User Stories & Agile Planning

CSCI 5828: Foundations of Software Engineering
Lecture 08 — 09/15/2016
Goals

• Present an introduction to the topic of user stories
  • concepts and terminology
  • benefits and limitations

• Present an introduction to iteration planning
  • Estimating User Stories
  • Planning a Release
  • Planning an Iteration
  • Measuring and Monitoring Velocity
Credit Where Credit is Due

• This material is drawn from a textbook I used for this class in Fall 2014
  • “User Stories Applied” by Mike Cohn
    Publisher: Addison-Wesley/Pearson Education
  • It’s a great book for going in depth on the topic of user stories
User Stories

• User stories are a means to **capture requirements** during the analysis phase of software development
  
  • whenever that phase occurs during your particular software life cycle
  
  • (in agile life cycles, analysis can happen at any time)

• They are a **lightweight mechanism** for *spreading decision making* out across a software development project with respect to individual features
  
  • We know we need feature X but we don’t know much about it?
    
    • name it and put it in a user story
  
  • We learned a little bit more about feature X today?
    
    • add a short note to the user story (or even better, write a test)
Background (I)

- Agile life cycles evolved the notion of a user story because *capturing software requirements is a communication problem*
  
  - Those who want new software need to communicate what they need to those who will build it
  
  - Many stakeholders will provide input to the process
    
    - customers, users, and domain experts
    
    - business and marketing
    
    - developers
Background (II)

• If any group dominates this discussion, the whole project suffers
  • if business dominates, it may mandate features and schedules with little regard to feasibility
  • if the developers dominate, a focus on technology may obscure business needs and the developers may miss important requirements

• Furthermore, the goal is to **understand the user’s problem** and ensure the software meets their needs
  • both business and developers will move on, the users have to live with the developed software day in and day out
Background (III)

• Furthermore, everything about the project is in flux
  • We still don’t understand *exactly* what the user needs
    • Their domain is complex; they are experts, we are novices
    • We’ll get things wrong and need to be corrected
    • We’ll get to a certain point and then they will remember things that they forgot to tell us
    • We’ll show them prototypes and they’ll come up with new ideas
  • We don’t have enough information to make accurate estimates
    • what we thought would be easy, turns out to be very complex
But, we must make progress!

And, so we have to make decisions based on the information we have

We set our scope small (one feature, for instance) and our development life cycle short (one week, for instance)

and then we show the customer what we have

By then, new information will be available and we’ll have feedback on the work we’ve done so far

With that input, we identify the new scope and start a new iteration

We thus spread out the decision making

It’s not “everything up front” but “a little at a time”
User Stories: The Basics (I)

• That’s where User stories come in; they describe **functionality that will be valuable to the user and/or customer**

• Note the distinction:
  
  • **user**: the people who actually use the produced software in their work
  
  • **customer**: a person, not necessarily a user, who is responsible for purchasing the software for a set of users

• Sometimes they are one and the same, but not always

• Note also the use of the word **“valuable”**

• We do NOT implement a feature because it is “cool”
  
  • *we implement features to provide value to users*
User Stories: The Basics (II)

- User stories consist of
  - a **short written description** of a feature used for planning and a reminder
  - **conversations** about the feature used to flesh out its details
  - **software tests** that convey details about functionality and help us determine when the story is completely implemented
- Ron Jeffries calls these three aspects Card, Conversation, and Confirmation
  - He says “card” because traditionally users stories are written on index cards and put up on a wall in the shared space of a development project
    - Using index cards **forces** you to keep the story brief!
User Stories: The Basics (III)

• Example users stories for a website that helps a person’s job search
  • A user can post a resume to the website
  • A user can search for jobs
  • A company can post new job openings
  • Users can restrict access to their resume

• Important:
  • User stores are written so that customers value them
  • This helps maintain a customer perspective within the development team
User Stories: The Basics (IV)

- So, is this a good user story?

  - The software will make use of a bloom filter to determine if a desired data element is in our data set before we perform disk I/O to retrieve it
Not Really

• Is your customer a distributed systems researcher?
  • Then, yes, *maybe*, this might be a good user story
    • (as it is for Cassandra, a popular NoSQL database)
  • But, in general, technical details like this do **NOT** make good user stories
    • These details may change
      • we need to switch from this framework to this other framework to be compatible on a wider range of devices
    • while the fundamental user story does not change
      • Users need to access schedule information
How do we track details?

• The users stories for an application can often be written simply at a high level of abstraction (known as **epic user stories** or **epics** for short); for example:
  • A user can search for jobs
  • A company can post job openings

• But, you need to specify details at a lower level of abstraction
  • how do we do that?

• Three places
  • in the conversations around a user story; we will converge on details
  • more users stories!
  • as tasks when we decide to implement user stories (discussed below)
More users stories

• You can take an epic like “A user can search for a job” and split it into new stories
  • A user can search for a job by attributes (such as …)
  • A user can view information about a job found by a search
  • A user can view profile information about a company offering a job
• On the epic, you note that it’s covered by these other stories and then you go work on those stories
• The challenge: getting the balance right
  • We want to resist the temptation to document everything on a user story
    • Our conversations are the key element where details live (since the details \textit{WILL change} while the user story remains the same)
Tests are integral to User Stories

- At the start of a user story, the “tests” might exist as a set of customer expectations written on the back of a card:
  - Try feature with an empty job description
  - Try feature with a really long job description
  - etc.
- In this form, the tests can come and go as we learn more about the feature:
  - As this particular user story is worked on and implemented:
    - these expectations are transformed into unit tests and integration tests that tell us when the feature is completely implemented
    - We’re not done until all tests have passed!
Benefits

• User stories provide the following benefits
  • They emphasize verbal rather than written communication
  • They are comprehensible by customers and developers
  • They are the right size for planning
  • They encourage and “work” for iterative development
  • They encourage deferring details until you have the best understanding of what you really need to implement a feature
Tool Support

- Systems, like Trello, can provide teams with support for tracking the status of user stories.

Here’s a Trello board with a mix of stories and tasks.
Overview of a Process

• A software development process driven by user stories feels very different than traditional life cycles; for instance, customers are included throughout the process (they do not disappear on you!)
  • to get a project started, a story writing workshop is held to brainstorm what features are valuable to the customer for an initial release
  • developers will assign initial estimates to each story using “points”
  • customers and developers set an iteration length (e.g. 2 weeks)
  • developers then determine their velocity (how much work they can get done in a single iteration)
  • customers assign priorities to the stories
  • iterations are formed by grouping stories by velocity based on their priorities and estimates
Midcourse Adjustments (I)

• This process is tunable (i.e. customizable)
  • It has to be because the developers will make mistakes with respect to
    • the points they assigned to a user story
    • the velocity (number of points per iteration) they chose
  • At the end of each iteration
    • they will know more about their true velocity and
    • they will know more about the skills of their team
      • and thus have different opinions about the estimates that should be assigned to each user story
Midcourse Adjustments (II)

• With this new information, you can
  • return to the remaining groups of user stories (i.e. iterations) and
  • rebalance them
    • stories will get new estimates
    • stories may get new priorities (low to high and vice versa)
    • new stories may get added
    • existing stories may get removed
      • “Our user doesn’t care about this anymore”
    • existing stories may get moved forward or pushed backward
Releases and Iterations

• An agile life cycle is thus broken down into planning releases and planning iterations

  • A release is some major group of functionality that can be put into production (used by its users)

  • A release is composed of many iterations which contain users stories that are going to be implemented during that iteration

• Iterations always last the same amount of time and produce a working system that can be reviewed by the customers

  • Customers provide feedback and midcourse adjustments are made

  • The next iteration begins

• Reminder: A user story is complete when it passes its user-specified tests
Estimating User Stories

- Developers need to assign “points” to a story to indicate how long it will take to implement
  - Our user/customer assigns priorities to stories, not estimates
- There are a number of desirable properties for this approach
  - it allows us to change our minds about an estimate when new info arrives
  - works for both epic stories as well as smaller stories
  - doesn’t take a lot of time; we want to spend our time developing
  - provides useful information about our progress and work remaining
  - is tolerant of imprecision in estimates
  - can be used to plan releases
Story Points

• A point is a unit that can be defined by the development team
  • It might represent “eight hours of uninterrupted work” for one team
  • It might represent “forty hours of uninterrupted work” for another
  • Some use points to represent complexity (lots of points == complex)

• Think of one point as “one ideal work day”
  • where ideal means: a day with no interruptions and the developer can be maximally productive on the task

• Two benefits with this approach
  • it avoids getting too specific: “this story will take 39.5 hours”
  • it gives people confidence: “Yeah, that story is about two days of work”
Estimates belong to the Team

• It is important to have **the team create the estimates for each story**
  
  • The success of the project is attributed to the team not to individuals
    
    • to establish this perspective: make estimates together
      
      • if you get it wrong, it’s the team that failed, not one individual
  
  • In addition, when creating/estimating stories, it may not be clear who will be assigned to this particular story
    
    • therefore, the team works to create the estimate and then individuals assigned to the story later know
      
      • they had a voice in creating the estimate they are working against
      
      • the team is responsible if the estimate is wrong
The Process of Estimation

- One way to do estimation was developed by Barry Boehm
  - the Wideband Delphi approach

- Gather the development team and the customer/user(s)
  - Bring the stories that need estimates and blank index cards
  - Distribute the cards to the development team

- Loop until all stories have estimates
  - Read a story out-loud

- Loop until estimates have converged
  - Engage in Q&A with customer/users about that story
  - Each developer writes an estimate; when ready, show all estimates
  - Developers discuss differences in estimates; raising questions/issues
    - New stories may be created due to this discussion
Triangulate

- After a set of stories have received estimates, developers need to review them and see if they are being consistent
  - Group the stories by number of points and discuss
    - For example, are these two point stories really twice as small as the four points stories?
      - If yes, continue estimating
      - If not, change the estimates
  - This helps the team achieve consistency across the entire set of user stories
  - Later in a development project, the need for triangulation may go down as the team becomes more confident and knowledgable of their abilities
Velocity

• The term velocity is defined as “number of story points completed per iteration”

• Agile software life cycles recommend that

  • before the first iteration begins, the team makes a guess at what their velocity will be

  • if a point means “ideal work day”, you can start with this formula

  • number of team members x number of days in iteration

  • then, your velocity for iteration N is the actual number of points completed for iteration N-1

  • if you completed 32 points in the previous iteration, your velocity for planning the next iteration is 32.
Release Planning

• A release is a version of the system under development that is going to be deployed and put into production use
  
  • Release planning in software development involves having a release roadmap in which the next several releases have been identified
    
    • and the functionality for each release has been specified at a high level
  
  • With a release roadmap, you need to engage in release planning
    
    • users/customers need to assign priorities to estimated user stories
    
    • all stakeholders need to work together to identify the length of an iteration
    
    • Issues include dealing with risk and determining velocity
Assigning Priorities

- One prioritization scheme that may be better than the typical “low/medium/high” approach
  - Must have
  - Should have
  - Could have
  - Won’t have (for this release)

- This approach divides stories into clear buckets that can then be used to assign stories to iterations within the release
  - If a customer can’t assign a priority to a user story, this (typically) indicates that the story needs to be split until clear priorities can be assigned
Risky Stories

• The issue here is what approach should agile projects take
  • tackle risky stories first
  • or go after “low hanging fruit”

• Agile life cycles like to go after low-hanging fruit
  • *high-value functionality that is straightforward to implement*

• This allows time for more information to be gathered about high-risk stories
  • and this additional information may reduce the risk associated with them

• I think you need to balance this with the common issue of “problem avoidance”; make sure you’re clear on what the risks are => such information may produce action items that can reduce the risk and make it feasible
Iteration Length and Expected Duration

- Iteration length is typically from one week to four weeks
  - Agile life cycles recommend selecting shorter lengths to increase the feedback loop with the customer
- The important thing is once the length is selected: **DON’T CHANGE IT!**
  - Your team needs to settle into a comfortable development pace
    - Arbitrary changes to the iteration length will hinder that goal
- Once you have an iteration length, an initial velocity, and a set of prioritized, estimated user stories, you can make initial “ballpark” predictions about how long it will take to create a release
  - \( \text{round\_up(\text{number of points} / \text{velocity})} = \text{number of iterations} \)
  - \( \text{number\_of\_iterations} \times \text{iteration\_length} = \text{number of days until release} \)
Velocity, revisited

• Previously we suggested
  • number of team members x number of days in iteration
  • is a good formula for picking an initial velocity
• However, you need to take into account that “number of days” means “number of IDEAL days”
  • You need to include a conversion factor between an IDEAL day and an ACTUAL day
    • An actual day won’t be eight hours of uninterrupted work due to meetings, interruptions, illness, turnover, etc.
• Ideal velocity for six people with two week iteration (10 business days): 60
• Converting to an ACTUAL day: $6 \times 10 \times .5 = 30$; $6 \times 10 \times .25 = 15$!
Iteration Planning (I)

- The points-based approach to release planning works well
  - It provides enough planning to make progress on the project
  - It lacks enough detail to avoid giving a false sense of accuracy
    - People will be aware that there can be errors made in the estimates and can react once new information is available to make the errors clear
- In iteration planning, you need to engage in more detail to help create accurate work plans over the days allocated to an iteration
  - An iteration planning meeting occurs “between iterations”
    - If it occurs “during” an iteration, then you need to include the time spent on it in your other estimates (perhaps by adjusting your velocity down by a point or two to account for it)
Iteration Planning (II)

• All developers and the customer/user must be present for an iteration planning meeting
  • The developers are required to help identify tasks and make estimates
  • The customer/user is required to answer questions about the stories
• The process involves
  • For each story in the iteration
    • engage in Q&A with customer/user about the story
    • convert story into tasks that need to be completed to finish the story
    • assign each task to a single developer
  • Each developer then estimates each assigned task; performs sanity check
    • if a developer is overloaded, rebalancing or more planning is needed
Tasks

- Task identification takes a story that is written in a customer perspective and transforms it into a set of steps that are written from a developer’s perspective (finally!)

- “A job seeker can search for jobs” might be transformed into
  - Code basic search interface
  - Write controller to handle submissions from search interface and perform the search
    - Ensure that controller can access the database correctly
  - Write a view that will display the results

- Working on this step will require “design thinking” either to come up with an initial design for a system or to integrate this feature into the existing design
Task Estimation

• In release planning, we worked with “ideal days”
  • With task planning, we work with “ideal hours”
• Once a developer has their assigned tasks, they estimate the number of hours it will take to complete each one
  • They then add those hours up to perform a sanity check
  • They can also include a factor to transform ideal hours into actual hours
• Sanity Check
  • Compare number of hours with the length of the iteration
  • If the number of hours to complete the tasks is greater than the number of available hours, then rebalancing is needed
• A team perspective is needed to make this successful
Measuring and Monitoring Velocity

- Once points/priorities have been assigned and releases and iterations have been planned, the most important metric for an agile life cycle is velocity.
  - Velocity tracks how much work is completed in an iteration.
    - Before the iteration, it is a “guess”.
      - A guess that we have increased confidence in over time.
    - After an iteration, it is an actual metric that can be used in assessment.
- How do we measure velocity?
  - The number of points associated with completed stories.
    - Incomplete stories are not included (velocity is an integer not a float).
  - With velocity measured, we can chart our progress in a variety of ways.
Iteration Burndown Charts

Important: This plots the remaining story points; the y value heads towards zero as we complete stories.
Important: This plots the **remaining** task points (i.e. hours); the y value heads towards zero as we **complete** tasks
Summary

• In executing an agile life cycle, you must
  • estimate your stories
  • plan your releases
  • plan your iterations
  • measure your progress

• We have looked at various recommendations for performing these tasks
  • using “ideal days” (stories) and “idea hours” (tasks) for estimates and then using a conversion factor to get to “actual days” and “actual hours”
  • saw example charts to measure actual progress
    • Agile life cycles are brutal; if you fall behind, you’ll know it fast
      • the good news is that you’ll deal with schedule delays quickly and hopefully before they become a problem