Introduction to User Stories

CSCI 5828: Foundations of Software Engineering
Lecture 14 — 10/08/2015

© Kenneth M. Anderson, 2015

Goals

- Present an introduction to the topic of user stories
  - concepts and terminology
  - benefits and limitations
  - writing styles
  - examples

© Kenneth M. Anderson, 2015

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
    ISBN-13: 978-0-321-20568-1
- 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)

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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"

© Kenneth M. Anderson, 2015

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 (as we discussed back in Lecture 7)
  - 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*

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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
        - (Note: tasks are discussed in more detail in Lecture 15)

© Kenneth M. Anderson, 2015

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 **WILL change** while the user story remains the same)

© Kenneth M. Anderson, 2015

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!

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

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

© Kenneth M. Anderson, 2015

Writing Styles

- Writing user stories is like any form of writing
  - there are good ways to do it and there are bad ways to do it
- If you adopt the wrong writing style, you can create horrible user stories
  - If you end up not liking user stories, it might be because of the style you used and not the technique itself!
- This was true of a technique called use cases that are sometimes used in object-oriented analysis and design
  - let's see an example

© Kenneth M. Anderson, 2014

Bad Use Case

- Access User Profile
  1. If user is on any page other than home page, user clicks on Home link
  2. On home page, user clicks on Account link
  3. System responds by fading in a dialog asking for username and password
  4. User clicks on Name field; User enters name; User hits the tab key
  5. User types password; User hits the enter key
  6. If the name/password combo is correct, system displays profile page
  7. If the name/password combo is wrong, system reveals a hidden HTML element that tells them to try again

© Kenneth M. Anderson, 2014

Better Use Case

- Access User Profile
  - Preconditions
    - User is viewing home page and is currently unauthenticated
  - Steps
    1. User makes request to view their profile
    2. User provides credentials
       2.1. Credentials are incorrect
          2.1.1. System informs user; use case ends
    3. System displays user profile

© Kenneth M. Anderson, 2014

Discussion (I)

- The "bad" use case would be horrible to work with on a daily basis
  - It's brittle
  - It's unstructured
  - It mixes conditional logic into steps in an awkward way
    - making the success path unclear (imagine if it had more conditionals!)
- The "better" use case provides a description of what functionality is needed without specifying how that functionality should be implemented
  - It clearly specifies preconditions and makes success path clear
  - It is resilient to changes in the actual prototype

© Kenneth M. Anderson, 2014

Discussion (II)

- When people tell me they “hate” use cases, which type of use case do you think they were working with?

- This underscores the need to understand the writing style that goes into the documents that you create in a software development process

INVEST

- INVEST is an acronym for the user story writing style
- User stories should be
  - Independent
  - Negotiable
  - Valuable to users and customers
  - Estimatable
  - Small
  - Testable

© Kenneth M. Anderson, 2014

Independent

- User stories should be self contained pieces of functionality
  - They should not be dependent on other user stories
- The main reason is that dependencies introduce problems with planning
  - A high priority story may depend on a low priority story
    - We're supposed to work on high priority stories first!
    - May require customer education
- Dependencies can also impact our ability to make estimates
  - Will the work on the first story mean that the second story can be completed more quickly?
- Find ways to combine the stories so the result is self contained

© Kenneth M. Anderson, 2014

Negotiable

- The textual details of a story should not become “set in stone” contracts
  - The story is a reminder to have a conversation about a particular feature
    - You want the feature to be described briefly
      - You can add notes about issues that need to be resolved via conversation
      - Details that become resolved should be turned into tests

© Kenneth M. Anderson, 2014

Example

- Compare
  - A company can pay for a job posting with a credit card
    - Note: Accept Visa, MasterCard. Consider other cards
- With
  - A company can pay for a job posting with a credit card. Note: Accept Visa, MasterCard, and American Express. Consider Discover. On purchases over $100, ask for CCV on back of card. The system can tell what type of card it is from first two digits of card number. The system can store a card number for future use. Collect the expiration date for the card.
- Writing style matters! All of the details in the second example can change. They clutter the card and obscure the intent. They hide what should be other user stories in among the details.

© Kenneth M. Anderson, 2014

Valuable

- Harks back to our discussion in Lecture 5 about customer and user
  - There may be a distinction between these roles; if so, you will write user stories that address both roles
    - User: A user can archive closed bug reports.
    - CTO: Updates to the bug tracking system will automatically be detected and applied by all machines used by employees
- The main point is that each user story should describe functionality that the end users and the customer find valuable
  - Avoid stories that are only interesting to engineers on the development team
    - “All connections to the database are through a connection pool”

© Kenneth M. Anderson, 2014

Estimatable (I)

- Write stories such that they can be given a solid estimate
  - (And, yes, at the beginning all estimates are guesses while the team gets used to the “points system” and tries to determine their “velocity”)
- The main issues that prevent a team from making good estimates are
  - The story requires domain knowledge that the developers do not have
    - Before determining exposure age, an accumulation rate should be calculated on a calibration set
  - The story requires technical skills not known by the developers
    - The amount of new data generated by the system is expected to be terabytes per day; The team has no expertise in “big data”
  - The story is too big
    - The system will process all of the companies financial transactions

© Kenneth M. Anderson, 2014

Estimatable (II)

- How to address?
  - Lack of Domain Knowledge
    - (Lots of) Conversations with the customer and end users
  - Lack of Technical Knowledge
    - Technology Spikes (Goal is not to develop software; Goal is to learn)
      - Handled by treating the spike as a story; allows integration into the planning process
  - Story is too big
    - Conversations with customer/end users to identify smaller stories
      - Story needs to be decomposed until the parts can be assigned solid estimates; problem: don't go too far in the other direction!

© Kenneth M. Anderson, 2014

Small

- The amount of functionality in a user story needs to be “just right”
    - Too small and it’s not worth the time it takes to create the story and insert it into the planning process
- Epic stories are too big
    - They often describe the purpose of the entire system
        - They still can be useful if you need to describe a major class of functionality that does not need to be implemented first
- Epics typically fall into two categories
    - Compound stories and Complex stories
        - For the former, you decompose until you can make estimates
        - For the latter, you identify domain spikes that let you learn what you need to make progress

© Kenneth M. Anderson, 2014

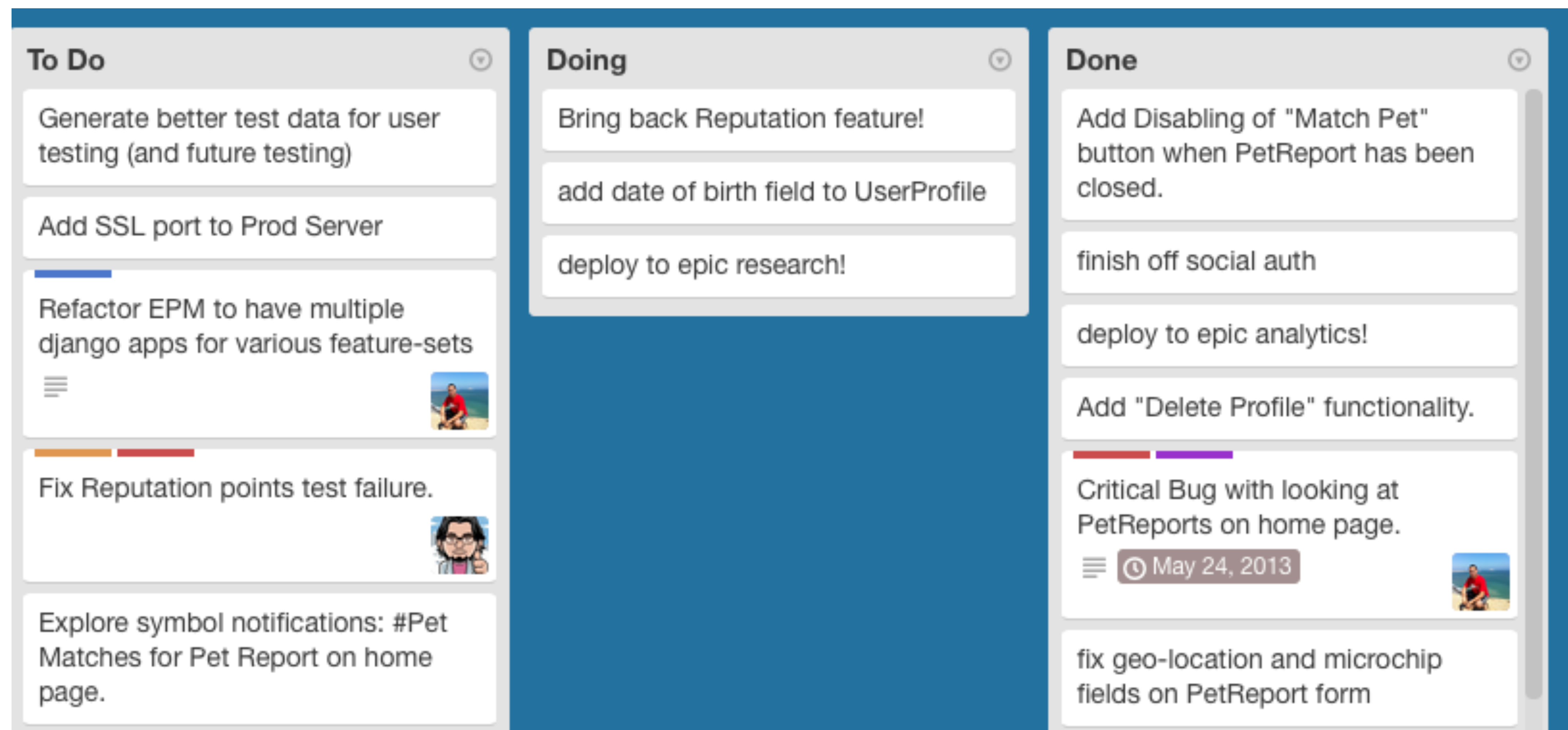
Testable

- We need to generate stories that can be tested
  - The details of a story are documented by tests
  - A story is considered finished when all of its tests finally pass
- A story that is not testable, would wreck our ability to make progress
- These problems sometimes arise with non-functional requirements
  - Avoid ambiguous terms when describing non-functional requirements
    - The system will display all pages quickly vs. The system will display all pages in under a second
  - Avoid non-functional stories that cannot be automated
    - Novice users can use the system without training

© Kenneth M. Anderson, 2014

Discussion

- These style guidelines are just that: guidelines
- In truth, you can write anything you want for your user story



Here's a Trello board with a mix of stories and tasks

© Kenneth M. Anderson, 2014

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

© Kenneth M. Anderson, 2015