User Stories, Part 3

CSCI 5828: Foundations of Software Engineering
Lecture 09 — 09/23/2014

Goals

- Continue our introduction to the topic of user stories
    - Gathering Stories
    - The Customer Team

Gathering Stories

- If you are going to drive your software life cycle via the use of user stories
    - then you need to be good at
        - getting your users, customer, and/or customer team at generating them
        - talking with them to generate ideas, clarify ambiguity, and provide details
- This is an age-old process in software development known as
    - requirements elicitation
- which is part of a larger process known as requirements engineering

IEEE Definition of a Requirement

- In contrast to user stories, traditional software requirements are defined by the IEEE as
  - A condition or capability needed by a user to solve a problem or achieve an objective
  - A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents
  - A documented representation of a condition or capability as above
- This is not too far off of the definition of a user story
  - “A user story describes functionality that will be valuable to the user and/or customer”

Requirements Engineering

- “The systematic process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained”
  - —Klaus Pohl, 1993
  - Pohl’s current work can be seen here:
    - <http://requirements-book.com/index.html>

Questions to Consider

- Requirements Engineering asks
  - Can one be systematic in the face of vaguely understood requirements?
  - Can one know whether the requirements are complete in the context of iteration?
  - How do you define cooperation among the stakeholders of a system?
  - What representations can be used to document requirements?
  - How can a genuine shared understanding be reached?

- Agile software life cycles and user stories represent one answer to these questions

The Two Sides of Requirements Engineering

- Requirements Elicitation
  - The process whereby a software development team discovers what its customers/users need and why
    - Makes use of knowledge elicitation techniques from various fields
      - ethnomethodology, human factors, ergonomics, etc.
- Requirements Analysis
  - The process of understanding the collected requirements
  - Elevates concerns of completeness and consistency
  - Makes use of formal methods of systems analysis
    - e.g. object-oriented analysis and design

Requirements Elicitation Via User Stories

- Our book focuses on the process of elicitation in the context of agile life cycles and the use of user stories
    - The book identifies the following important background concepts
        - Don’t think of requirements “capture”
            - It implies that requirements exist “somewhere out there” and all we need to do is find them
        - Don’t assume that your user knows all of the requirements and all you have to do is “elicit” them
            - The system you are building is still ill-defined for the user/customer; their understanding of what it is and what it does will evolve alongside the developers

Trawling for Requirements (I)

- Our book supports a metaphor of “trawling” for requirements
  - Trawling is the technique of pulling a net behind a boat in an attempt to catch fish
    - To be successful at trawling, you need
      - the right size net
      - know when/where to place your boat to get what you what

Trawling for Requirements (II)

- The trawling metaphor is useful because it conveys the notion of
  - granularity => there are small, medium, and large requirements
  - life cycle => like fish, requirements will mature and eventually die
    - that is the need for requirements change over time
    - a missed requirement may be due to the requirement not being important (right now)
  - triage => our net will capture both good and bad requirements, we will need to identify which ones are worth our time
  - skill => just as in trawling for fish, skill plays a role in how successful you are; what techniques should you use and when?

How many requirements up front?

- A traditional software life cycle approach to requirements is easily spotted
    - It wants you to “capture” ALL of the requirements up front
    - You then base a design on this complete understanding
    - Finally, you implement your comprehensive design and voilá you’re done!
- With agile, we recognize from the start that this is impossible
    - There is too much that is not known about the system up front
- Our book recommends identifying user stories for the first release
    - If you can be detailed, great
    - If you can’t, just make note of potential requirements that you’ll revisit later

Techniques

- User interviews
- Observation
- Story-Writing Workshops

User Interviews (I)

- Someone on the development team must
  - interview and/or hold conversations with the end users and the customer multiple times throughout the software life cycle
- Hopefully the whole team engages in this process (at least at the start)
  - because the user's application domain will be complex and unfamiliar
- Initially, the team should do most of the listening
  - let the user present the domain, the problems they encounter, the problems they want solved, and their initial ideas about the system
- Then, the team should lead the users with lots of questions that iterate over all parts of the potential system, extracting detail and priorities

User Interviews (II)

- At a minimum, you should interview
  - the customer
  - and multiple end users (at least one for each user role)
- Do **NOT** let them present you with solutions
  - You want to know “what” the problem is, not “how” to solve it

User Interviews (III)

- Finally, be careful with the types of questions you ask
  - You can bias the result
    - Would you like a free pony? vs. Would you like a free pony that will require you to get up at 4 AM each day to take care of it before work?
- Close-Ended Questions
  - Typically contain the answers in the question or constrain the types of answers too narrowly
    - Would you like the system to generate reports in HTML format?
  - Better to ask
    - What type of reports do you work with now? How do you receive them?

User Interviews (IV)

- Open Ended Questions and Context Free Questions
  - Open ended questions: do not constrain the way a user can answer
  - Context-Free questions: do not provide the answer in the question
- The last two questions on the previous slide get to an important issue but do not constrain the types of answers a user generates
  - The answers provided might be very detailed and contain lots of ideas for user stories
- Example
  - Is performance important in some parts of the application?
  - Let the users tell you where performance is a concern

User Interviews (V)

- Reminder: these recommendations are for elicitation
- If you have specific user stories that you are trying to implement
  - then ask for details and provide context (as appropriate) to get the information you need to make progress

Observation (I)

- If you can “shadow” an end user and watch them work with
  - a previous version of a system that you are tasked with upgrading
  - their current set of tools for which you are being asked to develop a system to replace
- then do so. You will discover the user’s work environment is rich with details that
  - can be converted into user stories
  - notated on existing stories
  - resolve ambiguities from previous conversations
  - reveal bottlenecks in workflows that may be “invisible” to the user

Observation (II)

- If you have a prototype for the user, you can also use observation to
  - identify usability problems
  - see where the current system falls short in meeting customer expectations
  - clarify ambiguous requirements
  - reveal the need for other features
- Similar to usability testing but not exactly the same
  - usability testing is often performed with a user by themselves and developers observing the user in another room
  - the goal there is to gain insight strictly into usability concerns
- With elicitation, you will typically be sitting right next to the user, holding a conversation with them, while they use the prototype

Story-Writing Workshops (I)

- Discussed briefly in Lecture 5
- A story-writing workshop is held at the start of each release
    - The goal is to brainstorm as many user stories as possible
        - No estimates are generated and no priorities are assigned
        - There are no “wrong” stories during the initial brainstorm
    - If you’re having problems getting started, then engage in user role modeling, as discussed in Lecture 6
- Low-Fidelity prototypes can be very useful
    - used to identify the high-level conceptual workflows of the system
    - no attempt is made to design how a screen would look

Story-Writing Workshops (II)

- The prototype can be used to then “walk through” the workflows and ask questions such as
  - What will the user do next?
  - What mistakes might the user make at this step?
  - What might confuse the user at this point?
  - What additional information would be useful to the user at this point?
- Book provides an example of a low-fidelity prototype on page 50
  - The prototype uses one index card for each of the major functional areas of the system
    - User stories can be derived after discussing each of these areas and the ways in which users access them

Questionnaires?

- What about using questionnaires to find user stories for a system?
  - This might be a reasonable thought when the existing user base is
    - a) large
    - b) reachable (i.e. via an e-mail distribution list)
- Questionnaires may be useful to determine information that can be used to prioritize some set of user stories over another
  - “The most hated problem with the previous version of the system was…”
- But it is not an effective technique to find new user stories. The format will either be too constraining (a list of possible answers) or difficult to process (free form text from hundreds of responses)
- Our book, thus, does not recommend using questionnaires for elicitation

The Customer Team (I)

- Or as our book put it “Working with User Proxies”

- The basic idea is that
  - Agile software life cycles tell us that “the end user” must be a member of the development team
  - The reality ⇒ This is really hard to do!
    - For a brand new product, the system does not have real users (just targeted users)
    - For an existing system, a user who is sitting with you is being pulled away from their job ⇒ their manager may not allow them to do this!

The Customer Team (II)

- As a result, we must identify people
    - who can be members of the development team (i.e. employees)
    - but whose job it is to represent the end user
- There are many different types of people who can play this role
    - the book reviews one class of stakeholders
        - managers, analysts, marketing types, etc.
    - but ignores another important class of stakeholders
        - employees who are trained to care about the user!
            - usability engineers, user experience designers, testers, etc.

The Book’s User Proxies (I)

- The User’s Manager
    - may be the person preventing you from talking to the real users!
    - will have different usage patterns and needs for the software system
    - but, may also be “as close as you can get”, you can at least ask them to ask their employees for the answers to questions you have
    - watch out for “tall tales”, details that are inflated for effect but do not accurately reflect the user’s needs or issues
- The Manager of the Development Team
    - This happens only because of circumstance not due to anyone thinking this person would make a good user proxy
    - Their stories will be influenced by what they want achieved with this project and not necessarily what’s best for the users

The Book's User Proxies (II)

- Salespeople
  - will likely have a deeper understanding of the application domain
  - but will also likely prioritize features that will help them make sales rather than address all of a user's needs
  - often can serve as a good conduit to actual users; use them to get your foot in the door!
- Doman Experts
  - a.k.a. subject matter experts
  - great for domain modeling and identifying business rules
    - but, likely, bad at representing users of the system you're trying to build
      - they will miss details related to workflow and what information needs to be available to support the current task

The Book’s User Proxies (III)

- Marketing Groups
  - They understand markets not users
  - May be able to help prioritize certain features over others
  - Will likely focus on the quantity of features rather than quality
- Former Users (of previous versions of the system)
  - Will likely be very useful to get a good understanding of the domain and the basic features needed
  - Will need to confirm that their experience still relates to the experience of current users, however
- Customers
  - The purchaser of the system; may be a CTO or CFO in the enterprise context; will desire features that relate to deployment/management

The Book's User Proxies (IV)

- Trainers and Technical Support
  - These people support actual users either to learn the system or to deal with problems that occur when using the system
  - They may know very little about the actual domain; instead they are experts on the previous version of the system!
  - They will prioritize—either consciously or subconsciously—features that make their lives easier (which is why you really need to talk to your users!)
- Business or System Analysts
  - Similar to domain experts; they are experts at requirements analysis
    - they have one foot in technology and one in the target domain
    - May serve as a conduit to actual users; or be willing to talk to real users on your behalf

Other User Proxies

- The domain of human-centered computing
    - consisting primarily of HCI (human-computer interaction) and CSCW (computer-supported cooperative work)
- provides a wealth of knowledge, concepts, and techniques for making software design and development centered around the user
- Graduates from these disciplines are trained to interact and work with users
    - They LIKE getting into the details of the user’s application domain and documenting the details
    - They are trained that iteration is absolutely essential in getting to a user experience (not “how it looks” but “how it works”)
    - They are therefore ideal people to serve as user proxies
- For this to work, however, they need access to real users (at least once!)

Other Techniques

- If access to users is limited
    - try forming a volunteer “user task force”, meet with them as often as you can
- When no user is available
    - make use of multiple user proxies to gain lots of perspectives
    - review competing products in the application domain
    - get something working and release it early
        - Boom! You have real users now. 😃

Summary

- Gathering Stories
  - We have reviewed the concept of requirements elicitation and requirements engineering
  - Discussed the metaphor of “trawling for requirements” vs. capturing
  - Reviewed techniques for elicitation
    - interviews, observation, story workshops
- The Customer Team
  - reviewed the types of stakeholders that can serve as user proxies
  - identified members of the development team who can serve as proxies in a pinch
  - identified other techniques that can be used if development must proceed without access to real users

Coming Up Next

- Lecture 10: User Stories, Part Four
- Lecture 11: Concurrency in Clojure, Part One