



Western  
UNIVERSITY • CANADA

Topic 4

# Agile Requirements with User Stories

Based on the excellent book, *User Stories Applied: For Agile Software Development* by Mike Cohn. Read this book.

**Computer Science 2212b**  
**Introduction to Software Engineering**  
**Winter 2014**

**Jeff Shantz**  
**[jeff@csd.uwo.ca](mailto:jeff@csd.uwo.ca)**

# Big Design Up Front

## Traditional requirements analysis:

- Talk to stakeholders (customer, end users, etc.)
- Think about the planned system; develop UML models
- Maybe prototype a bit
- Spend months developing a big document covering every requirement of the system up front
- Give said document to developers
- Receive word from developers that the project will actually take 24 months instead of the desired 6 months

# Big Design Up Front: Problems

- Difficult to envision every possible feature needed up front
- Process of documenting every requirement is tedious and error-prone
- Customers often change their minds or come up with new ideas as they see the software being developed
- Requirements documents are long and boring
  - Greater chance they will simply be skimmed or entire sections will be skipped
  - Hard to grasp the big picture behind a 300-page requirements spec

# Big Design Up Front: Problems

- Time wasted writing 3/4 of the requirements that the team won't be able to complete in the allotted 6 months
- More time wasted as the development team decides which requirements it can implement in time
- Levels of indirection between customers and developers
  - Lack of direct communication leads to misinterpretation of the intended functionality
  - Remember the telephone game?

# Big Design Up Front: Problems

- Focusing on a checklist of requirements rather than on the user's goals does not necessarily lead to a good overall understanding of a product:

- 3.4) The product shall have a gasoline-powered engine
- 3.5) The product shall have a four wheels
  - 3.5.1) The product shall have a rubber tire mounted to each wheel
- 3.6) The product shall have a spinning blade mounted on its underside
- 3.7) The product shall have a foam seat

VS.

- The product makes it fast and easy for me to mow my lawn
- I am comfortable while using the product

# Big Design Up Front: Problems

- Software is intangible
  - Very hard to estimate reliably
  - **Pipe dream:** glorious Gantt charts enumerating
    - every task that must be completed
    - the duration of the task
    - the order in which the task must be completed

# Big Design Up Front: Problems

- As software is built, customers often change their minds about existing features and think up new features
- Requirements changes are called a *change of scope*
  - Implies that the scope of the project was previously fully defined
  - Implies that a project is complete when it fulfills its *list of requirements* rather than its *intended users' goals*

# Software Requirements = Communication

**Those who want the software must communicate with those who will build it**

- A project relies on information from those who view the software from a *business perspective*, and those who view it from a *development perspective*

# Software Requirements = Communication

- **If the business side dominates**, it mandates functionality and deadlines with little concern that:
  - the development team can deliver the requested functionality on schedule
  - the development team understands exactly what is being requested
- **If the development side dominates**, it
  - replaces the language of business (the language of the domain) with technical jargon
  - loses the opportunity to learn what is needed by listening

# Solution?

## Given that:

- Requirements often change throughout a project
- Reliable estimation is extremely difficult
- It is difficult to come up with all requirements up front

## What do we do?

- Make decisions based on the information we have
- Do it often

## Spread decision-making across the duration of the project:

- Need a process that gets us information as early and often as possible

# User Stories

- **Describe functionality valuable to a user/customer**
- **3 components**
  - **Card:** A written description of the story used to plan and serve as a reminder
  - **Conversation:** Conversations about the story to flesh out its details
  - **Confirmation:** Tests that convey and document details; confirm to us when the story is complete
- Often written on the front of an index card, with confirmation details written on the back

# User Stories: Customer Team

- Stories are usually written by a **customer team**:
  - Ensure the software will meet the needs of its users
  - Includes developers, testers, product managers, actual users, customers, etc.
- Stories must be written in the *language of the business* – not in technical jargon
  - Allows the customer team to be able to prioritize stories

# Example: User Stories

## Good examples:

A user can post her resume to the web site

A user can search for jobs

A company can post new job openings

## Poor examples:

The software will be written in C++

The program will connect to the database through a connection pool

Users *don't care* about the programming language used or technical details such as how the application connects to a database

# User Stories: Conversation

- Notice that *many details are missing*
  - What fields can the user search for jobs on?
  - Does the user have to be logged in?
- A user story is a **reminder to have a conversation**
  - We will discuss these details with the customer at the time they become important (just-in-time requirements analysis)
  - The conversation is the important part – not the story itself

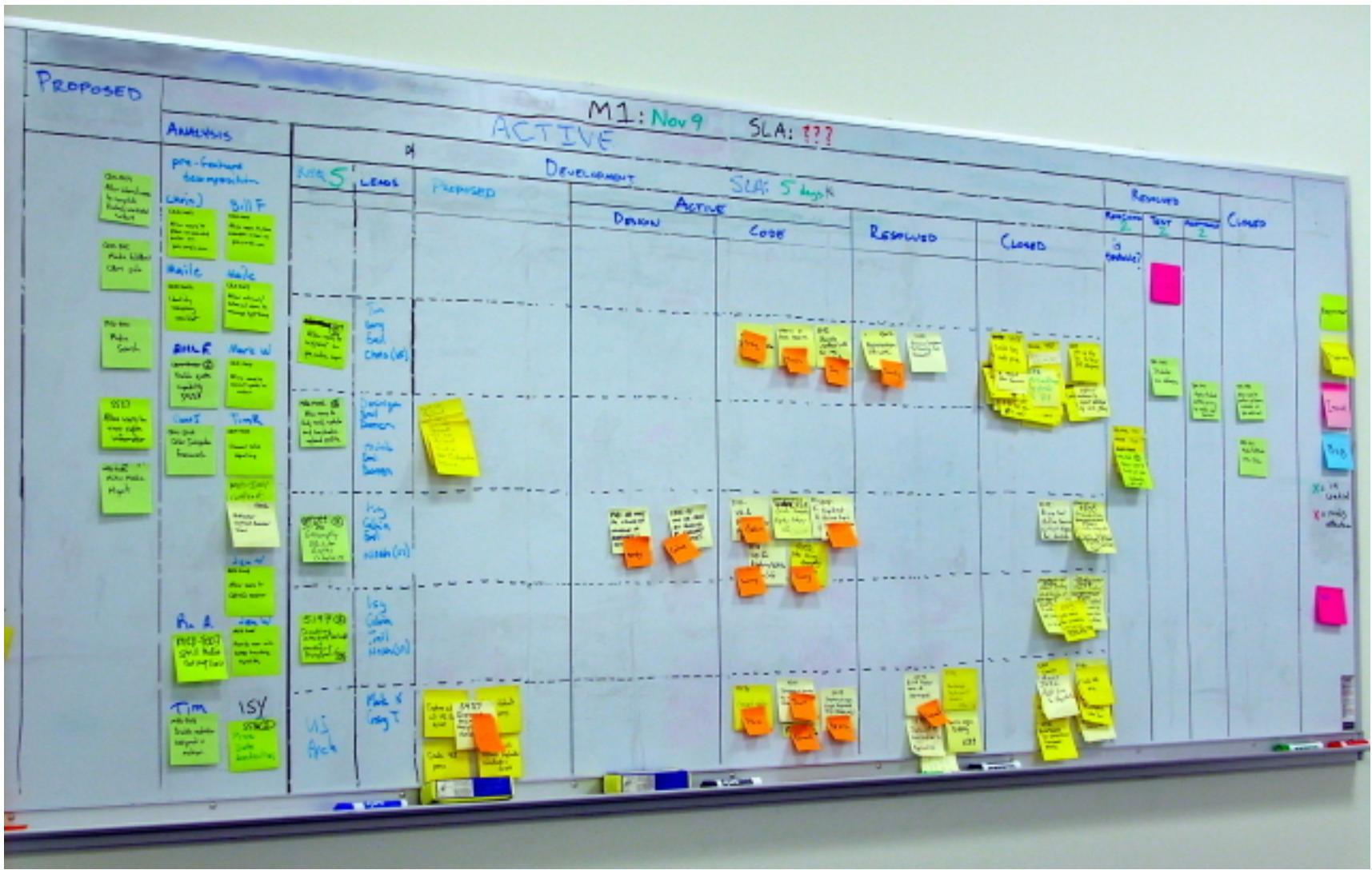
# User Stories: Conversation

## Bottom Line

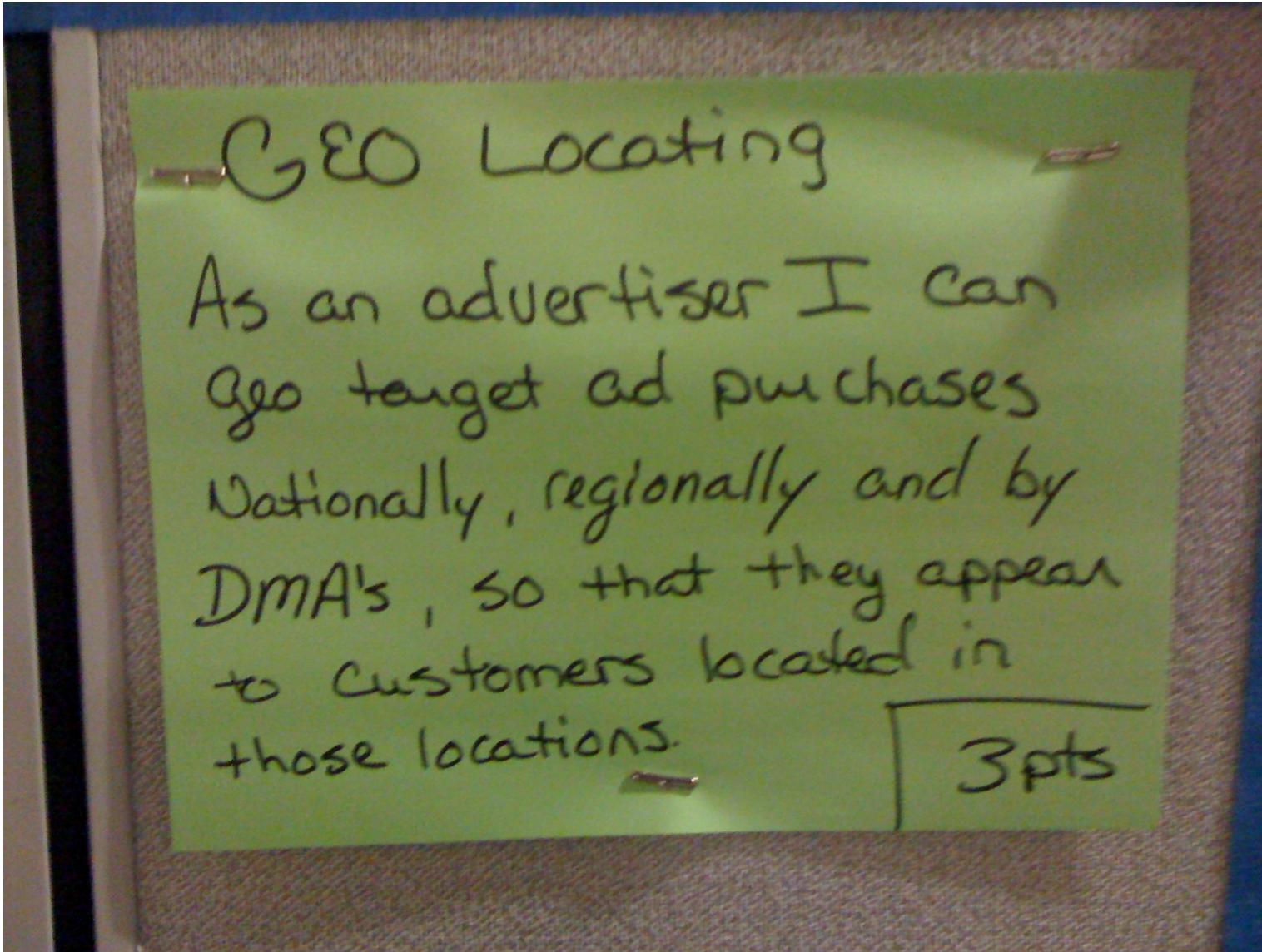
Cards *represent* customer requirements rather than *document* them.

**The card contains the story; the details will be worked out in the conversation, and recorded in the confirmation.**

# User Stories: People Really Do Use Them



# User Stories: People Really Do Use Them



# User Stories

## Front of Card

- User story
- Estimate in story points (more later)

A recruiter can post a new job (3 points)

# User Stories

## Back of Card

- Captures expectations of the user in the form of acceptance tests
- These tests will help to confirm that a story is complete

Try it with an empty job description (fail)

Try it with a really long job description (fail)

Try it with a missing salary (fail)

Try it with a six-digit salary (pass)

# User Stories: Estimation

- Cost of a story is the estimate given to it by the developers
- Each story is assigned an estimate in *story points*
- **Story Points**
  - Indicate the size and complexity of the story relative to other stories
  - A story estimated at four points is expected to take twice as long as a story estimated at two points

# User Stories: Estimation

**Each team defines its *own* meaning of story points:**

- One team might treat a point as an ideal day of work
- Another team might define a story point as an ideal week of work
- Yet another team might treat a story point as a measure of the complexity of a story

**Recommendation for the project:** treat one story point as an *ideal* evening of work for one developer

- ideal = no interruptions, good focus/concentration

# User Stories: Estimation

**The back of each card contains *acceptance tests***

- Capture important aspects about stories
- Verify that a story was developed to work exactly the way the customer team expected it to work
- Run frequently – usually automated
- Intentionally left short and incomplete
- Can add/remove tests at any time

**The goal is to convey additional information so developers will know when the story is done**

# Example: Acceptance Tests

**Story:** A user can pay for the items in her shopping cart with a credit card

Test with Visa, MasterCard and American Express (pass)

Test with Diner's Club (fail)

Test with a Visa debit card (pass)

Test good, bad, and missing security codes

Test with expired cards (fail)

Test with different purchase amounts (try going over card limit)

# Example: Acceptance Tests

- These tests capture the expectations that the system will handle Visa, MasterCard, AMEX, and no others
- By providing tests to the developers early, the customer team has:
  - Stated their expectations
  - Reminded the developers of a situation they might have forgotten (e.g. trying expired cards)

# Story-Driven Development

- **Iteration Length:** customer team and developers choose length of each iteration
  - Might be anywhere from 1 to 4 weeks
  - Your iterations will be approximately 2 weeks
  - Same iteration length used for entire project
- At the end of each iteration, developers responsible for delivering *fully usable code* for some subset of the application
  - Functionality captured by the stories in the completed iteration should now be fully usable

# Story-Driven Development: Release Planning

**Velocity:** Developers estimate how much work (in story points) they'll be able to complete each iteration

- First estimate will be wrong but is useful to roughly determine the iteration schedule

## To plan releases

- Sort stories in piles representing iterations
- Story points in each pile must be  $\leq$  estimated velocity
- Highest-priority stories go in the first pile (iteration 1)
- Next highest-priority stories go in the second pile (iteration 2)
- ...

# Story-Driven Development: Release Planning

**Before each iteration starts, customer team can make mid-course corrections to the plan**

- As we complete iterations, we learn the development teams' ***actual velocity***
- Can work with it instead of the estimated velocity
- Each pile of stories may then need to be adjusted by adding or removing stories

# Example: Story-Driven Release Planning

## Stories

Story	Story Points
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story G	5
Story H	5
Story I	5
Story J	2

**Release Plan**  
Estimated velocity of 13

Iteration	Stories	Story Points
1	A, B, C	13
2	D, E, F	12
3	G, H, J	12
4	I	5

# Story-Driven Development: Prioritization

**To plan releases, customer team must prioritize stories**

- Desirability of a feature to a broad base of users/customers
- Desirability of a feature to a small number of important users/customers
- Cohesiveness of a story in relation to others
  - e.g. a *zoom out* story might not be high priority on its own, but might treated as such because it is complementary to *zoom in*, which is high priority

# Story-Driven Development: Prioritization

**Developers have different priorities for many stories:**

- May suggest a story's priority be changed based on technical risk or because it is complementary to another
- Customer team listens to their opinions, but ultimately sets priorities in a manner that maximizes value delivered to the organization

# BDUF vs. Story-Driven Development

## Traditional waterfall model

- Write requirements; analyze them; design, code, and test solution
- Customers involved at beginning to write requirements
- Customers involved at the end for acceptance testing
- Between requirements and acceptance: customers often disappear

## Story-Driven Project

- Customers and users involved throughout duration of the project
- Not allowed to disappear in the middle of the project

# Writing Good User Stories: INVEST

Good user stories should follow the **INVEST** acronym:

- I ndependent
- N egotiable
- V aluable
- E stimable
- S mall
- T estable

# Writing Good Stories: Independent

**Dependencies between stories should be avoided**

- Lead to prioritization and planning problems
- Make estimation more difficult
- Need to have the flexibility to easily move stories around in the release schedule, if needed

**Solutions:**

- Combine dependent stories into a larger, independent story
- Find a different way of splitting the stories

# Writing Good Stories: Independent

## Dependent Stories:

The customer should be able to pay with Visa

The customer should be able to pay with MasterCard

The customer should be able to pay with AMEX

- The first credit card will take 3 days to support
- Each additional card will take 1 day
- Which day should receive the 3 day estimate?

## Better Split:

The customer should be able to pay with one type of credit card

The customer should be able to pay with two additional types of credit cards

# Writing Good Stories: Negotiable

## **Stories should be negotiable**

- They are not written contracts or requirements

## **Stories are reminders to have a conversation**

- Do not need to include all relevant details
- We negotiate the details in a conversation between the customer team and development team
- If we do know some details when a story is written, we annotate the card with those details

# Writing Good Stories: Negotiable

**Right amount of information to the developer and customer who will discuss the story:**

A company can pay for a job posting with a credit card.

Note: accept Visa, MasterCard, AMEX. Consider Discover.

- Too many missing details for developers to view the story as definitive

# Writing Good Stories: Negotiable

## Too much detail:

A company can pay for a job posting with a credit card.

Note: accept Visa, MasterCard, AMEX. Consider Discover. On purchases over \$100, ask for card ID number from back of card. The system can tell what type of card it is from the first two digits of the card number. The system can store a card number for future use. Collect the expiration month and year of the card.

- Can lead to mistaken belief that the story card reflects all the details and there's no further need to discuss the story with the customer.

# Writing Good Stories: Valuable

**User stories should be valuable to the *users and customers***

- Avoid stories valued only by developers
  - Should be written so that the benefits to the customers / users are apparent
  - This allows customers to intelligently prioritize the stories

# Writing Good Stories: Valuable

## Poor Examples:

All connections to the database are through a connection pool.

All error handling and logging is done through a set of common classes.

## Better:

Up to fifty users should be able to use the application with a 5-user database license.

All errors are presented to the user and logged in a consistent manner.

# Writing Good Stories: Estimable

**Developers must be able to estimate the size of a story**

- Three reasons why a story might not be estimable:
  1. Developers lack *domain knowledge*
  2. Developers lack *technical knowledge*
  3. The story is too big

# Writing Good Stories: Estimable

## 1. Developers lack domain knowledge

- If developers do not understand a story:
  - Should discuss it with the customer who wrote the story
  - Not necessary to understand all details
  - Only need a general understanding of the story

# Writing Good Stories: Estimable

## 2. Developers lack technical knowledge

- Send one or more developers on a *spike*
  - A brief experiment to learn about an area of the application
  - Developers learn just enough to estimate the task
  - Spike is time-boxed, allowing us to estimate the spike
  - Inestimable stories turn into two stories:
    - The spike
    - The actual story

# Writing Good Stories: Estimable

## 3. The story is too big

- **Example:** *A job seeker can find a job.*
- Developers will need to disaggregate it into smaller, constituent stories

### Stories that are too large are called **epics**

- Too large to estimate, but useful as placeholders
  - Reminders about big parts of the system to be discussed
  - Allow us to make a conscious decision to temporarily gloss over large parts of the system
  - Can be assigned a large, pulled-from-thin-air estimate

# Writing Good Stories: Small

## **Stories should be small, but not too small**

- Stories that are too large or small make planning difficult

### **Example for a job search web site:**

A job seeker can post her resume.

Too large? Small? Just right?

# Writing Good Stories: Small

A Job Seeker can enter a date for each community service entry on a resume.

A Job Seeker can edit the date for each community service entry on a resume.

A Job Seeker can enter a date range for each prior job on a resume.

A Job Seeker can edit the date range for each prior job on a resume.

Too large? Small? Just right?

# Writing Good Stories: Small

A Job Seeker can create resumes, which include education, prior jobs, salary history, publications, presentations, community service, and an objective.

A Job Seeker can edit a resume.

A Job Seeker can delete a resume.

A Job Seeker can have multiple resumes.

A Job seeker can activate and deactivate resumes.

Too large? Small? Just right?

# Writing Good Stories: Testable

## **Stories must be written so as to be testable**

- Successfully passing its tests proves a story has been successfully implemented
- If the story cannot be tested, how will the developers know when they have finished implementing it?
- Untestable stories commonly appear with non-functional requirements:

A user must find the software easy to use.

A user must never have to wait long for any screen to appear.

# Writing Good Stories: Testable

- Tests should be automated as much as possible
  - Strive for 99% automation – not 10%
- Things change quickly
  - Code that worked yesterday may be broken today
  - Automated tests detect this quickly
- Some stories cannot be tested automatically, however

A novice user is able to complete common workflows without training.

- Can be tested, but not *automated*

# User Stories: Summary

- Emphasize verbal rather than written communication
  - Remind us to have a conversation
- Comprehensible by both customers and developers
  - Written in the language of the domain
- The right size for planning
  - Not too large, not too small
- Work for iterative development
  - As long as we follow INVEST, we can easily move stories between iterations

# User Stories: Summary

- Encourage deferring detail
  - We may not even need a story – don't waste time detailing it up front

# User Stories: Team Assignment 1

- We will use an iterative, story-driven process for the team project
- Team assignment 1 requires you to develop user stories for all requirements of the project:
  - You will enter these on <http://www.pivotaltracker.com>
  - You are expected to keep this tracker up to date as the project progresses
  - We will be checking your trackers on each team assignment

# User Stories: Team Assignment 1

## As the owner of a story:

- Once finished coding, you are expected to run the acceptance tests from the story to ensure it is complete
- Debug as needed until acceptance tests pass
- Mark the story as **Accepted** once everything is passing

# User Stories: Team Assignment 1

**Your goal for each iteration:**

*Have a fully usable application at the end of the iteration,  
extended with the functionality described by the stories included  
in the iteration.*

# User Stories: Team Assignment 1

- Create a public project on pivotaltracker.com
- Use 2 week iterations
- Select an initial velocity that allows:
  - At least 6 user stories to be completed by the due date for Team Assignment 2
  - All stories to be completed by the project deadline
- Create epics for every major feature in the program
  - You might have an epic for each window/view, or you might choose to break up the requirements differently

# User Stories: Team Assignment 1

- Break each epic into multiple user stories to cover each requirement of the project
- Each story should include:
  - The story itself
  - A list of acceptance tests and any other notes in the **Description** field
  - An estimate of the length of each story in story points
  - The epic with which it is associated in the **Label** field

# User Stories: Team Assignment 1

- Prioritize each story
- For each story in the first iteration:
  - Decide who is going to complete the story
  - Mark that/those individual(s) as the owner(s) of the story



Western  
UNIVERSITY • CANADA