Introduction to User Stories

CSCI 5828: Foundations of Software Engineering
Lecture 05 — 09/09/2014
Goals

- Present an introduction to the topic of user stories
  - concepts and terminology
  - benefits and limitations
  - examples
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 A 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 A 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 store 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 produced software day in and day out
Background (III)

• Another important issue during this phase is resource allocation (who should work on what and when and supported by $x amount of funds)

  • If developers have this responsibility, they may
    • trade quality for more features (or vice versa)
    • only partially implement some features
    • or make decisions on their own when they should have sought feedback from business and from the users

  • If business has this responsibility, they may
    • generate way too many features on too small of a budget
      • leading to (lots of) features being removed as the project progresses
Background (IV)

• 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
Background (V)

• 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
Who is the customer?

• The person or persons playing the role of the customer can vary across development contexts

  • This is very important because sometimes the answer will be hard to pin down

    • Consider a case where you are asked to develop a website for a small business

      • The owner of the small business is clearly the customer at first

        • he/she is providing requirements and paying for the work

      • But when the website is deployed, who becomes the customer?

        • The customers of the small business
Customer == User

- HCI and CSCW research shows that systems live or die by how happy the “end users” are with the system
  - The customers of the small business in this case are the end users
    - However, in the initial development project, we will only have access to the owners of the small business and we’ll have to go by what they say
    - In the future, they will be hearing from their customers about the utility and usability of our website and they will convey that feedback to us
  - What’s the difference between utility and usability?
Other Types of Customers

• You (!)
  • Often for only small scale software

• CTOs
  • Acquiring enterprise level systems for an organization
  • Who are the end users in this situation?

• New Application Development (be it desktop, web, mobile)
  • For version one: development team
    • How can you avoid this? Who are the end users?
Customer Team

• Our book addresses this concern in a short section in Chapter 1 (and will go into more detail in Chapter 5)

• The Customer Team
  • If you can’t have a user sit on your development team then assemble a team of employees whose job is to represent the customer
    • testers, product managers, usability and interaction designers
    • these people work hard to understand the customer and represent their needs as best as possible in each iteration

• **Important** because: The **highest priority** of an agile life cycle is **meeting a customer’s needs** via early and frequent delivery of working software
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 use case?

  • 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
It depends

• Is your customer a distributed systems researcher?
  • Then, yes, this may 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 our jobs website
  • 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?
• Two places
  • in the conversations around a user story; we will converge on details
  • more users stories!
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)
Confirmation

• 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!
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
Iteration is important because requirements change.
With iteration a project can make course corrections as requirements change so that what’s delivered matches what’s needed.
Benefits

• Our book ends Chapter 1 with a short list of benefits for user stories
  • 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

• What do you think (so far)? Anyone have experience using user stories?
Summary

• User stories are short statements of customer-valued functionality
  • The story serves as a visible reminder about a particular feature
  • While that reminder is important, it is not as important as the conversations around the story
    • it is the conversation that helps us track details and understand what the customer wants
    • this conversation is supported and tracked by tests that can be executed and tell us how close we are to being done
• User stories can drive software development via the concepts of releases and iterations
  • these are formed by assigning estimates and priorities to individual stories and by determining an iteration length and the team’s velocity
Coming Up Next

• Lecture 6: User Stories, Part 2

• Lecture 7: Concurrency: Threads and Locks

• Homework 2 Due on Thursday