CS189A Capstone
Fall 2023

Lecture 3: Scrum, Sprints, Getting Started, Product Requirements Document

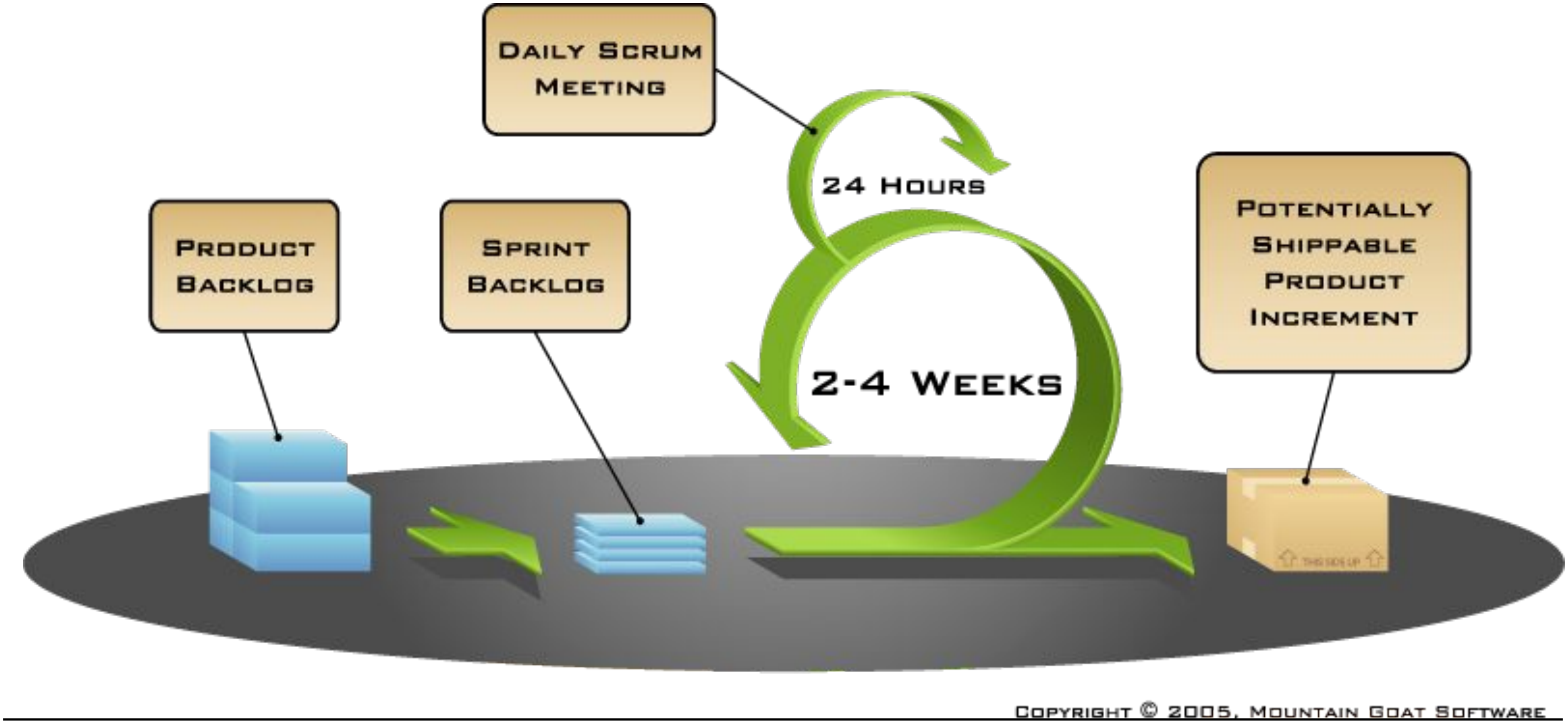
CS189A Schedule

Week 1 (Oct. 2)	project pitch meeting, team formation, project selection	
Week 2 (Oct. 9)	contact mentors, select team lead and scribe	sprint 1 starts
Week 3 (Oct. 16)	vision statement due on Oct. 17	
Week 4 (Oct. 23)		sprint 1 ends, sprint 2 starts
Week 5 (Oct. 30)	Requirements and Design PRD version 1 due Oct. 31	
Week 6 (Nov. 6)		sprint 2 ends, sprint 3 starts
Week 7 (Nov. 13)	Requirements and Design PRD version 2 due Nov. 14	
Week 8 (Nov. 20)		sprint 3 ends, sprint 4 starts
Week 9 (Nov. 27)		
Week 10 (Dec. 4)	Final presentations and demos	sprint 4 ends

Agile Software Development with Scrum

Scrum

- An evolutionary/iterative/incremental/agile software process implementation
  - See: **Scrum and XP from the Trenches** -- free online book by Kniberg



Copyright © 2005, Mountain Goat Software

Scrum

- An evolutionary/iterative/incremental/agile software process implementation
  - See: **Scrum and XP from the Trenches** -- free online book by Kniberg
- The main roles in Scrum are:
  - Scrum team: Team of software developers
  - Scrum master : Project manager
  - Product owner: Client
- Characteristics of Scrum:
  - Self-organizing teams
  - Product development in **two** to four week **sprints**
  - Requirements are captured as items in a list of **product backlog**
    - **Yours will come from your requirements document (PRD)**
- Homework: read the links on webpage under today's date

Product Backlog (Scrum Artifacts)

- An ordered list of everything known to be needed:
  - These are the requirements – in your requirements document (PRD)
  - A list of all desired work on the project
  - features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases
- Prioritized by the product owner
  - Reprioritized at the start of each sprint
- Each backlog item also has an estimated time it will take to complete it
  - Sum of tasks that make up an item (story, use case) should be the total
- Product Backlog is never complete

Sprint and Scrum: Implementation

- Sprint planning
  - Use cases or user stories broken down into tasks (from prioritized product backlog) -> sprint backlog
    - Team members assign timings (how long each will take)
    - And pick tasks
  - Tasks = designing, implementing, testing, and demo'ing
    - Includes code review with a second member (demo individual task)
  - Any new tasks identified put onto backlog for inclusion next time
- Daily standup
  - What I did yesterday, what I'm doing today, + any blockers
  - Short/quick so done standing up!
- At end of each iteration/sprint
  - Sprint review: demo each story and end-to-end system to team
  - Retrospective and end of each iteration– what worked and what didn't
    - Vote on what to fix in the next sprint

Scrum Roles

- Product owner (mentor + team in our case)
  - Defines the features of the product
  - Decides on release date and content
  - Prioritize features according to market value
  - Adjust features and priority every iteration as needed
  - Accepts or rejects work results

- Scrum Master (team lead in our case)
  - Represents management of the project
  - Responsible for following the Scrum process
  - Ensures that the team is fully functional and productive
  - Shields the team from external influences

Scrum Roles

- Scrum Team
  - Typically 5 to 9 people
  - Cross-functional team that does the software development including designing, programming and testing
  - Co-location and verbal communication among team members
  - Teams are self-organizing, no titles
  - Team membership should not change during a sprint

Scrum Meetings

- Sprint Planning
  - This is done at the beginning of every sprint cycle (2 to 4 weeks)
  - Team selects items from the product backlog they can commit to completing
  - Sprint backlog is created
    - Tasks for this sprint are identified and each is estimated (hours, points, partial days). This is done collaboratively, **not** by Scrum Master
  - High-level design is discussed

- Daily Scrum (at most 15 minutes)
  - Daily, stand-up meeting
  - Not for problem solving
  - Every team member answers three questions:
    - What did you do yesterday?
    - What will you do today?
    - Is anything in your way? (Scrum Master is responsible for following up and resolving the impediments)

Scrum Meetings

- Sprint Review
  - Team presents what it accomplished during the sprint
    - Typically **a demo of new features or underlying architecture**
    - Incomplete work should not be demonstrated
  - Informal meeting, no slides
  - Whole team participates
  - Open to everybody

Scrum Meetings

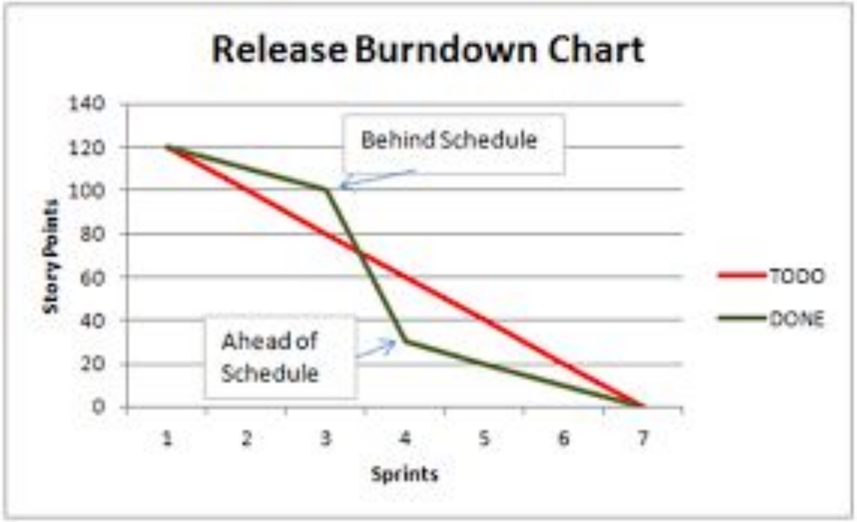
- Sprint Retrospective
  - Periodically take a look at what is and is not working
  - Done after every sprint
  - Scrum Master, Product owner, Team and possibly customers and others can participate
  - One way of doing sprint retrospective is to ask everyone what they would like to

1) Start doing, 2) Stop doing, 3) Continue doing

Or 1) What worked, 2) What didn’t, 3) What should change

Scrum Artifacts

- Sprint Backlog
  - Team members sign up for work (break stories into tasks) of their own choosing
  - Estimated work remaining is updated daily
  - Any team member can add, delete or change the sprint backlog
  - Each sprint backlog item has **daily estimates** for the amount of time that will be spent on that item each day
- Burn Down Chart
  - A daily updated chart displaying the remaining cumulative work on the sprint backlog. It gives a simple view of the sprint progress
- Many tools on the web to track sprint
  - Google Worksheet is easiest
  - Backlogs, burndown
  - Trello, PivotalTracker



Plan for Sprint 1

- Overview of project
- Sprint planning (using Trello): Backlog, on deck, in progress, done
  - Break up into tasks **with durations** (hours, part/days, points)
    - Identify/record initial requirements
    - Start writing sections of the PRD v1
    - Identify/install/test tools & technologies
    - Sketch out design and start listing terminology
    - Coding should be done for establishing basic use of technologies
      - Must become github commits this week
  - Assign 2+ members to each (implementer and tester/reviewer)
    - Fill 9 days according to durations for each member
    - Order tasks by priority (top = highest): top total 9 days * 5 members
  - Any new tasks identified put onto bottom of backlog for next time
- Setup burndown using google worksheet (shared w/ all)
- **Daily standup/scum** (start tomorrow (scribe records in google doc))

More on Scrum

- More information about Scrum process is available at:
  - www.mountaingoatsoftware.com/scrum
  - www.scrumalliance.org
  - www.controlchaos.com

- Required reading
  - Scrum - A Breathtakingly Brief and Agile Introduction (free for class use only)

Getting Started with the Project

Technologies to Consider + Ask Mentors

- Work on **tutorials** if new to you
- To support workflow (required in red)
  - Trello or PivotalTracker, Podio, Jira
  - Github
  - Issue tracking (github, Jira)
- Fast prototyping
  - firebase, react/react-native, angular vs bootstrap
- Continuous builds
  - Jenkins, travis
- Wireframes
  - gomockingbird (mockingbird), balsamiq
- Useful components/technologies
  - Oauth
- Mobile app platforms
- IDEs, programming languages

- Server and cloud:
  - System configuration: Ansible, Puppet, Chef, Saltstack/Saltcloud
  - Containers: Docker/kubernetes
  - Virtual servers/object store: AWS, Google, Azure
    - Use **free tier** & student credits
  - Platforms: Google App Engine, Heroku
  - Mobile Backends: Backendless, Google Endpoints, AWS Lambda
  - Services: MongoLab, Instacluster, Amazon RDS, Firebase
  - Hadoop/ElasticMapReduce, Spark
  - APIs: Twitter, Facebook, Google technologies (maps/earth/drive)
  - Multicloud/Java: Apache Jclouds
  - Local Linux server/DB: ask instructor/TA

Getting Started

- Setup public repository (GitHub: github.com or github.ucsb.edu)
  - Identify workflow: https://www.atlassian.com/git/workflows
    - Suggested: feature branch, gitflow
    - Git branching basics
- Setup an issue tracker – GitHub, Jira, other…
- UML design tools:
  - Draw your UML diagrams onine (no SW installation necessary!): http://yuml.me/
  - http://www.visual-paradigm.com/solution/freeumldesigntool
- Understand/learn about writing user stories
  - http://www.mountaingoatsoftware.com/agile/user-stories, http://www.romanpichler.com/blog/user-stories/writing-good-user-stories/, http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/, https://help.rallydev.com/writing-great-user-story

Getting Started

- Investigate language, frameworks, install necessary software
  - Debuggers, IDE, tools, **TDD**
    - Firebug: http://www.w3resource.com/web-development-tools/firebug-tutorials.php
    - Web site testing: **Selenium, Watir** Chrome DevTools, javascript debugging tools
  - Python: unittest, py.test, nose, pymock, http://wiki.python.org/moin/PythonTestingToolsTaxonomy
  - Ruby: http://guides.rubyonrails.org/testing.html
  - C++: http://praveen.kumar.in/2010/04/30/introduction-to-test-driven-development-in-c-using-boost-test-library/, http://vimeo.com/13240481
  - Java: http://vimeo.com/10569751, http://www.javabeat.net/2011/04/test-driven-development-in-java/

- Investigate mock up tools: flexmock, wireframing/mock ups
  - http://www.justinmind.com/, http://mashable.com/2010/07/15/wireframing-tools/, Balsamiq

- Code coverage/metrics
  - http://c2.com/cgi/wiki?CodeCoverageTools
  - Java: http://emma.sourceforge.net/, http://cobertura.sourceforge.net/
  - Ruby: https://www.ruby-toolbox.com/categories/code_metrics
  - Python: https://coverage.readthedocs.io/en/v4.5.x/

Getting Started: Recommendations

- Set up agile process for sprint plan
  - Story writing & tasks (Trello),
    - Alternatively**: PivotalTracker** is free for public projects
    - Project board with stories (and perhaps story breakdown via tasks)
  - Trello board per sprint (Backlog, on deck, in progress, done)
    - Each card is a task with a duration/timing and 2+team members (one for implementation, one for testing/followup/review)
      - Copy tasks from story board
  - Burndown chart per sprint (google drive worksheet)
- Set up continuous build process: Travis CI or Jenkins
- Setup container system (Docker) on everyone's laptop
- $100 reimbursable for tools/clouds/services you really need. Speak to mentor/Instructor if you need more (before you spend!)
  - Turn receipts into CS financial office, tell them to contact Instructor for approval and to charge the CS Capstone grant

Use Cases and User Stories

- Use cases:
  - http://en.wikipedia.org/wiki/Use_case
  - http://en.wikipedia.org/wiki/Use-case_analysis
  - http://alistair.cockburn.us/get/2465
  - ftp://ftp.software.ibm.com/software/rational/web/whitepapers/RAW14023-USEN-00.pdf
  - http://www.gatherspace.com/static/use_case_example.html
- User stories:
  - http://en.wikipedia.org/wiki/User_story
  - http://capstone.cs.ucsb.edu/cs189a/support/DanNorth-Stories.pdf
  - http://capstone.cs.ucsb.edu/cs189a/support/AgileModeling-Stories.pdf
  - http://www.romanpichler.com/blog/10-tips-writing-good-user-stories/
  - http://www.mountaingoatsoftware.com/agile/user-stories
  - http://www.romanpichler.com/blog/user-stories/writing-good-user-stories/
  - http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/
  - https://help.rallydev.com/writing-great-user-story

Capstone Project Requirements (1/2)

- Use of agile development process with per-sprint task tracking (recommended: Trello or PivotalTracker)
- Daily scrums recorded by scribe in shared Google Doc
  - Class/discussion days: last 15mins of class
  - Shared with Instructor, Mentor, TA, and team
- Weekly meetings (virtual is ok) with mentor
- Weekly meeting with TA
- Class/discussion attendance and participation in team activities
  - Bring laptop to class
- Vision statement turned in by deadline (& approved by mentor)
- Draft 1 and 2 of requirements specification turned in by deadlines
  - Evolve as you design and prototype; approved by mentor
- Working prototype for base functionality demonstrated in the last week of the quarter

Capstone Project Requirements (2/2)

- Use of a code repository (recommended: GitHub)
  - Ongoing contributions by all members throughout
    - Using a clear workflow
  - Can include preparation of requirements documents
- Use of an issue tracker (recommended: github)
- Documented code
- Automated unit tests and integration and/or functional tests
  - Code defensively!
- Use of user stories and/or use cases for requirements and design
- Use of UML for system requirements modeling and design
- Wireframes for user interfaces if any
- Complete 4 2-week sprints, record retrospectives for each

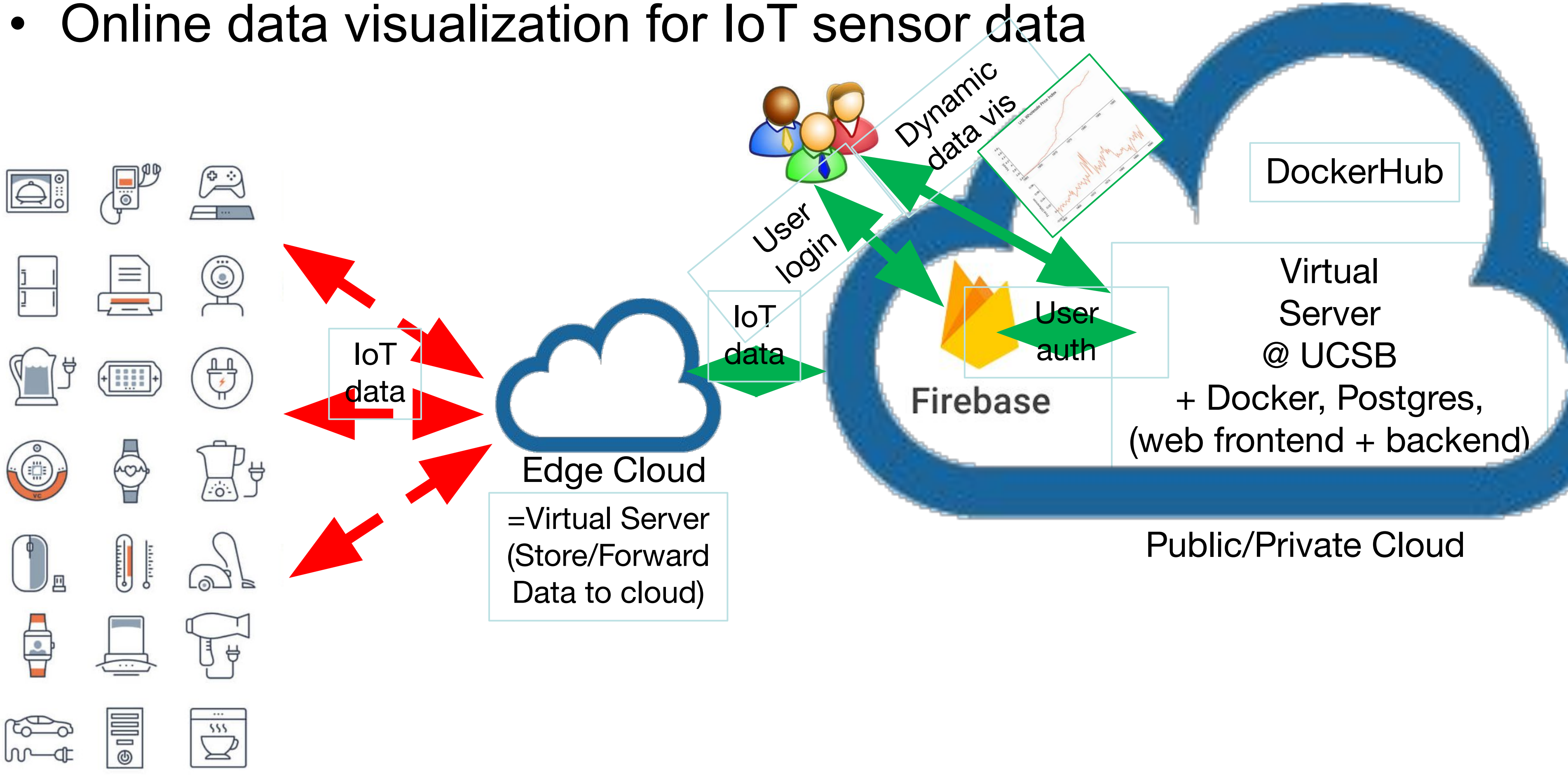
On-going Process

- Evolving (aka "living") requirements document
  - Identify/learn (and teach each other) the technologies required
  - Write user stories in particular; update the requirements as you go:
    - Prioritize stories and mark **mandatory, important, or desirable**
    - Assign time estimates to stories; improve your estimation ability over time
    - Specify **acceptance test** for each story – should be in code
- Concurrently as part of Sprint
  - Break down stories into tasks (begin design/prototyping process)
    - Prioritize tasks
    - Assign timings to tasks
    - Specify what (code) test(s) are to be used as evidence of task completion/acceptance
    - Each member/developer chooses task, implements, and tests task
    - Another member does code review/test and accepts the pull request
      - Test is the one specified above (Acceptance)
    - When a Story is complete, some member performs story test/acceptance

A Hypothetical Example Project

Example Project

- Online data visualization for IoT sensor data



IoT data

Edge Cloud

=Virtual Server (Store/Forward Data to cloud)

IoT data

User login

Dynamic data vis

Firebase

User auth

DockerHub

Virtual Server @ UCSB + Docker, Postgres, (web frontend + backend)

Public/Private Cloud

Example Project

- Online data visualization for IoT sensor data
- Technology Investigations:
  - Virtual linux server (Edge and cloud) + Docker/Dockerhub
    - Setup and configuration
  - user login support: front end and backend
    - Web front end (html, css, javascript, 3rd party/oauth?)
      - Bootstrap, material design for React w/ bootstrap
    - Backend (firebase)
      - When you should and shouldn't use firebase
      - Link bootstrap to firebase
  - Time series graphs from local data: Chart.js
  - Dynamic time series graphs
    - GraphQL + Postgresql + Chart.js
  - Connect logged in user to authenticate data access
  - Ingress data to edge cloud server
  - Forward edge data to cloud postgresql DB

Example Project

- Online data visualization for IoT sensor data
- Sprint 1 plan: (Sprint 1 is 9 days total)
  - AWS EC2 virtual server tutorial with persistent volume storage
    - **test**: start/stop server, (un-)mount volume, ssh, config Docker/ web backend (below)
  - Docker tutorial: configure/deploy server image: **test**=all team members laptops and EC2 virtual server
  - user login support: front end and backend
    - Bootstrap/React tutorial: **test**=deploy simple website, be able to change it
      - Bootstrap, material design for React w/ bootstrap
    - firebase tutorial, link to bootstrap: **test**=simple login page
    - Investigate how to share info on logged in user to remote service
  - Web page static graph: Chart.js tutorial: **test**=website with graph w/ fake data (hand coded in javascript)
  - Dynamic time series graph
    - Tutorials + Integration tutorial: GraphQL Engine + Postgresql+ Chart.js
    - **Test**: website that is dynamically updated when new data is added to db
  - PRD work (pbm, innovation, detailed core advance, background, sys arch picture, technologies list), test = review by team mate

Example Project

UC Santa Barbara

- Online data visualization for IoT sensor data
- Sprint 1 plan: (Sprint 1 = 9 days – breakdown is below in red)
  - ¼ AWS EC2 virtual server tutorial with persistent volume storage
    - **test**: start/stop server, (un-)mount volume, ssh, config Docker/ web backend (below)
  - ¼ Docker tutorial: configure/deploy server image: **test**=all team members laptops and EC2 virtual server
  - ½ user login support: front end and backend
    - Bootstrap/React tutorial: **test**=deploy simple website, be able to change it
      - Bootstrap, material design for React w/ bootstrap
    - 1 firebase tutorial, link to bootstrap: **test**=simple login page
    - 1 Investigate how to share info on logged in user to remote service
  - 1 Web page static graph: Chart.js tutorial: **test**=website with graph w/ fake data (hand coded in javascript)
  - 1 Dynamic time series graph
    - ½ Tutorials + Integration tutorial: GraphQL Engine + Postgresql+ Chart.js
    - ½ **Test**: website that is dynamically updated when new data is added to db
  - 2 PRD work (pblm ¼, innovation ¼, detailed core ½, advance, background ½, sys arch ¼, picture, technologies ¼ list), test = review by team mate
  - 2 Trello setup for Sprint 1

Example Project

- Online data visualization for IoT sensor data
- Sprint 1 backlog (will move to sprint 2)
  - Connect logged in user to authenticate data access
  - Setup/configure edge server (using docker container for portability)
  - Ingress data to edge cloud server
  - Forward edge data to cloud postgresql DB
  - Features as determined by requirements analysis: user stories and use cases (more on this next Monday)
- Add these to new Sprint 2 board (Backlog)

- Create burndown

Introduction to PRD

Product Requirements Document

- The official statement of what is required of the system developers
- Includes a specification of both user and system requirements

- Defines WHAT the system should do, *not HOW it should do it*
  - Design/Impl comes later; give engineers freedom in how to go about it

- Agile and extreme SWE processes express requirements as
  - **Use cases – how a system will act**
  - Or as scenarios called **user stories** (describe result/benefit of it)
  - **We will discuss/practice these next week**

Agile Requirements Specification

1. Define project specifics
2. Team goals and objectives
3. Background and strategic fit
4. Assumptions

Intro + System Architecture

5. User Stories or Use Cases
6. User Interaction and Design

Requirements & Sys. Models

7. Questions
8. What we’re NOT Doing

Appendices

❖ Evolve the document over time, concurrently with development

Required reading:
https://www.atlassian.com/agile/requirements

PRDv1: Your **Living** Requirements Document: A Shared Google Doc (due in ~3 weeks)

- Authors, Team, Project Title
- Intro – including problem, innovation, science, core technical advance (2-3 pages)
  - Define project specifics, team goals/objectives, background, and assumptions
- System architecture overview
  - High level diagram (1 page)
  - User interaction and design (1+ page)
- Requirements (functional and non-functional)
  - User stories or use cases (links) □ 10 for PRDv1 prioritized
  - Prototyping code, tests, metrics (5+ user stories): github commits/issues
- System models: contexts, sequences, behaviorial/UML, state
- Appendices
  - Technologies employed

Today

- Work on vision statements (turn in as PDF via email to TA by Wednesday)
- Setup trello or pivotal tracker, go through tutorial
- Review technologies and getting started pointers
- Discuss and determine github workflow (read tutorials if needed)
  - Feature branch or gitflow: https://www.atlassian.com/git/workflows
  - Git branching basics
- **Plan Sprint 1**, include PRD v1 work (See previous years' example projects)
  - Identify tutorials, technologies, configuration/deployment, prototyping (use of technologies)
    - Specify **duration**, review/testing step is to show another team member
  - PRD v1 work: intro (each member makes a pass), arch diagram, technologies
    - User stories/use cases (initial feature set) – more details on this next Monday
- Sprint 1 starting today
  - Start daily scrums (set times) (Monday and Tuesday daily scrums will be in class and discussion section)
  - Setup periodic meetings with the mentor