Class Notes

* Resources - Each team member with 8-12 hours a day
* Date - Two week sprints
* Easier to sacrifice time for 2 week sprints. Fixed for short period
* Questions
  + Context free
  + Binary

Persona - Staff adding a ministry or campus

Ch 1 Software Requirements Methods

* Waterfall
  + Requirements
  + Design
  + Implementation
  + Verification
  + Deployment
* Iron Triangle (All trying to be fixed, so quality diminishes)
  + Requirements - Known first (Fixed)
  + Cost (Estimate)
  + Schedule (Estimate)
* Iterative Processes
  + Spiral
    - Iterative requirements, design, integration, and testing
    - Goes through the process multiple times
    - Grows in scale with each iteration
  + RAD (Rapid Application Development)
    - Short cycle of design, development, testing, and documentation
    - Small and fast development iterations
  + RUP (Rational Unified Process)
    - Has life cycle phases of inception, elaboration, construction, and transition
    - Overlap in requirements, design, etc
* Agile fixes cost and schedule, estimates requirements
* Agile Manifesto
  + Individuals and interactions over processes and tools
  + Customer collaboration over contract negotiation
  + Working software over comprehensive documentation
  + Responding to change over following a plan
  + Satisfy customer with early and continuous valuable software
  + Always welcome changing requirements
  + Quick ROT (Return on Investment)
* Agile Processes
  + XP (Extreme Programming)
    - Small team with customer representation
    - Small iterations
    - Requirements made as user stories
    - Work in pairs
  + Scrum
    - Sprints of short iterations of a few weeks (Often 30 days, 3 sprints per release)
    - Sprints are fixed set of work, categorized through a backlog
    - Scrum master mentors teams
    - Product owners play role of customer proxy
    - Daily stand ups
* Lean Software
  + Sustainable delivering value fast (The Goal)
    - Focus on requirements rather than managers of requirements
    - Minimize non-value-adding activities
  + Respect for people (Pillar 1)
  + Continuous improvement (Pillar 2)
    - Personally make choices and observe data to improve
    - Reflect on milestones
  + Management support (Foundation)
    - Managers take active role in improvement and leadership
  + Product development flow (Content)
    - Take an economic view
    - Actively Manage Queues
    - Exploit Variability
    - Reduce Batch Size
    - Apply WIP Constraints
    - Control Flow
    - Apply Fast Feedback
    - Decentralize Control
* Kanban
  + Uses units if value such as features or requirements
  + Created pipeline of these units

Ch 2 Big Picture

* Team level - Define, build, test user stories in iterations and releases
  + Small “pods”/ teams
    - Product owner owns backlog. Prioritizes and maintains backlog
    - Scrum master assists teams in adjusting to agile and maximizes performance
    - Developers and testers make up teams
  + Features and components
  + Iterations / Sprints - Short time periods to create functionality
  + Releases aggregate larger functionality for users
  + User stories - Software requirements, statements of intent
    - As a <user role>, I can <activity>, so that <business value>
  + Backlog
    - Comprised of user stories
  + Tasks deliver stories, provides micro-work breakdown for estimation, assigning responsibility, and tracking
* Program level - Development of large scale systems with a synchronized Agile Release Train
  + ART - Standard of timeboxed iterations, Releases or potentially shippable increments
  + Product management maintains Vision
    - Represents interests of stakeholders
    - Manage release content
    - Maintain roadmap
    - Build effective team
  + Vision - set of features and nonfunctional requirements (Accuracy, responsiveness, etc.)
  + Release planning
  + Roadmap outlines planned releases and shows increase of value over time
* Portfolio Level - Investment themes, delivering on business strategies in larger epics
  + Investment themes - investment objectives, key product value propositions that provide marketplace differentiation and competitive advantage
  + Epics - High level expression of a customer need
  + Epics accumulate on Portfolio Backlog
  + Architectural runway - Ability to implement without excessive refactoring
    - Architecture must be addressed
* Vision scope is creating a roadmap on a portfolio level
* Software Requirement Specifications are defined by user stories

Ch 3 Agile Requirements for the Team

* Team is involved in defining requirements, optimizes for development, and testing
* Story is created
  + Define - details of story
  + Build - development of code
  + Test - passing acceptance test to assure code meets intent of story
* Teams need to have capability to define, build, test, and accept stories daily
* Redefine and eliminate functional silos that separate developers, business analysts, etc.
* Organization
  + Product Owner
    - Determines requirements
    - Maintains backlog
    - Sets objectives
  + Scrum Master
    - Facilitates progress
    - Leads efforts in improvement
    - Enforces agile process
    - Eliminates impediments
  + Developers
    - Collaborate
    - Write code and test code
  + Testers
    - Write acceptance tests
    - Tests
    - Develops test automation
  + Other
    - Architects (Infrastructure)
    - QA (Overall system level quality)
    - Specialists
* Backlog
* User stories
  + Card - Intent of story
  + Conversation - Details of intent of card made with customer
  + Confirmation - Understanding that code meets full intent of the story
* Task - A small unit of work that is necessary for the completion of the story
* Test driven development
* Acceptance tests
  + All code is tested code
  + Every story has one or more acceptance tests
  + Story is done when it passes acceptance tests
* Unit tests
  + Tests code

Pg. 251-253

* Vision
  + Why, what, and for whom?
  + Features, benefits, performance, and platforms
  + Advanced Data Sheet Approach
    - 2 page template to define features, specifications, message, etc.
  + Preliminary Press Release Approach
    - Create a mock press release of finished product
  + Feature Backlog with Briefing Approach
    - Backlog created to define requirements

Appendix A & B

* Content free interview
  + Establish Customer or User Profile
  + Assess Problem
  + Understand User Environment
  + Recap for Understanding
  + Analyze Customer Problem
  + Assess Solution
  + Assess Opportunity
  + Assess Reliability, Performance, Support Needs
  + Address Other Requirements
  + Wrap up
  + Summarize
* Vision Document Template
  + Introduction
    - Purpose
    - Solution Overview
    - References
  + User Description
    - User/Market Demographics
    - User Personas
    - User Environment
    - Key User Needs
    - Alternatives and Competition
  + Stakeholders
  + Product Overview
    - Product Perspective
    - Product Position Statement
    - Summary of Capabilities
    - Assumptions and Dependencies
    - Cost and Pricing
  + Product Features
    - Feature 1
    - Feature 2
  + Exemplary Use Cases
  + Nonfunctional Requirements
    - Usability
    - Reliability
    - Performance
    - Supportability
    - Other Requirements
  + Documentation Requirements
    - User Manual
    - Online Help
    - Installation Guides, Configuration, “Read Me” File
    - Labeling and Packaging
    - Glossary

Ch 6 User Stories

* Tested, integrated code implements a story
* Statement of intent of something system does
* Common language
* Not requirements
  + Common language
  + Short
  + Small increment
  + Easy to estimate
  + No maintenance
* Form
  + Card
    - Intent
  + Conversation
    - Conditions of satisfaction
  + Confirmation
    - Acceptance test
* User Voice - As a <role>, I can <activity> so that <business value>
* Writing code often isn’t as hard as knowing the objective
* A good INVESTed story
  + Independant
  + Negotiable
  + Valuable
  + Estimable
    - Fibonacci
  + Small
    - Cycle Time = WIP / Throughput
    - Decrease work in progress to release faster
    - Decreased complexity (Nonlinear increase of complexity with size)
      * Do one thing
      * Keep them small
      * Make them smaller than that
  + Testable
* Compound and Split Stories
  + Workflow steps
  + Business Rule Variations
  + Major Effort - One story has most of the implementation
  + Simple / Complex - Start simple
  + Variations in Data
  + Data Entry Methods
  + Defer System Qualities
  + Operations - Seperate “manage” and “control” with specific functions
  + Use-Case Scenarios
  + Break Out a Spike
* Spikes
  + Technical Spike - Research
  + Functional Spike - Prototype or implement
  + Estimable, Demonstrable, and Acceptable
* Index cards
  + Small
  + Differentiable
  + Physical

Ch 7 Stakeholders, User Personas, and User Experiences

* Stakeholders
  + System Stakeholders
    - Directly uses system
    - Works with results of system
    - Impacted by deployment and operation
    - Examples
      * Consumers
      * Vendors
      * Manufacturers
  + Project Stakeholders
    - Has a vested interest in budget and schedule
    - Interest in understanding development
    - Involved in marketing, installing, maintaining
    - Examples
      * Partner
      * Sales
      * Operations
      * Security
  + Product Owner - Voice of stakeholder
    - Facilitates
    - Leads
  + Involvement
    - Kept informed
    - Consulted
    - Partners in development
    - Control of outcome
  + Build trust and know what is important to them
  + Expectations
    - Time decisions
    - Active participation
    - Enterprise view of working with others
    - Production and support should be involved from start
    - Plan for maintainance
* User Personas
  + Types
    - Primary - Satisfied only with UI designed for them
    - Secondary - Can use interface designed for primary
  + Guidelines
    - Identify goals
    - Produce from discovery, not creation
    - Specific
  + Identify what they expect and what they need
* UX Problem
  + Agile goes through many (“wastes”) designs and design alternatives
  + No Big Up Front Design (BUFD)
* Low Fidelity Options
  + Define how it should look ahead of iteration w/ simpler tools
* UX Spikes - Separate UX stories from coding
* Centralized UX Development
  + Single, separate UX team
  + Iterates ahead of other teams
  + Can create conflicting styles across a system
* Distributed but Governed
  + UX Distributed throughout teams, but with a central authority or guide

Ch 19 Use Cases

* User stories lack context
* Help explore interactions of Users, Systems, Subsystems
* 5 Reasons to use
  + List of goals / Skeleton for management
  + Context
  + Framework for investigating small issues that take time to answer
  + Answers to details for programmers
  + Think through every customer need
* Use case is a sequence of actions between an actor and a system that produces a result of value for that actor
  + Sequence of actions - Atomic inputs (performed or not)
  + System - Function with orders from actor
  + Result of Value
  + Actor - Initiates action
    - Users
    - Other systems
    - A device
* Structure
  + Name - Description of goal
  + Description
  + Actor
  + Flow of Events
    - Basic
    - Alternative
  + Preconditions, exit conditions, system, other stakeholders, special requirements
* Steps
  + Identify Actors
  + Identify Use Cases
  + Identify Relationships
  + Outline Flow
  + Refine
* Can be elaborated and used in conjunction with backlog items

10/27

UML Ch 1 Intro

* UML - Unified Modeling Language
* Model Driven Architecture - standard as programming language
* Platform Independant or Platform Specific Model (PIM vs PSM)
* Conceptual vs software perspective

Development Process

* Iterative and Waterfall
* Regression tests
* Refactoring
* Continuous iteration

ASR

Ch 13 Vision, Features, & Roadmap

* Prioritize by ROI (Return on investment)
  + Relative Priority = Relative ROI = Relative (Value / Cost)
  + If high ROI is less sensitive to delays, counterintuitively, the lower ROI feature should be implemented first
  + COD (Cost of Delay)
    - User value
    - Time value
    - Risk
  + Feature Effort Estimate
    - When CODs are equals, do smallest (effort) features first
    - When efforts are equal, do high COD features first
    - When differ, prioritize by CoD / Effort (aka Weighted Shortest Job First)
  + Features
    - Basic (must have) (high satisfaction slope for present, low slope for enhanced)
    - Linear Features (linear slope)
    - Exciters and Delighters (low present, high enhanced)
  + Roadmap
    - Feature set
    - Release dates

Ch 18

* Requirements Analysis Toolkit
  + Sample Reports
  + Pseudocode
  + Decision Tables / Trees
  + Finite State Machines

Ch 20 Agile Architecture

* Portfolio Planning - larger with investment themes
* Architectural epics - Tech dev initiatives
  + Time
  + Scope
  + Organization
  + Values other than user values, such as costs and efficiency
* Requirements cause design decisions, design options initiate requirements
* Some architecture comes out of agile, some doesn’t
  + Something has to tie everything together
* Business Drivers exist for Architectural Epics
  + New products, changes in technologies, existing problems, etc
* Job of system architect
  + Issues
    - Agile often doesn’t define that role
    - Experience is often outside of agile
    - Concerns and differing opinions of architect
  + Legitimize role to avoid some of these issues
* Eight Principles of Agile Architecture
  + Teams that code system design system
  + Build simplest working arch
  + When in doubt, code or model
    - Architecture UML
      * Use-case view
      * Logical View
      * Process, Deployment, Implementation, Data
  + They build it, they test it
  + Bigger the system, longer the runway
  + System arch is a role collaboration
  + There is no monopoly on innovation
    - i, i, i, h, k cycle
    - i is standard, h is hardeneing, k is innovation
  + Implement architectural flow
* Implementing Arch Epics
  + Big, incremental, system always runs
    - Best case
  + Big, not entirely incremental, system takes a break
    - Stop and rethink
    - Break it down, refactor, put it back together quickly
    - Higher risk, not much value delivery
  + Really big and not incremental, system runs when needed
    - Built in background, isolated
* Splitting Architectural Epics
  + Subsystem / product
  + System qualities incrementally
  + Incremental functionality
  + Scaffolding
  + Major / minor effort
  + Complex / simple
  + Variations in data
  + Spikes

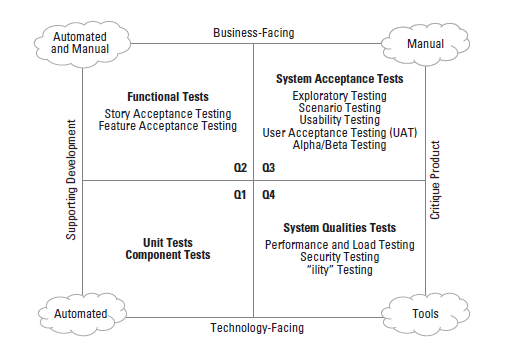
Ch 8 Agile Estimating and Velocity

* Goal: More Reliable Estimates
  + Know where you are now
    - Evaluation of working code, knowing exactly where code is
  + More accurately predict where you will be next
* Why estimate?
  + Determining cost
  + Establishing prioritization
  + Scheduling and commitment
  + *Ability to commit*
* Story Points
  + Knowledge
  + Complexity
  + Volume
  + Uncertainty
  + *Relative*
* When in doubt estimating, get clarification
* Estimate is from team as a whole
* Cohn’s modified Fibonacci- 0, 1, 2, 3, 5, 8, 13, 20, 40, 100
  + 0 means quick changes won’t affect estimates
  + Higher numbers are harder to estimate, leans you towards splitting stories
* Spending too much time is a waste
* Planning poker - every writes estimates, shows, and debates
* Tabletop relative estimation
  + Placed in order, shuffled, pointed (What I do)
* Velocity - points / iteration
* Responses to velocity, things to be wary of for “improving velocity”
  + Improving productivity and agility
    - Less immediate
  + Cut back on quality
  + Increase estimates
* Schedule- # days to do work = # days per iteration \* (backlog size estimate / velocity)
* Cost- average cost / velocity = cost per story point
* Problems with story points
  + Hard to understand
  + Hard to get started
  + Cost and schedule is indirect
  + Struggle to adjust velocity
  + Not normalized team to team
* Ideal Developer Days
  + Estimate from days to accomplish
  + Easier to understand
  + Easy to adjust for sick leave, vacation, so on
  + People get caught up in time estimation
  + More personal and political
* Hybrid
  + Smallest story that takes a day is 1 point
  + About 8 IDDs per member per 2 week iteration
  + More obvious translation
  + Less personal, still consensus
  + Normalizes velocity

Ch 9 Iterating, Backlog, Throughput, and Kanban

* 2 weeks
  + Meaningful incremental value
  + fastest affordable feedback
  + forces small stories
* Plan, execute, review, retrospective
  + Team backlog
  + Plan
    - Refine stories
    - Prioritize
  + Iteration Commitment
    - Velocity based commitment
      * Based of previous iteration velocity
      * Broad brushstroke
    - Objective based commitment
      * Broader brushstroke
      * Pull items in as you go
      * Set objectives for iteration
    - Task based commitment
      * Tasks
        + Specific
        + Measurable
        + Achievable
        + Relevant
        + Timeboxed
    - Iteration with
      * Objective
      * Prioritized list
      * Estimated tasks and owners
      * Commitment to objectives
      * Documentation
  + Execution
    - Take responsibility
    - Develop
    - Deliver to build
    - Declare as developed, ready for review
    - get Accepted
  + Tracking and Adjustment
    - Daily stand ups
      * What I did
      * What I will do
      * What’s blocking
    - Meet after
    - Track with agile tool (like jira)
  + Review and Retrospective
    - Demo with deliverables
    - Discussion and feedback
    - Did we accomplish everything?
    - if not, why not? And impact on future.
    - Retrospective
      * What went well
      * What didn’t
      * What can we improve
* Backlog grooming
  + One source, one owner
  + If it isn’t there, it isn’t happening
  + Current prioritization
  + Large backlog can mean months or years between request and delivery
    - Increase rate of story completion
      * Gain better understanding
      * Decreasing size of stories
    - Decreasing length of queue
      * Forcing limit on length of queue
  + Long backlog means
    - Increased risk
    - Increased variability
    - Increased costs
    - Reduced quality
    - reduced motivation and initiative
* Kanban systems
  + WIP limits on backlog
  + Pipeline of value flow
  + Pull system when there is capacity for new work
  + Core properties
    - Visualize workflow
    - Limit WIP
    - Measure and manage flow
    - Make process policies explicit
    - Use models to recognize improvement opportunities
  + Classes of service
    - Expedite: Unacceptable cost of delay
    - Fixed delivery date: Step function cost of delay
    - Standard class: Linear cost of delay
    - Intangible: Intangible cost of delay

Ch 10 Acceptance Testing

* Business facing tests
* Technology facing tests
* Support programming
* Critique the product
* Acceptance testing
  + Story acceptance - delivers expected behavior
    - Business domain language
    - Conversation of developers, product owner, testers
    - Product owner owns tests
    - Black box test
    - Implemented during iteration the story is implements
  + Test good user stories
  + Unambiguous and test all scenarios
  + They persist, documented
* Acceptance test driven development
  + Write test first
  + Run it and watch it fail
  + Write minimum code to pass test
* Conditions of satisfaction
* Automated acceptance testing
  + Framework for Integrated Tests
    - Created and run under test separate from system
    - Like unit tests, easier to write
* Unit testing
  + Written before or concurrently
* Component testing