

CS1530 LECTURE 5

Agile / Scrum



Background picture courtesy of Maree Reveley, used under Creative Commons Attribution-Share Alike 2.5 Generic license.

SOFTWARE DESIGN METHODOLOGY

- Our methodology: Agile / Scrum
 - Not the “perfect” methodology - which does not exist - but for our project, it fits well
 - Relatively lightweight
 - Flexible
 - Focus is on understanding customer needs
 - Popular in industry
 - Allows you to focus on development - doesn't get in your way

AGILE MANIFESTO

- <http://agilemanifesto.org/>
 - *Individuals and interactions **over** processes and tools*
 - *Working software **over** comprehensive documentation*
 - *Customer collaboration **over** contract negotiation*
 - *Responding to change **over** following a plan*

SCRUM

- Almost certainly the most popular agile methodology, although far from the only one...
- Extreme Programming (XP)
- Lean Software Development (LSD)
- Dynamic Systems Development Method (DSDM)
- Feature-Driven Development (FDD)
- Agile Unified Process (AgileUP)
- Crystal {Clear, Yellow, Orange, Red, Maroon}

SCRUM

- Not an acronym!
- Comes from a rugby scrum - everyone on team moving in one direction
- Teams are almost entirely self-managed
- Three roles
 - Product owner - Act as representative for the customer
 - Scrum master - They act as a “firewall” for the outside world and a centralized place to ask for help / facilitate meetings / etc.
 - Team - Everyone else: QA, developers, etc.

SCRUM

- Product-focused, end-user focused
 - Transparency - Work should be visible to those who need to see it
 - Inspection - Work should be examined regularly to ensure that the team is on the right path, or are doing things in a suboptimal manner
 - Adaptation - Work should be modifiable as requirements and limitations are better understood

USER STORIES

- A description in “plain language” that states what the user needs the software to do
- Related to requirements, but not exactly the same!
- Often in the *Connextra template*:
 - As a <role>
I want <feature>
So that <reason>

USER STORIES

- Examples:

- As a manager

- I want the software to display the current status of each engineer

- So that I can more effectively write status reports

- As an Engineer

- I want the ability to enter my daily status on a web page

- So that I can update my manager on my status more easily

- As a user of Excel

- I want a keyboard shortcut to select text

- So that I can quickly grab text without spending extra time reaching for my mouse

USER STORIES

- Allows us to not only see what they want, but more importantly, *why*
- Gives us further flexibility if what they say they want is difficult/impossible, but can do something else that gives them the same result, or if there is a better way to achieve that objective

PRODUCT BACKLOG

- List of all items to be done
- In the beginning, should be all user stories
- Should be prioritized
- Differs from a Software Requirements Specification in that this is a living document - it will change as defects are added, user stories modified or removed, etc.
- Think of it as a kind of mixed to-do list / software specification

SPRINTS

- Software development is split into “sprints” - iterations of 2 - 3 weeks where work is done from the backlog (our sprints will be two weeks)
- At the beginning of the sprint, there is a sprint planning session where it's determined which user stories will be worked on and who will work on which ones
- These are not set in stone! Some may run over or you may work on extra. We will *estimate* how much can get done during our sprint planning sessions.
- This session is facilitated by the Scrum Master

ESTIMATING – USER STORIES AND STORY POINTS

- Note that this is one method for estimating effort
- For each user story, allocate story points.
- Story points can be powers of two - 1, 2, 4, 8, or 16 points
- To start:
 - 16 points - a story which would take one developer the entire sprint to do
 - 1 point - Trivial change (fixing typo, off-by-one error, etc.)
- Why powers of two? We're horrible at estimating software development time.
 - “Better to approximately right than exactly wrong”

VELOCITY

- Definition - how many story points got done last sprint?
 - NOT how many were planned, how many were completed
- Example - You have four devs on a team. You plan to complete $(4 * 16)$ or 64 story points' worth of stories. You actually complete 32. Your velocity is 32.
- Now you can better estimate / calibrate the number of story points to shoot for next time, as well as your estimating ability - maybe you've been underestimating the amount of work necessary for stories, or overestimating how much time you'd be able to dedicate to development and not external meetings, etc.

SPRINTS

- At every point, and ESPECIALLY at the end of the sprint, you should have WORKING software
- It does not need to be feature-complete, but compiles, runs, etc.
- Adding a feature means it has met “the definition of done”
 - Code
 - Documentation
 - Integration
 - Testing

STANDUPS

- Standups - usually daily and very short communications with the rest of the team during the sprint
 - What have I done in the last 24 hours?
 - What do I plan to do in the next 24 hours?
 - Do I need any help or have any blockers?
- You can probably do this 3x/week, probably not necessary for every day
- However, this is up to you
- Facilitated by scrum master

RETROSPECTIVES

- At the end of each sprint, the team comes together to discuss:
 - What went well?
 - What could go better?
 - What can we do different in the next sprint?
- Once again facilitated by the scrum master

SPRINTS, STANDUP AND RETROSPECTIVES

- For our class, every other Thursday (end of sprint), we will meet in class and have:
 - Short retrospective on previous sprint
 - Short sprint planning for next sprint
 - You may decide to do more outside of class
- Scrum Master position will change each sprint (different person each sprint).
 - Old scrum master handles retrospective; new scrum master handles sprint planning

KANBAN BOARD

- A board or software which shows status of user stories and other items (research, defects, etc.)
- At a glance, can show status of all items to be worked on during this sprint
- There should be a small number of possible statuses.

Example:

- Planned
- In progress
- In test
- Completed

KAIZEN

- Japanese for “continuous improvement”
- You should work to not only think “how can the software be made better?” but also “how can the process be made better?”
- Explicit discussion of this in retrospectives
- Examples:
 - We spend lots of time getting authentication working. Could a third-party provider do this better?
 - Our stand-ups are taking up too much time. Could we limit people talking to one minute?
 - I never know where to find information on the product. Can we put together a wiki page?