,

    so that I don't have to wait in line at the bank

Developing a User Story

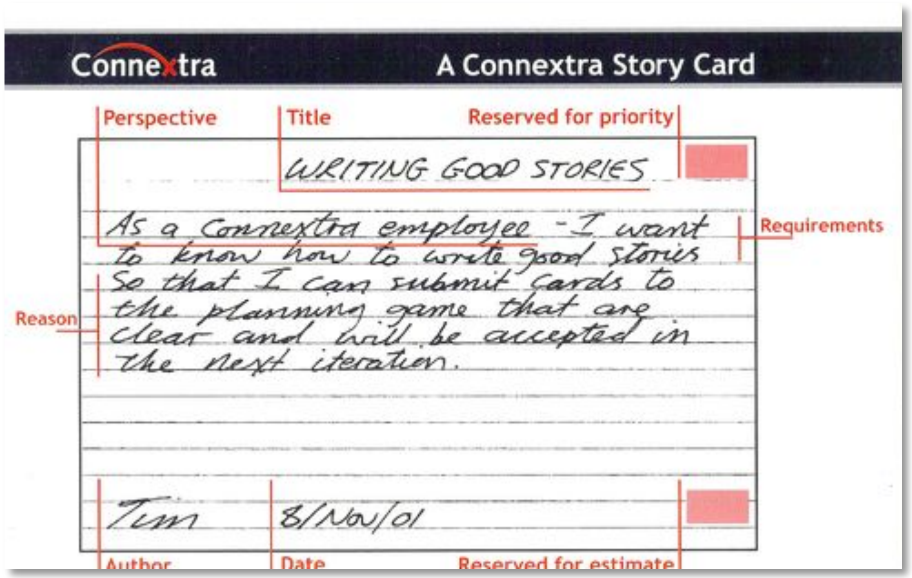
Stakeholders may need help framing the narrative

- **Involves multiple people**
  - The customer may know what they want, but not the cost/benefit
  - A developer can provide a ballpark estimate of what it would take
  - Tester helps define the scope in the form of an acceptance test
- **Missing stories**
  - If the "want" won't deliver the "benefit" is there a missing story?
  - If a story is too complex to fit into an iteration, break it down
- **A spike is an investigation**
  - If the developers don't see how to even make a ballpark estimate, they may need a spike to understand the customer requirements

Structured User Stories

Keep stories simple enough to fit on a 3x5 index card

- Connextra Story Card
  - As a … I want … so that …
  - Date
  - Author
  - Customer Priority
  - Developer Estimate of Effort



Connextra A Connextra Story Card

Perspective | Title | Reserved for priority

WRITING GOOD STORIES

As a Connextra employee - I want to know how to write good stories So that I can submit cards to the planning game that are clear and will be accepted in the next iteration.

Requirements

Reason

Tim 8/Nov/01

Author | Date | Reserved for estimate

- The template lives on
  - Sadly, the startup, Connextra, didn't make it

3C's Checklist for User Stories

Purpose of a User Story

- **Card**
  - Identify a requirement, not capture all details
- **Conversation**
  - Aid discussions with the customer, initially and during planning
- **Confirmation**
  - In the form of an acceptance test

Acronym due to Ron Jeffries http://ronjeffries.com/xprog/articles/expcardconversationconfirmation/

INVEST check list for User Stories

A good user story should be:

• **Independent**	of all others
• **Negotiable**	not a specific contract for features
• **Valuable**	or vertical increment of functionality
• **Estimable**	to a good approximation
• **Small**	so as to fit within an iteration
• **Testable**	in principle, even if there isn't a test for it yet

Acronym due to Bill Wake https://www.agilealliance.org/glossary/invest/

SMART Acronym applied to User Stories

A good user story should be:

- **Make user stories SMART, where SMART is for**
  - *Specific*
  - *Measurable*
  - *Achievable*
  - *Relevant*
  - *Time-bound*
- **Minimum Viable Product**
  - Subset of the full set of user stories that would make for a viable product

Acceptance Criteria Template for User Stories

Simple Form: Given … when … then …

- ***Given*** some initial context (the preconditions),

  and some more context, …

  When an event occurs,

  Then ensure some outcome

  and another outcome …

- **Not all cases are this simple**
  - May need a sequence of "thens" and "whens"; e.g., with menus

Acceptance Tests should be Executable

May have multiple given-when-then for a user story

- **ATM Case 1:** Account is in credit

Given the account is in credit

    and the card is valid

    and the dispenser contains cash

When the customer requests cash

Then ensure the account is debited

    and ensure cash is dispensed

    and ensure the card is returned

Acceptance Tests should be Executable

THE UNIVERSITY OF ARIZONA
Computer Science

May have multiple given-when-then for a user story

- **ATM Case 2:** Account is overdrawn past the limit

  Given the account is overdrawn

  and the card is valid

  When the customer requests cash

  Then ensure a rejection message is displayed

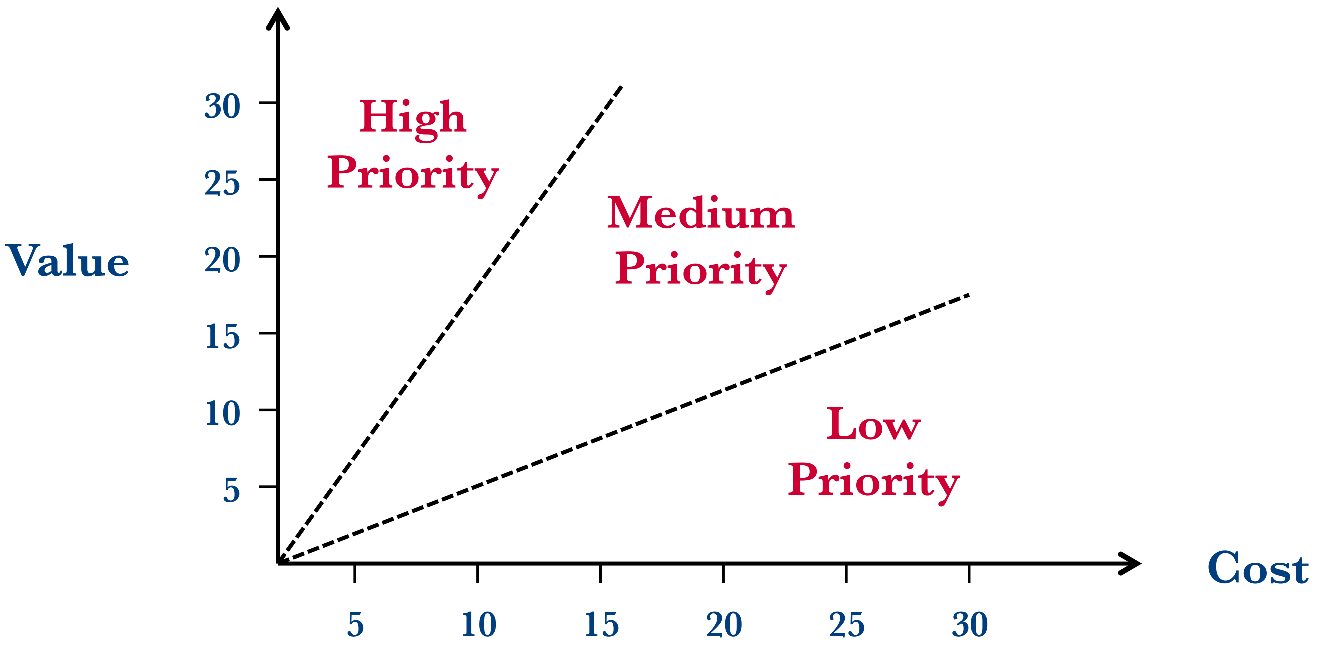
  and ensure cash is not dispensed

  and ensure the card is returned

Rough Estimates for Iteration Planning

Not All Requirements are Equal!

- **Perform Triage**
  - Some requirements "must" be included
  - Some requirements should definitely be excluded
  - That leaves a pool of "nice-to-haves," which we select from

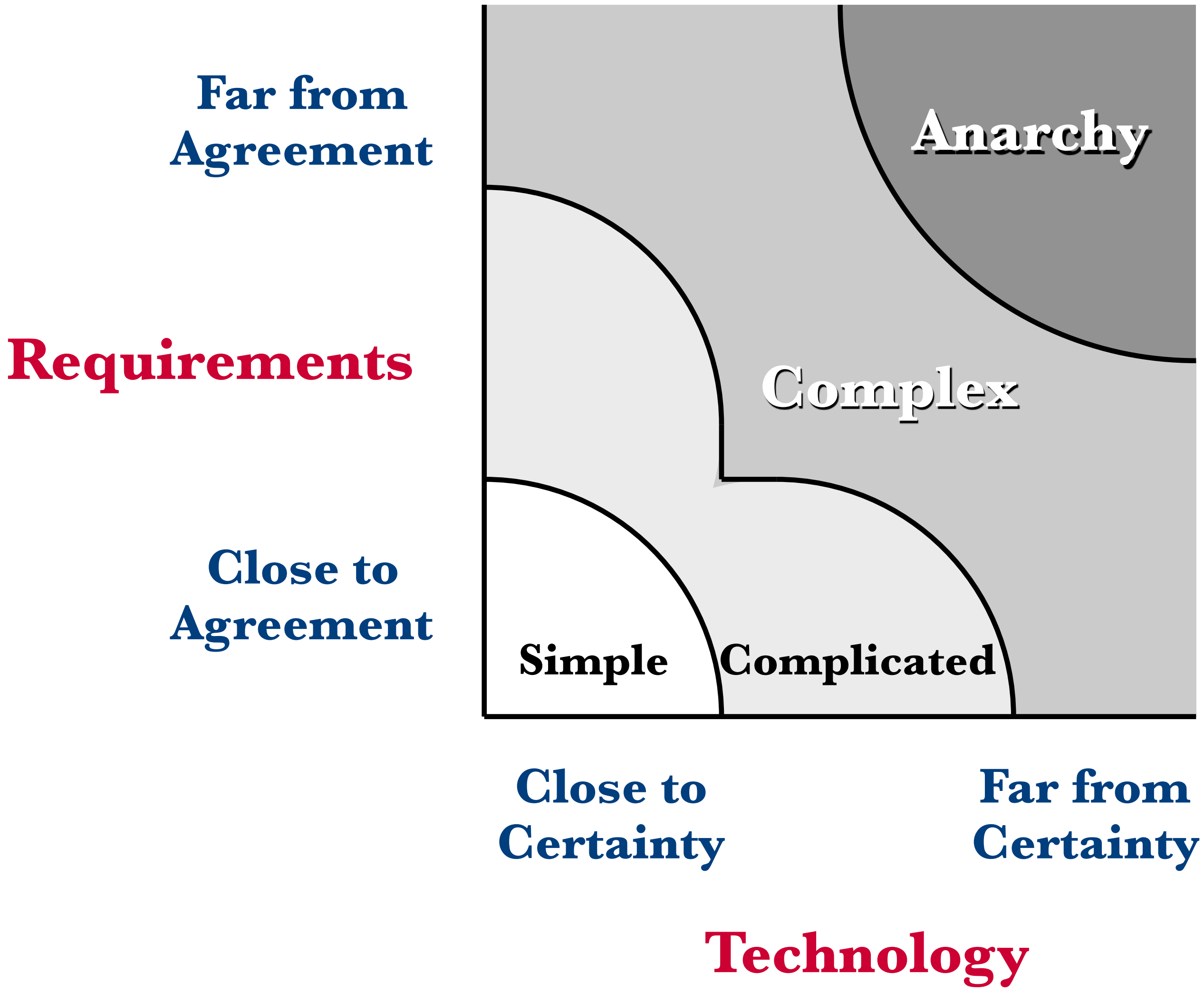


Estimation in Practice

- People tend to underestimate effort needed
- Rule of thumb: Make an estimate, then double it
- Most estimates are made to please the {boss, customer, …}
- Easier to estimate small chunks of work than large ones

Complexity in Development Projects

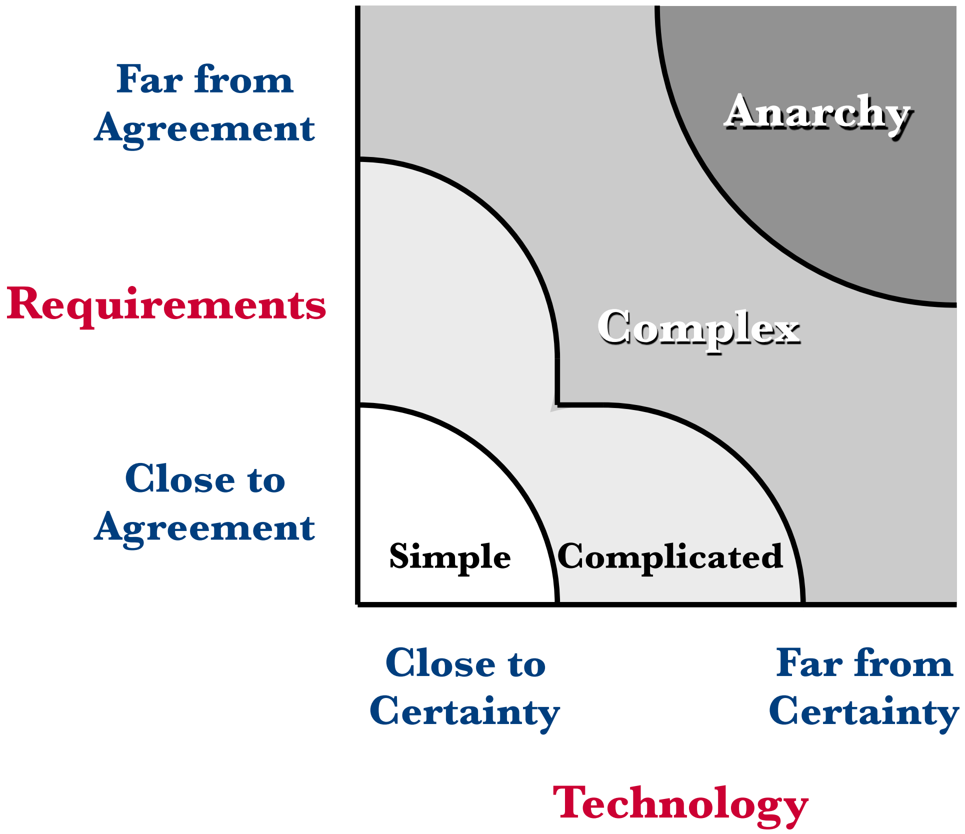
Guidance from Chemical Engineering Projects



Complexity in Development Projects

Applied to User Stories

- **Story Points**
  - Simple: 1 point
  - Complicated: 2-3 points
  - Complex: Break down the story
  - Anarchy: Keep talking to clarify customer needs and goals



Quick Estimates: Another Approach

Story Points for User Stories

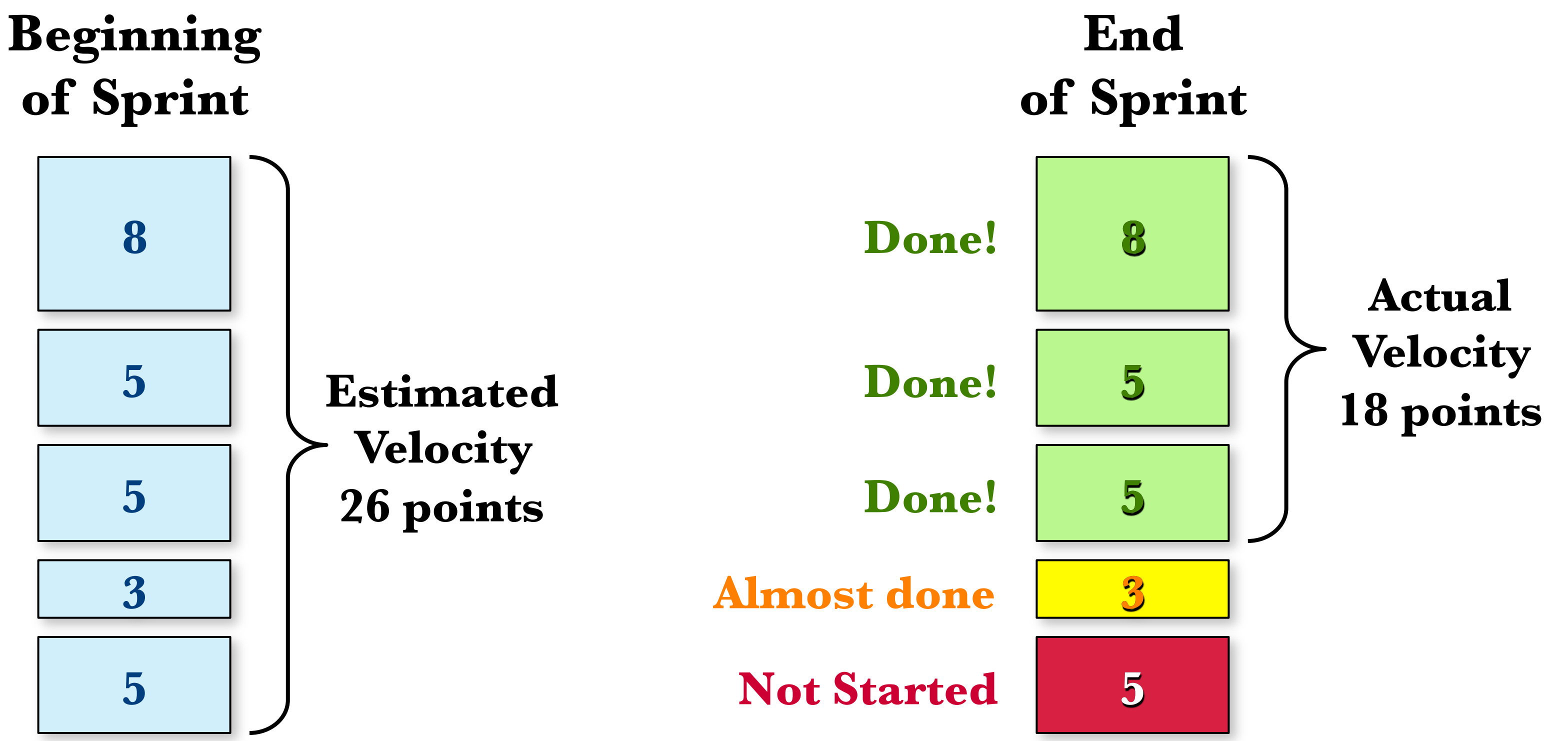
- **1 point**
  - I know how to do it and can do it quickly
  - The team defines quickly
- **2 points**
  - I know how to do it, but it would take some work
- **3 points**
  - I would need to figure out how to do it
  - Candidates for splitting into simpler stories

Quick Estimates

Fibonacci Story Points

- **Point values: 1, 2, 3, 5, 8, 11, 19, …**
  - Easier to assign points if there are some gaps in the scale
- **Relative estimation is easier than absolute**
  - Easier to estimate whether *A* is harder than *B*
  - Than it is to assign point values to *A* and *B*
  - Start with 2 points for a simple story

Continuously Re-estimate Velocity



Last Sprint: Completed

18 points / 45 staff days = 0.4

Next Sprint: Estimated

0.4 × 50 staff days = 20 story points

Limitations of User Stories

User stories can be used together with Use Cases

- **Big Picture?**
  - User stories are at the level of individual features
  - May need Big stories, not just Little stories
- **Completeness?**
  - Customers may have left out something they take for granted
  - Non functional requirements may not have been discussed
- **Deeper Needs?**
  - An "I want" discussion is best suited to Expressed needs
  - Observations and empathy are not explicit in the template