Introduction to Agile Life Cycles

CSCI 5828: Foundations of Software Engineering
Lecture 07 — 09/15/2015

Goals

- Introduction to Agile Life Cycles
  - The Agile Manifesto and Agile Principles
  - Agile Life Cycles
    - Example: Extreme Programming
- Additional Insight
  - Delivering Value to Your Customer
  - Adopting Customer Perspective
  - Adaptive Planning and knowing when you are done
  - Agile Teams

Agile Life Cycles

- Agile Methods were created in response to the negative qualities of traditional software development life cycles
  - too many documents created/maintained
  - a focus on the need for consistency across all documents
  - inflexible plans; process being valued over people
  - seeing the customer as external to the software development team
  - late, over budget, buggy software
- Agile focuses instead on human communication and collaboration
  - with software engineering practices designed to increase developer knowledge (and thus confidence in) the code base

Agile Manifesto

- “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”

Agile Principles

- From this statement of values, twelve principles have been identified that distinguish agile practices from traditional software life cycles
- Lets look at five of them
  - Deliver Early and Often to Satisfy Customer
  - Welcome Changing Requirements
  - Face to Face Communication is Best
  - Measure Progress against Working Software
  - Simplicity is Essential

Deliver Early and Often to Satisfy Customer

- MIT Sloan Management Review published an analysis of software development practices in 2001
  - Strong correlation between quality of software system and the early delivery of a partially functioning system
    - the less functional the initial delivery the higher the quality of the final delivery!
  - Strong correlation between final quality of software system and frequent deliveries of increasing functionality
    - the more frequent the deliveries, the higher the final quality!
- Customers may choose to put initial/intermediate systems into production use; or they may simply review functionality and provide feedback

Welcome Changing Requirements

- Welcome change, even late in the project!
- Statement of Attitude
  - Developers in agile projects are not afraid of change; changes are good since it means our understanding of the target domain has increased
- More importantly
  - agile practices (such as pair programming, refactoring, test driven development) produce systems that are flexible and thus, it is argued, easy to change

Face to Face Communication is Best

- In an agile project, people talk to each other!
  - The primary mode of communication is conversation
    - there is no attempt to capture all project information in writing
    - artifacts are still created but only if there is an immediate and significant need that they satisfy
    - they may be discarded, after the need has passed
      - as Kent Beck says “Shred It!”

Measure Progress against Working Software

- Agile projects measure progress by the amount of software that is currently meeting customer needs
    - They are 30% done when 30% of required functionality is working AND deployed
- Progress is not measured in terms of phases or creating documents

Simplicity is Essential

- This refers to the art of maximizing the amount of work NOT done
- Agile projects always take the simplest path consistent with their current goals
    - They do not try to anticipate tomorrow's problems; they only solve today's problems
    - High-quality work today should provide a simple and flexible system that will be easy to change tomorrow if the need arises

Agile Life Cycles

- Quite a few agile life cycles out there
  - Extreme Programming
  - Scrum
  - Lean Development
  - Feature-Driven Development
  - Crystal
- Our textbook will present insight into the benefits of the Agile approach
  - In class, I will supplement that book with coverage of additional material from previous versions of this class
    - We will look at Extreme Programming today and Scrum later this year

Extreme Programming

- One example of an Agile method is extreme programming
    - It was developed by Kent Beck during the late 90s when he became the project lead on a system called Chrysler Comprehensive Compensation System (C3). C3 was a payroll system written in SmallTalk
- The basic idea is that
    - it takes standard programming practices to the “extreme”
    - if software testing is good
        - then we’ll write test cases every day
        - and run them every time we make a change, etc.
- As Kent Beck says extreme programming takes certain practices and “sets them at 11 (on a scale of 1 to 10)”

XP Practices (I)

- Insight into Agile Methods can be gained by looking at some of XP's practices
  - Customer Team Member
  - User Stories
  - Short Cycles
  - Acceptance Tests
  - Pair Programming
  - Test-Driven Development
  - Collective Ownership
  - Continuous Integration
  - Sustainable Pace
  - Open Workspace
  - The Planning Game
  - Simple Design
  - Refactoring
  - Metaphor

XP Practices (II)

- Customer Team Member
  - The client should have a representative on the development team
- User Stories
  - Requirements are captured in brief statements about the functionality discussed with the client
- Acceptance Tests
  - Details of a user story are documented via test cases
  - The user story is complete when the test cases pass
- Short Cycles
  - Too many things can change during development, so plan to release working software every few weeks (typically 2 weeks, 10 working days)

XP Practices (III)

- Pair Programming
  - All production code is written by pairs of programmers working together
  - Studies in 2000/2001 indicated that pair programming helped to significantly reduce a project's defect rate while minimally impacting team efficiency
- Test-Driven Development
  - No production code is written except to make a failing test case pass
- Collective Ownership
  - A pair is allowed to check out any module and improve it
    - Developers are never individually responsible for a module
    - The system is owned by the team

XP Practices (IV)

- Continuous Integration
  - The system is built and deployed at least once per day
  - Helps to identify integration problems early
  - Encourages developers to “grow” a system incrementally
- Sustainable Pace
  - Software development is not a 5K race, it’s a marathon
  - You need a sustainable pace or your team will burn out
  - As a result, XP teams do not work overtime; “40 hour work week”

XP Practices (V)

- Open Workspace
  - Pairs work near each other in order to promote “team awareness” of the current state of the system
  - The team naturally helps each other as problems are encountered
    - Some pushback on this: others prefer pairs to work in isolation to allow them to “get in the flow” and avoid interruption
- The Planning Game
  - Estimates are attached to ALL user stories
    - The team creates the estimate (in terms of points)
    - The customer assigns priorities
    - Each iteration, we use the priorities and estimates to decide what to work on

XP Practices (VI)

- Simple Design
  - XP emphasizes simplicity at all times
    - “Consider the simplest thing that could possibly work”
    - “You ain’t going to Need It”
    - “Once and Only Once” (Don’t Repeat Yourself)
- Refactoring
  - Supported by test cases, XP teams constantly refactor their code to fight “bit rot”: clutter that can accumulate over time in a design
- Metaphor
  - Make sure to have a theme that ties the entire system together
  - Can be used to discuss the system’s architecture and improve morale (t-shirts!)

Shared Goal: Delivering Value to your Customer

- Extreme programming is just one example of an agile method
  - Other agile methods will differ in some of the practices, the way they arrange the work day, or the way they arrange the team (such as Scrum)
- However, they all have a shared goal
  - Delivering something of value to your customer every iteration
- If you adopt the customer's perspective, this makes sense
  - What do you want to see from the developers working on your project?
    - Status reports or working code?

Who is the customer?

- The person or persons playing the role of the customer can vary across development contexts
    - This is very important because sometimes the answer will be hard to pin down
        - Consider a case where you are asked to develop a website for a small business
            - The owner of the small business is clearly the customer at first
                - he/she is providing requirements and paying for the work
            - But when the website is deployed, who becomes the customer?
                - The customers of the small business

Customer == User

- HCI and CSCW research shows that systems live or die by how happy the "end users" are with the system
    - The customers of the small business in this case are the end users
        - However, in the initial development project, we will only have access to the owners of the small business and we'll have to go by what they say
        - In the future, they will be hearing from their customers about the utility and usability of our website and they will convey that feedback to us
- What's the difference between utility and usability?

Other Types of Customers

- You (!)
    - Often for only small scale software
    - “Scratch an itch”
- CTOs
    - Acquiring enterprise level systems for an organization
    - Who are the end users in this situation?
- New Application Development (be it desktop, web, mobile)
    - For version one: development team
        - How can you avoid this? Who are the end users?

Benefits from Adopting a Customer Perspective

- Iterations are short, so break big problems down into small ones
  - Deliver incremental sets of functionality each iteration
  - Let customer priorities help you decide what functionality to focus on
    - which helps ensure that what you deliver has value to the customer
- Deliver working software
  - This will, in turn, place a priority on testing and encourage the adoption of best practices with respect to testing: unit tests, continuous testing, etc.
- Seek feedback and be prepared to shift directions in response to feedback
  - The goal is to please the customer and deliver the functionality they want
- Accountability: Short iterations encourage taking ownership of the process and accepting responsibility if things go wrong

Agile in a Nutshell

- The highest priority of an agile life cycle is meeting a customer's needs via early and frequent delivery of working software

- Discussion
  - "Not everyone likes working this way"
  - However, this stance provides benefits in that development is always directed at what the customer wants
    - and is never allowed to drift too far off course
  - But it is a highly visible process that requires people to make commitments and stick to them, work together rather than in isolation, and be ready to take responsibility when they fail to deliver

Agile Planning

- The generic way in which an agile project works is the following
  - the team has a **master story list** that documents all requested features
    - each item on the list is a **user story**, a short description of the functionality requested by the user
- The team's work is split into **iterations** that typically last 1 to 3 weeks
  - An iteration has a set of assigned user stories to be turned into software
  - The stories assigned to an iteration is determined by **team velocity**, a metric based on the amount of work performed on previous iterations
  - If you have too much to do on an iteration, **you do less**; this is called reducing the scope of the iteration; cornerstone of **adaptive planning**
  - if budget/schedule are fixed, the only thing that can change is scope

Be Open to Change

- There are three facts about software development that, if accepted, can eliminate many of the problems encountered in traditional life cycles

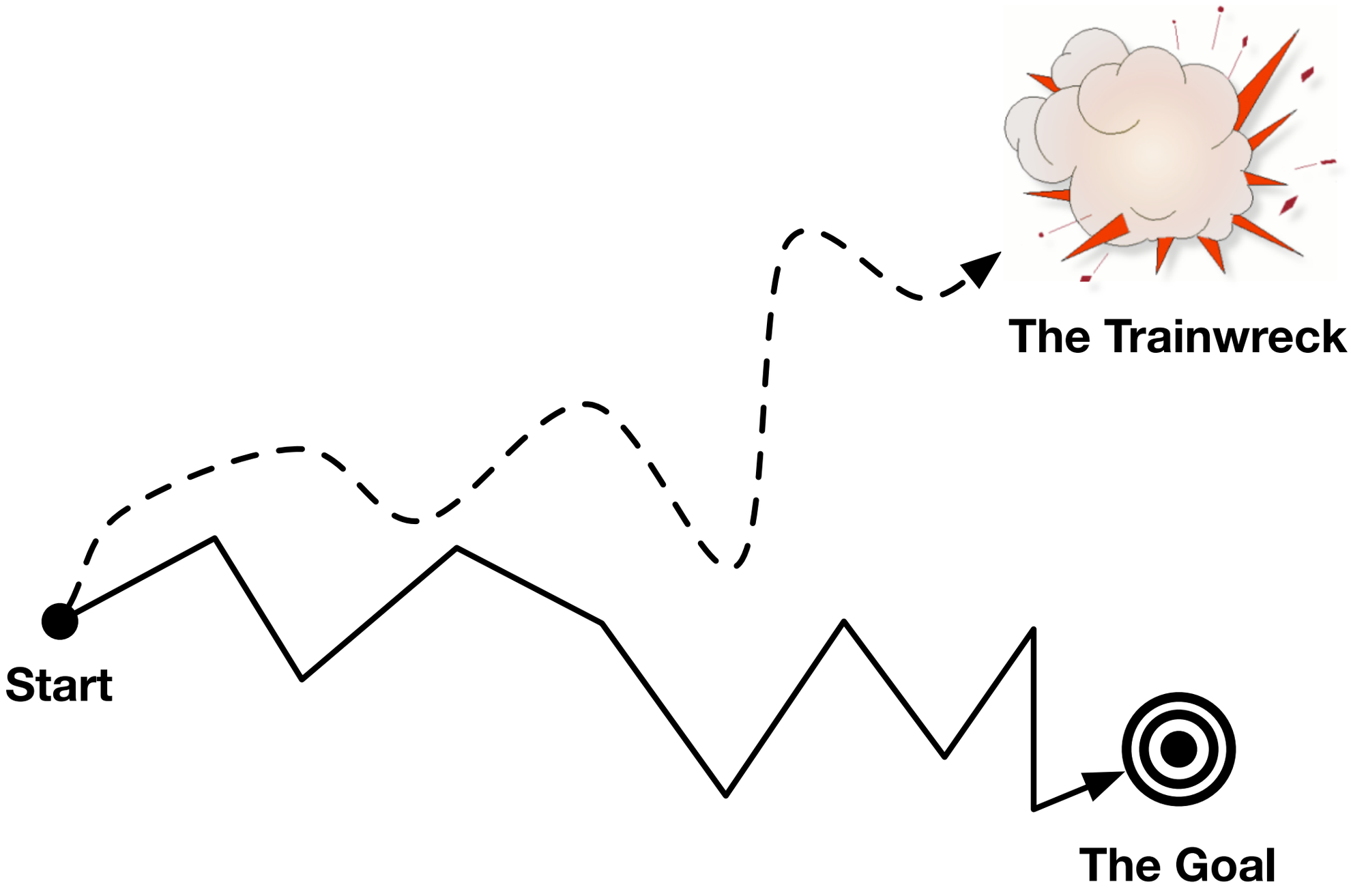
    1. It is impossible to gather a complete set of requirements at the start of a project

    2. It doesn't matter, because the requirements will change

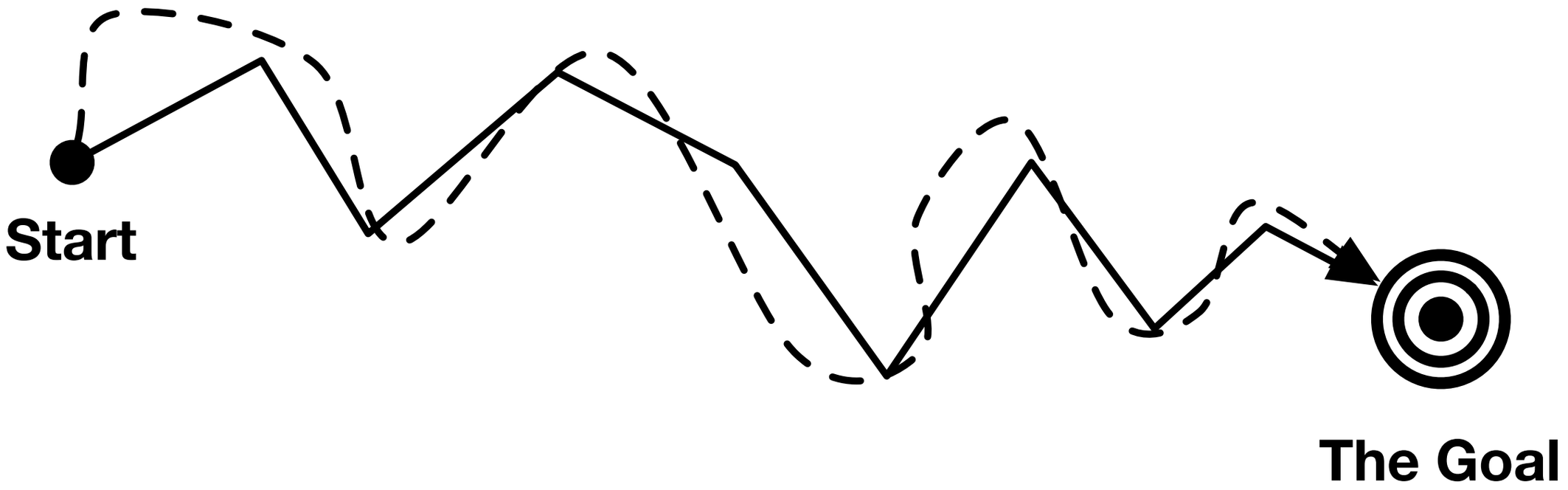
    3. There will always be "one more requirement"

        - you must prioritize and complete what can be done within the constraints of the schedule and budget

Iteration is important because requirements CHANGE



With iteration a project can make course corrections as requirements change so that what's delivered matches what's needed



When are you done?

- Progress in agile projects are measured by working software
  - 30% done when 30% of required functionality is working AND deployed
  - A user story is done when the functionality it describes has been **shipped**
  - The project is done when all user stories that can be completed *within the constraints of the schedule and budget* are shipped

- That means each time we tackle a user story, we will be performing
  - analysis, design (both UI and software design), implementation, testing, and deployment
- for that story

Team Structure

- The software engineering literature has a lot to say on team roles and team structures
    - dating all the way back to the late 1960s when Fred Brooks and others would talk about structures such as the “Chief Programmer” team
- Typical Roles:
    - analyst, developer, tester, UI designer / usability engineer, tool builder, database administrator, language expert, etc.
- Agile methods turn this on its head
    - roles blur and team members are expected to switch roles and do more than one thing

Different Environment

- The roles in an agile team blur because the traditional stages of software development are blurred as well
  - In one iteration, you might be
    - performing requirements analysis on one user story
    - fixing bugs for a user story deployed on the previous iteration
    - creating a rapid prototype on a third user story
    - implementing the design of a fourth user story
- Analysis, Design, Implementation, Testing, and Deployment are continuous activities as opposed to distinct phases

One Team

- Finally, to support this blur of roles and activities, Agile places the emphasis on “one team” versus “collection of developers”
    - It is not “Ken” who is responsible for the quality of the “Add Employee” feature
        - it is the team’s responsibility
        - There is no QA department; YOU’RE the QA department
- As a result, there is less structure and more responsibility in agile environments
    - There is also a lot of variety in daily work and a need for people who are good at more than one thing

Characteristics of an Agile Team (I)

- Co-location of team members
  - It's very important to have the team be co-located
  - The Agile Manifesto emphasized "face-to-face" interactions
  - If you can't pull this off, bring the team together on a regular basis and otherwise make use of social media, video conferences, etc. to facilitate day-to-day interactions
- Engaged Customers
  - An agile team works to develop trust with their customer and to encourage that customer to be invested in the process and engaged with the development
    - You want that person or persons to be ready to answer your questions and provide feedback both during and at the end of an iteration

Characteristics of an Agile Team (II)

- Flexibility
  - Agile teams value flexibility in individual team members as well as in the team as a whole
  - To maximize flexibility, Agile teams much be
    - **self-organizing**: once they know their goal, they work together to achieve it
    - **accountable and empowered**: they have the power to make their own choices about design and implementation; they also have to take responsibility when those choices go wrong
      - the fact that working software must be delivered at the end of each iteration will help instill this in the team or cause the team to fail early
    - **cross-functional**: the team needs people who can do more than one thing; we will prefer generalists over specialists

Agile Team Archetypes

- Across life cycles, there tend to be six archetypes for agile teams
  - agile customer
  - agile analyst
  - agile programmer
  - agile tester
  - agile project manager
  - agile UX designer

Agile Customer

- The agile customer serves as “ground truth” for the requirements captured in user stories
    - If ever the developers detect ambiguity in the user stories, the agile customer is the person who will resolve that ambiguity
- They are also, ideally, **one of the people who will use the system being developed**
    - As opposed to a CIO type, who might purchase the system but never use it
- By setting requirements, they determine what gets built
- By setting priorities, they determine what gets built first
- They are the ones who determine what stories get dropped if the team is running out of time and/or money

Agile Analyst

- The Agile analyst is a team role that orients more towards the customer
  - The analyst will help write user stories
  - The analyst will elicit feedback from the customer and convey it back to the rest of the team
- During an iteration, the analyst will track down all of the relevant requirements for a user story as questions are generated by the rest of the team
  - They can develop paper prototypes, storyboards, UML diagrams and the like that help to capture the details of the story now that it is up for implementation

Agile Programmer

- The Agile programmer is someone who can convert a user story into code
- They can help to generate estimates for user stories that have recently been added to the master story list
- They are technically literate and can make design decisions about the tools, frameworks and architectures to use to help address the needs of the current set of user stories
- They will generate test cases and use the tests to move the project forward
- They will look for opportunities to refactor the code to keep it as simple as possible
- They will work to identify, configure, deploy and use the tools that the team will need to be successful

Agile Tester

- The Agile tester is a developer who has decided to focus on ensuring quality of the overall system via tests
  - They will write test cases for user stories
  - They will gather data about the system
    - in terms of the number of tests that pass
    - in terms of the performance of the system
- Another way to state this is that they will test both the **functional and non-functional** aspects of the system
  - The functional aspects of a system refer to its capabilities, the tasks that it supports, the non-functional refer to things like performance, scalability, reliability, robustness, security, etc.

Agile Project Manager

- The Agile project manager is a team member who spends time
  - tracking how well the team is doing both during an iteration and across several iterations
  - communicating the state of the project to the team, the customer, and all other relevant stakeholders
  - works to remove obstacles that the team might be facing so the rest of the team can focus on the current iteration and producing working software
  - identifying when the current plan isn't working and leading the team through the adaptive planning techniques that will get them back on track

Agile UX Designer

- A team member who focuses on
  - the visual aspects of the system
  - the types of interactions the system supports
  - ensuring that the user experience is of high quality
- The UX Designer will work closely with the Agile analyst in making sure the customer's requirements are well understood
  - and then provide the team with the information they need to translate the requirements to design decisions
    - personas, scenarios, prototypes, storyboards, etc.

Summary

- High level Introduction to Agile Methods and the Agile approach to development teams
  - Reviewed Extreme Programming as one specific Agile life cycle
  - Delved more deeply into principles associated with Agile methods
  - Looked at the characteristics and roles of Agile teams

Coming Up Next

- Lecture 8: Introduction to Concurrency