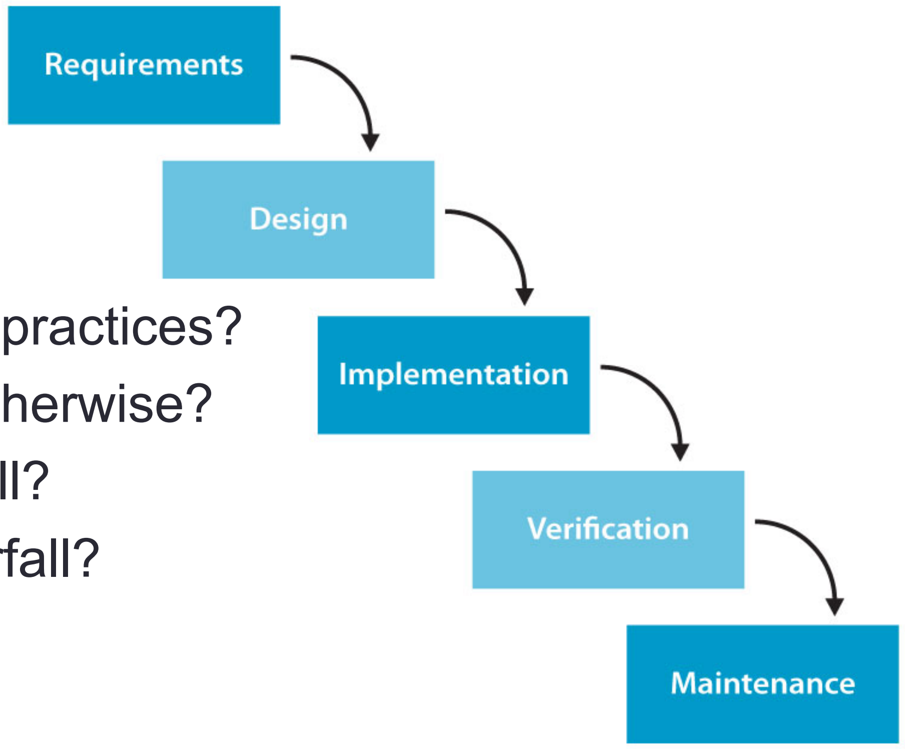
SOFTWARE ENGINEERING: AGILE DEVELOPMENT

Professors M. Brian Blake and Iman Saleh

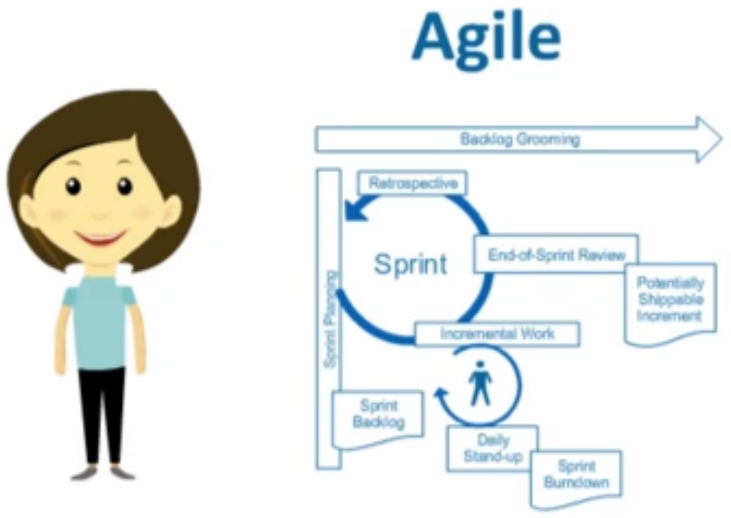
Waterfall Methods…



- How do you feel about waterfall practices?
- How would you start a project otherwise?
- What are the benefits of waterfall?
- What are the limitations of waterfall?

Introduction to Agile Development

- http://www.youtube.com/watch?v=OJflDE6OaSc



A Manifesto for Agile Software Development

- Irritated with heavyweight software engineering practices that did not seem to fit well with smaller projects with ever-evolving requirements.
- In February 2001, 17 software engineers met at Snowbird to create the "Manifesto for Agile Software Development"

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

12 Principles: Agile Software Development

- Customer satisfaction by rapid delivery of useful software
- Welcome changing requirements, even late in development
- Working software is delivered frequently (weeks rather than months)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Close, daily cooperation between business people and developers

12 Principles: Agile Software Development

- Face-to-face conversation is the best form of communication (co-location)
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Software Engineering Methods

Adaptive ⟷ Predictive

Agile	**Plan-driven home ground**
Low criticality	High criticality
Senior developers	Junior developers
Requirements change often	Requirements do not change often
Small number of developers	Large number of developers
Culture that thrives on *chaos (?!)*	Culture that demands order

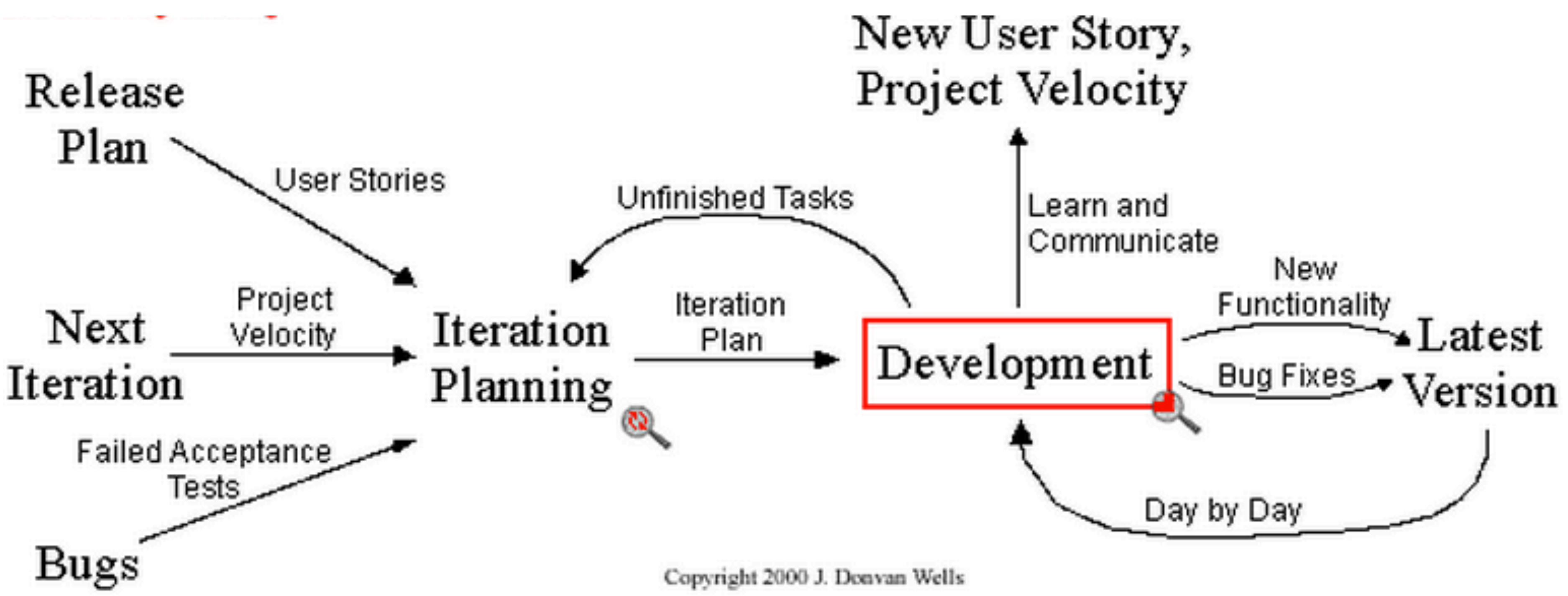
Extreme Programming (XP) One Flavor of Agile Development

- **Rapid feedback.**
  - Confronting issues early results in more time for resolving issues. This applies both to client feedback and feedback from testing.
- **Simplicity.**
  - The design should focus on the *current* requirements.
  - Simple designs are easier to understand and change than complex ones.
- **Incremental change.**
  - One change at the time instead of many concurrent changes.
  - One change at the time should be integrated with the current baseline.

XP Mantras

- Embracing change.
  - Change is inevitable and frequent in XP projects.
  - Change is normal and not an exception that needs to be avoided.
- Quality work.
  - Focus on rapid projects where progress is demonstrated frequently.
  - Each change should be implemented carefully and completely.

XP Iteration



How much planning in XP?

- In XP, planning is driven by requirements and their relative priorities.
  - Requirements are elicited by writing stories with the client.
  - Stories are high-level use cases that encompass a set of coherent features.
  - Developers then decompose each story in terms of development tasks that are needed to realize the features required by the story.
  - Developers estimate the duration of each task in terms of days.
  - If a task is planned for more than a couple of weeks, it is further decomposed into smaller tasks.

How much planning in XP?

- Team Organization
  - Production code is written in pairs.
  - Individual developers may write prototypes for experiments or proof of concepts, but not production code
  - Moreover, pairs are rotated often to enable a better distribution of knowledge throughout the project.

How much planning in XP?

- Ideal weeks
  - Number of weeks estimated by a developer to implement the story if all work time was dedicated for this single purpose.
- Fudge Factor:
  - Factor to reflect overhead activities (meetings, holidays, sick days...)
  - Also takes into account uncertainties associated with planning.
- Project velocity:
  - Inverse of ideal weeks
    - i.e., how many ideal weeks can be accomplished in fixed time.
- Heuristic for new teams with no previous experience in XP
  - Start with a fudge factor of three (i.e., three actual weeks for one ideal week)
  - Lower this factor as time progresses.

How much planning in XP?

- Stacks
  - After estimating the effort needed for each story, the client and the developers meet to assign specific stories to releases, which correspond to versions of the system that will be deployed to the end user.
  - The user stories are organized into stacks of related functionality.
- Prioritization of stacks
  - The client prioritizes the stacks so that essential requirements can be addressed early and optional requirements last.
- Release Plan
  - Specifies which story will be implemented for which release and when it will be deployed to the end user.
- Schedule
  - Releases are scheduled frequently (e.g., every 1–2 months) to ensure rapid feedback from the end users.

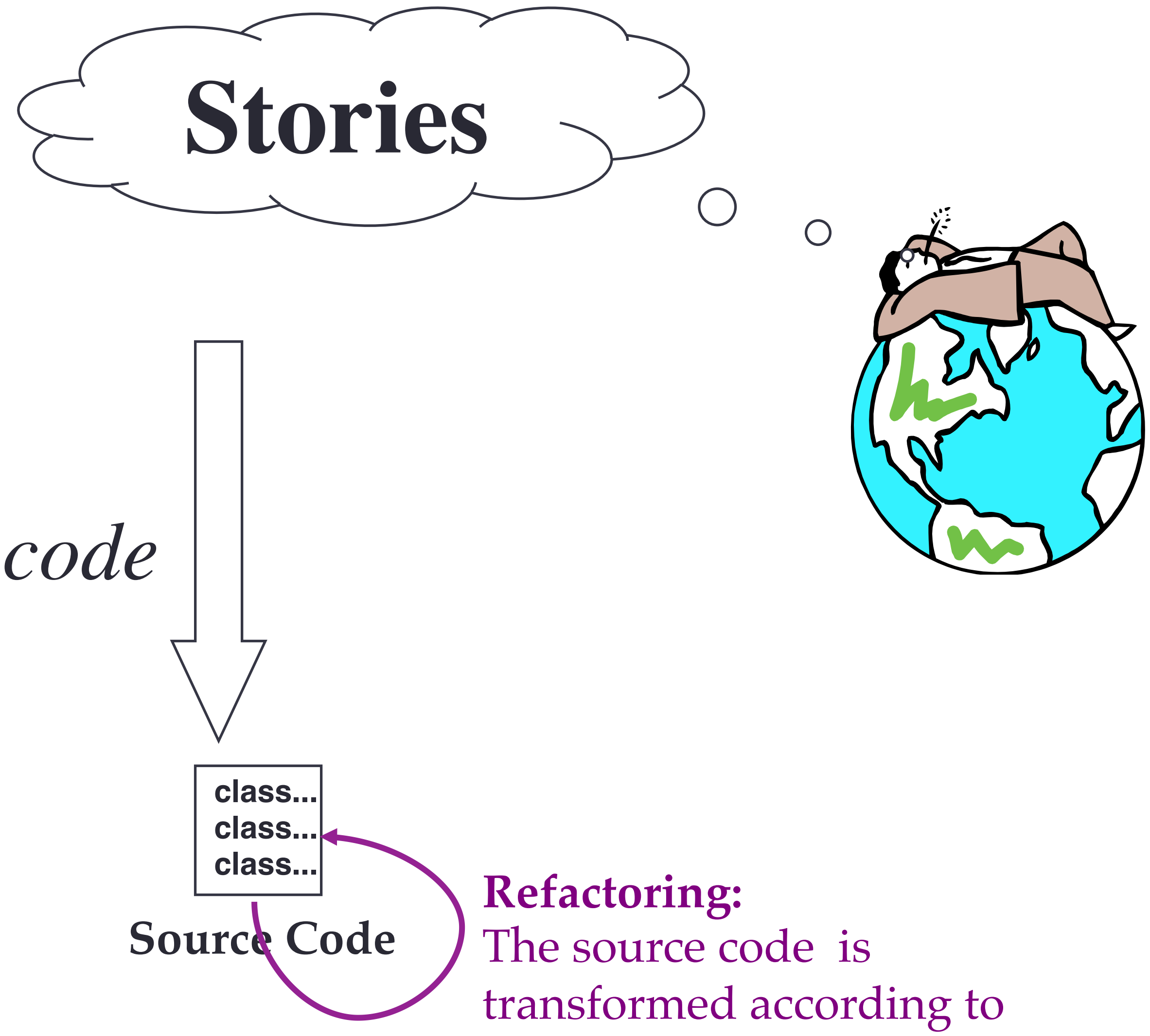
How much reuse in XP?

- Simple design
  - Developers are encouraged to select the most simple solution that addresses the user story being currently implemented.
- No design reusability
  - XP differs from conventional methodologies because it does not focus on the system architecture, which would allow such reuse to be planned up front.
  - XP argues, that the system architecture can be refined and discovered one story at the time, as the prototype evolves towards the complete system.
- Focus on Refactoring
  - Design patterns might be introduced as a result of refactoring, when changes are actually implemented.
  - Reuse discovery only during implementation

How much modeling in XP?

- No explicit analysis/design models
  - XP goal: Minimize the amount of documentation produced beside the code.
  - The assumption is that fewer deliverables reduces the amount of work and duplication of issues.
- Models are only communicated among participants
  - The client is seen as a "walking specification"
- Source Code is the only external model
  - The system design is made visible in the source code by using explicit names and by decomposing methods into many simpler ones (each with a descriptive name).
  - Refactoring is used to improve the source code.
  - Coding standards are used to help developers communicate using only the source code.

Models in XP (Story-Based)



Stories
generate source code

Refactoring:
The source code is transformed according to refactoring rules (program transformation)

How much process in XP?

- Very simple iterative life cycle model with activities: planning, design, coding, testing and integration.
  - Planning occurs at the beginning of each iteration.
  - Design, coding, and testing occur incrementally in rapid succession.
  - Source code is continuously integrated into the main branch, one contribution at the time.
  - Unit tests for all integrated units are used for regression testing.
- Constraints on these activities:
  - **Test first.** Unit tests are written before units. They are written by the developer.
  - **Write tests for each new bug**. When defects are discovered, a unit test is created to reproduce the defect. After the defect is repaired all unit tests are run again.
  - **Refactor before extending.** To ensure that the design does not decay as new features are added.

How much control?

- Reduced number of formal meetings
  - Status is communicated in the team in a daily stand up meeting.
  - Information sharing only, no discussions to keep the meeting short.
- No inspections and no peer reviews
  - Pair programming is used instead.
  - All production code is written in pairs, review occurs during coding.

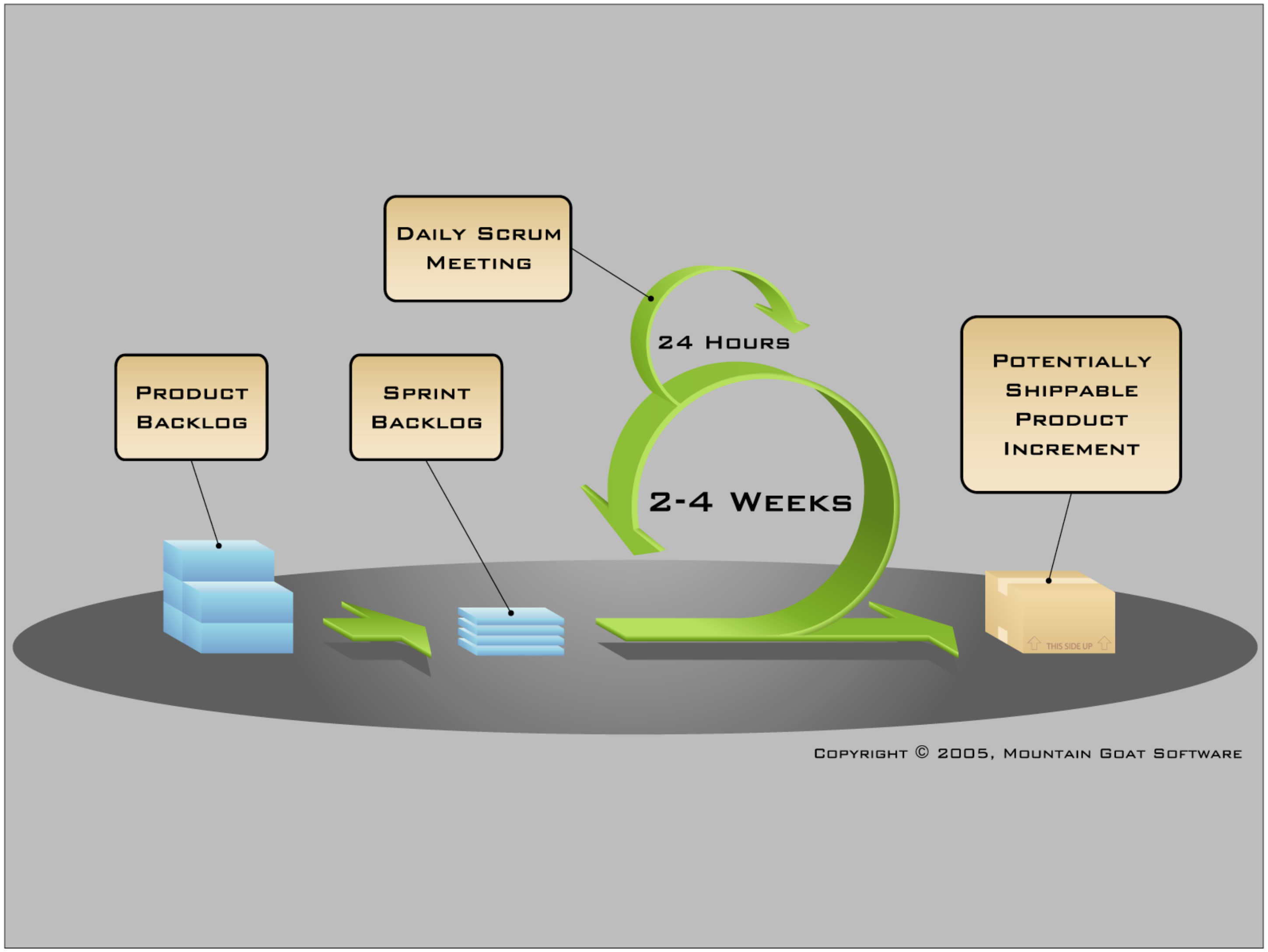
How much control?

- **Self-organizing System with a Leader**
  - The leader communicates the vision of the system.
  - The leader does not plan, schedule or budget (not a project manager)
  - The leader establishes an environment based on collaboration, shared information, and mutual trust.
  - The leader decides when to build consensus and when to dictate.
  - The leader ensures that a product is shipped.
  - Makes sure that participants pull the project in the same direction

Summary: XP Methodology

Planning	Collocate the project with the client , Write user stories with the client, Plan frequent small releases (1-2 months) Create schedule with release planning, Kick off an iteration with iteration planning, create programmer pairs, allow rotation of pairs
Modeling	Select the simplest design that addresses the current story Use a system metaphor to model difficult concepts Write code that adheres to standards. Refactor whenever possible
Process	Code unit test first, do not release before all unit tests pass, write a unit test for each uncovered bug, integrate one pair at the time
Control	Code is owned collectively, Adjust schedule, Rotate pairs, Start the day with a status stand-up meeting, Run acceptance tests often and publish the results

Scrum


Daily Scrum Meeting
24 Hours
Product Backlog
Sprint Backlog
2-4 Weeks
Potentially Shippable Product Increment
This side up
Copyright © 2005, Mountain Goat Software

Scrum Roles

- Core roles (often referred to as *pigs)*
  - Product Owner
  - Development Team
  - ScrumMaster
- Ancillary roles (*chickens)*
  - Stakeholders
  - Managers

HEY PIG, I WAS THINKIN' WE SHOULD OPEN A RESTAURANT.
I DON'T KNOW. WHAT WOULD WE CALL IT?
Ham -n- Eggs
HOW ABOUT "HAM -N- EGGS."
NO THANKS, I'D BE COMMITTED, BUT YOU'D ONLY BE INVOLVED!
By Clark & Vizdos
© 2006 implementingscrum.com

Scrum Roles

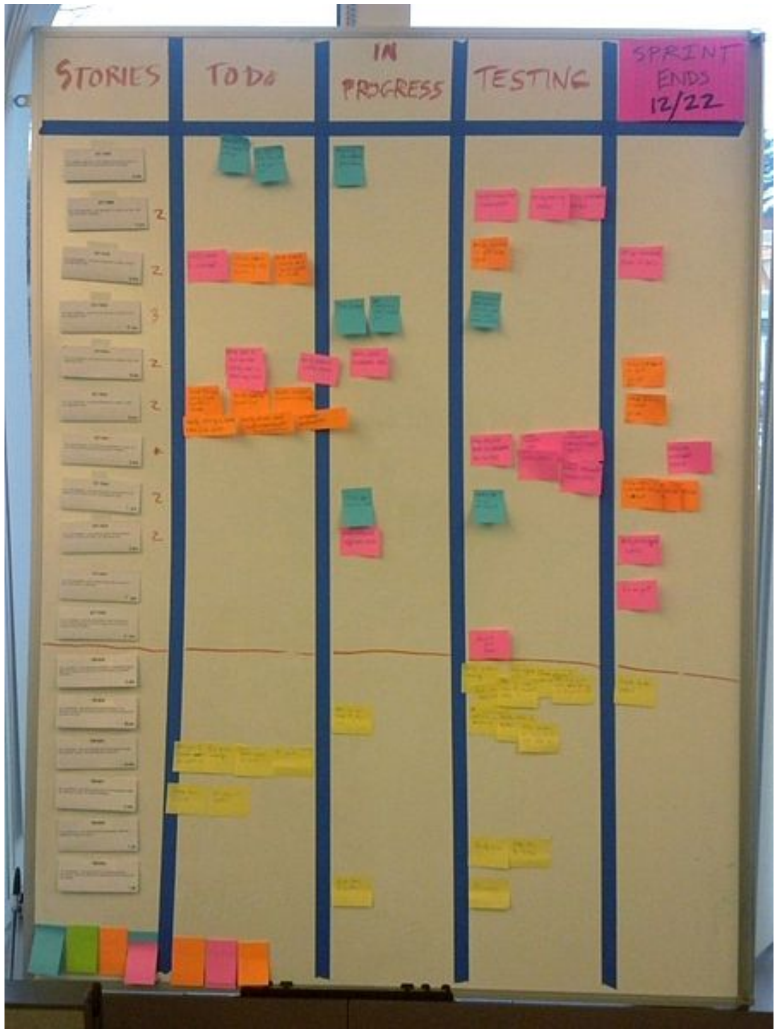
- Product Owner
  - represents the stakeholders and is the voice of the customer.
  - writes customer-centric items (typically user stories), prioritizes them, and adds them to the product backlog.
- Development Team
  - A self-organizing team responsible for delivering potentially shippable product increments at the end of each Sprint.
- ScrumMaster
  - accountable for removing impediments to the ability of the team to deliver the sprint goal/deliverables.
  - The ScrumMaster is not the team leader, but acts as a buffer between the team and any distracting influences.
  - A key part of the ScrumMaster's role is to protect the Development Team and keep it focused on the tasks at hand.
- Stakeholders: are the customers, vendors.
- Managers: People who control the work environment.

Scrum Terminology

- Product backlog
  - A prioritized list of high-level requirements.
- Sprint backlog
  - A prioritized list of tasks to be completed during the sprint.
- Stories
  - High-level use cases that encompass a set of coherent features.
- Daily Scrum
  - A daily standup meeting where each team member answers three questions:
    - What have you done since yesterday?
    - What are you planning to do today?
    - Any impediments?

In-Class Exercise

Form groups, assign a scrum master, define your product backlog, sprint backlog and plan a 2-week Sprint for your project



Any Questions?

iman@miami.edu

Any Questions?

iman@miami.edu