Agile Recap - Answers/Summary

Agile principles

- Put customer at the center
- Let team self-organize
- Work at a sustainable pace
- Develop minimal software (functionality, only product requested, only code & tests)
- Accept change
- Develop iteratively (Frequent working iterations, freeze reqs during iterations)
- Treat tests as key resource (No new development with failing tests, Test first)
- Express Reqs. through scenarios

Practices

- Daily meeting
  Coordinate 24hrs team plan, raise blockers. What did you do? what will you do? any blockers?. Around 15 min, no technical discussion allowed
- Planning game/poker
  Onsite customer/PO prioritizes stories, developers estimate them, game finishes when all agreed on estimate
- Continuous Integration
  Reduce tech debt by bringing together software/product components frequently. Automated build detects integration errors.
- Retrospective
  What went well? What went less well? How do we improve? Team + coach (SM), PO can attend
  1. Brainstorm subjects to improve
  2. Vote on highest priority improvement
  3. Do short analysis
  4. Create improvement story, place it atop next sprint backlog
  5. Schedule Problem Solving Workshop if necessary
- Shared code ownership & cross-functional teams - it's lower down in this file
- TDD, Refactoring also lower down

XP Values

- Communication - everyone on team works jointly on all stages
- Simplicity - write simple code
- Feedback - deliver frequently & get feedback on reliveries
- Respect
- Courage - Evaluate own results without making excuses, be ready to respond to change

XP Principles

- Rapid Feedbvack - Team members understand and respond to feedbvack
- Assumed simplicity - Follow YAGNI and DRY
- Incremental changes - Small changes to the product
- Embracing change - if client requests changes, support the decision and plan implementing the reqs.
- Quality work - Team works well, makes a valuable product, is proud of it

XP Practices

- Pair programming
- Onsite-Customer
- Test-first
- Planning game
- Feedback
- Continuous process - Small releases, CI, Refactoring
- Programmer welfare - sustainable pace
- Shared understanding- Coding standards, Collective Code Ownership, System metaphor, simple design

Scrum roles

- Product Owner maximizes value added by owning and deciding on priorities in the Team Backlog from a business perspective. Maintains technical integrity of features + components
- Dev teammember is part of cross-functional team accountable for providing value to the larger solutions/systems
- Scrum master is a servant leader + coach that uses/enforces Agile practices (sprint planning, dailys, demos, retrospectives). Focus opn flow and continuous improvements

Product backlog

- List of prioritized things to be done
- Continuously updates (never complete)
- DEEP: Detailed appropriately, Emergent, Estimated, Prioritized

A product backlog is the aggregation of all the project backlogs pertinent to the product.

- Acceptance criteria: define scope of user story/feature/capability
- DoR: List of criteria for a user story/features/capability to be pulled from backlog
- DoD: Criteria to consider accepted

If stuck, reduce functionality and keep to the schedule, instead of trying to modify the schedule or add developers

- '9 pregnant women don't make a baby in a month'. Essentially, adding more developers rarely speeds up development in the short-term (can work long-term, but not relevant to this question)
- K eeping to deadlines in timeboxed iterations/sprints ensures a predictable delivery cycle and workload
- Lengthening development cycles reduces feedback from deliveries
- Reprioritization helps deliver on critical features while allowing those that appear less essential to be deferred
- Sustainable pace (No death marches or whatever)

Big upfront requirements, why Agile opposes them and why some are still beneficial

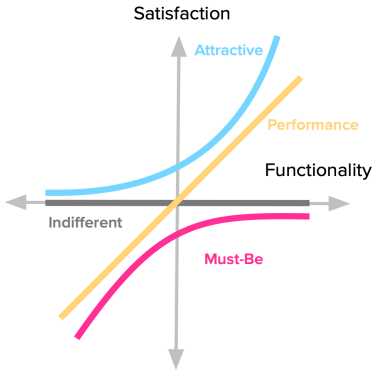
- Agile seeks adaptability, feedback, delivering value early

  - Collecting big upfront requirements delays the start of development
  - Predefined requirements/product specifications hinder adaptability & customer feedback
  - Embrace change
  - Working software over comprehensive documentation
  - Incremental + iterative approach
  - Reduce risk + waste
- Some upfront planning is useful
  - Define a product vision + goals, tech stack, etc. (As we did in Sprint 0)
  - Set architectural foundations. Concerns over scalability, security, and integration should be considered early on in a project
  - External dependencies, risk analysis, legal requirements, budgeting, stakeholders cannot be 'discovered iteratively'

Thinking on the video of the ping pong ball, establish a flow before optimizing resources.
Importance of having customer relations

- XP has a customer representative role within teams (This isn't very viable, but still you get the point)
- Agile teams should have some role that interfaces with customers.
- Not all customer feedback is great - Customers don't really know what they want much of the time

Kano model:
Graph satisfaction on y-axis, level of fulfillment on x-axis.



Projects/developers should be executed at a sustainable pace, sprints should be planned with cushions of leftover time for the team members in case of unforeseen obstacles
Additive vs Multiplicative complexity

- Additive complexity refers to the linear growth of software complexity that simply results from the growth of a codebase/product
- Multiplicative complexity is often the result of a poorly designed architecture; increasing dependencies make it such that every additional feature multiplies the number of interactions, increasing complexity further

Accepting vs liking change

- I wouldn't dwell too much on this, the concept is self-explanatory
- PMs don't really enjoy change, but as part of Agile teams they should be willing to expect and accept it

Weighted-shortest-job-first feature prioritization

- I vaguely remember there being a slide and a short exercise on this, there will be an exam question about it, I will probably not have read the slide because I don't wanna search for it, assuming the concept is self-explanatory and then realize I probably should have at least skimmed the slide.

Iterative development

- Small incremental improvements. Break work down into iterations (Sprints in Scrum), each should produce a shippable product increment. Deliver working software in small steps rather than delivering all at once
- Frequent feedback and adaptation. Sprint review, demos, user testing are some of the feedback loops involved in iterations.
- Continuous learning & improvement. Retrospectives and learning on each cycle improve processes and decision-making
- Value prioritization. Features are delivered based on business value, backlog is refined to reflect changing priorities
- Reduced risk & early problem detection
- In Scrum, iterations are defined as sprints (1-4 weeks) where a set of backlog items are developed and reviewed
- In Kanban, Continuous iteration occurs through small batch delivery and WIP limits
- In XP, there are frequent releases (sometimes multiple per day) with constant refactoring and feedback

Requirements vs Scenarios

- Requirements cover the complete scope of functionalities, user stories cannot replace these, as you can't fully define a specification with them
- Sometimes requirements are necessary, think of legal requirements, architectural, performance, etc.

Scrum ceremonies

- Daily meeting
- Sprint demo
- Sprint planning
  - Planning Poker

Shared code ownership & cross-functional teams

- In shared code ownership, the entire team is responsible for all code, instead of having individual devs responsible for/'owning' sections
- Cross-functional teams refers to teams with knowledge from different areas. The professor speaks of wanting 'T-shaped' team members in Volvo Cars, with expertise in one specific area but at least surface-level knowledge on the rest of the areas relevant to the project
- Shared code ownership can improve code quality by bringing more eyes to each piece of code, speed up development by reducing bottlenecks on specific devs, encourages knowledge sharing and makes refactoring easier (no approval from a specific dev)
- Inconsistent coding styles, merge conflicts & overwrites, reduced sense of ownership/accountability and a need for strong communication are some drawbacks of shared code ownership
- Cross-functional teams can speed up delivery by reducing handoffs between teams, they improve collaboration and flexibility and are able to consider more aspects of a product, improving quality
- Cross-functional teams can face challenges with coordination, conflicting priorities, skill gaps and scaling (managing cross-functional teams in large organizations is complex)

Personal thoughts on pair programming

- Personal preferences and different work ethics: should not be forced onto all developers
- XP likes it

Test-Driven Development - Know 5 steps in detail, be able to provide examples (our own work is relevant, mention the Unit tests we wrote with JUnit for our backend, be aware that we wrote those tests after implementing the feature though so maybe google a better example and write it down here please uwu)

1. Quickly add a test
2. Run all tests, see the new one fail
3. Make a small change
4. Run all tests again, see them succeed
5. Refactor & remove duplication

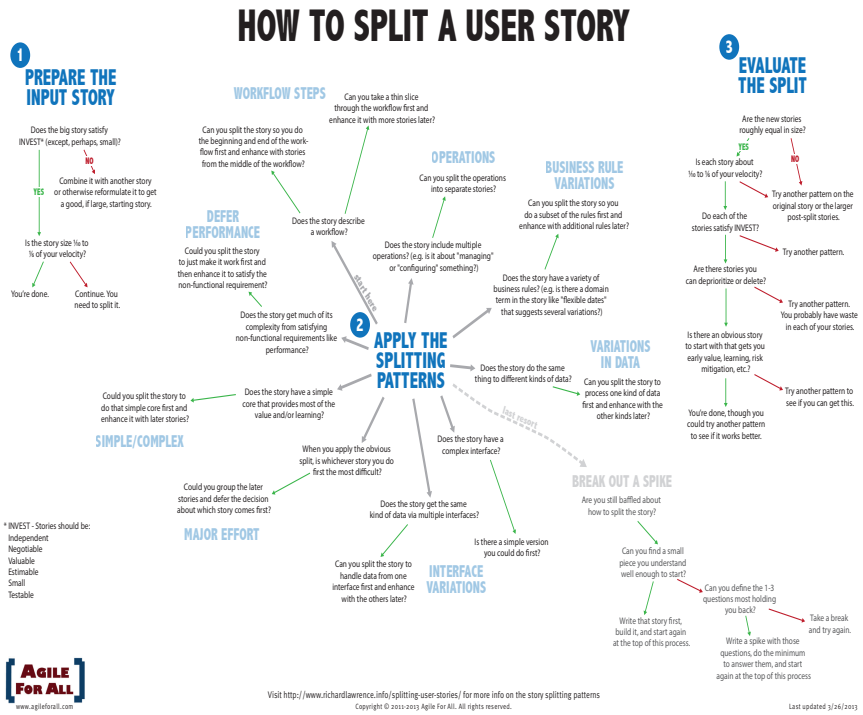
Extreme Programming (XP)
Practices - Please list them down here if you have the slide with them on you :D. We need to be able to explain them listing them by heart is not necessary though
I am starting to believe that this subject is not worth the 3 credits I'll be getting at my home uni
Also, someone tell the student representatives that is the professor wants to forego the exam but also ensure that we actually learn the contents while keeping the primary focus on the project then having us answer the questions to the exam by sections as group assignments throughout the project would be a very effective way of achieving this while also being much less of a pain in the big O for all of us.

- Pair Programming
- System Metaphor
- Shared Code Ownership
- Small Releases
- Simple Design
- 40 Hour Week
- Planning Game
- Coding Standards
- Test first
- Continous Integration
- Customer On-site
- Refactoring
- Product Owner: customer representative, owns the product backlog, ensures team is building the right thing, prioritizes items in the product backlog, must also be able to negotiate and say "no" to the customer
- Scrum Master: Servant Leader ensures good working environment for the development team and removes impediments, coaches, initializes the agile ceremonies
- Development Team: self-organized cross-functional team, commits to delivering items in the sprint backlog, T-shaped team

Product backlog, why it should be detailed, what a DEEP backlog is, why we want ONE product backlog
Difference between project and product
User stories, INVEST format

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

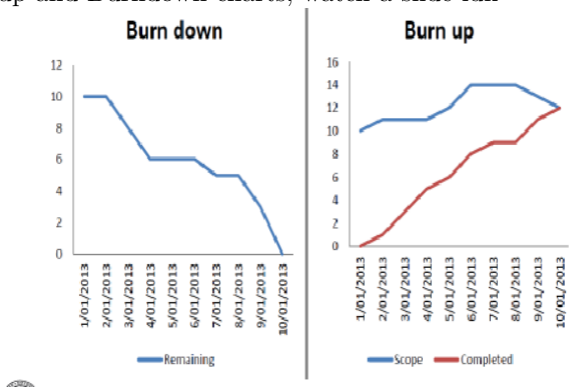
Be able to break down a functionality into around 10 user stories:



Functional vs Non-functional requirements

- Functional: Define what the system should do Independent, Negotiable, Valuable, Estimatable, Small, Testable
- Non-functional: Define the constraints and how the system should behave Bounded, Independent, Negotiable, Testable

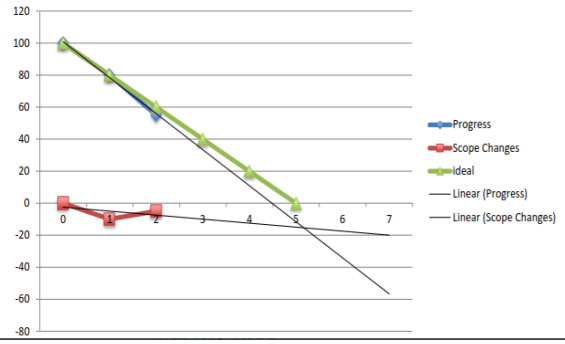
Enable stories refer to technical needs, architectural improvements, or infrastructure required to deliver normal user stories. Normal user stories describe functionality/scenarios that deliver value to the customer
Capacity expresses an amount of work that can be done given a space of time, velocity refers to the amount of work that has been done in a space of time
Be able to explain a proposed-job allocation as PO or SM
Story points are estimations of the amount of effort (sometimes time) required to complete an element in a backlog (usually user story/task)
Sprint 0 is a preliminary sprint often added on Scrum methodologies for dealing with architectural/tech stack planning, definitions of DoD, product vision, etc. (all those small big upfronts we all hate)
Burnup and Burndown charts, watch a slide idk



Case #2

- PO planned the release to include 20 items estimated to 100 SP. The team was usually able to deliver 20 SP per Sprint, thus PO assumed the release will be ready in 5 sprints.
- Before the 2nd Sprint, it turned out that the team was able to deliver 4 items (worth 20 SP) in the first sprint. However, PO added 2 items to PB worth 10 SP.
- At the beginning of the 3rd Sprint it turned out that the team was able to deliver 25 SP in the second sprint. PO also removed one item from PB worth 5 SP.
- *Do you think that they will deliver in 5 Sprints?*

Case #2 burndown chart

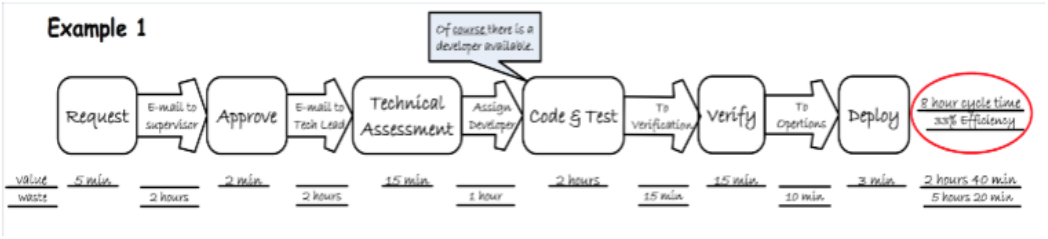


Lean originates in manufacturing, Scrum in software dev.
Lean principles:

- Eliminate waste:
  Tool:

  1. Inventory/partially done work
  2. Extra processing - paperwork
  3. Over production - unused features
  4. Transportation - task switching
  5. Waiting - staffing, approval, review, excessive reqs., testing, deployment
  6. Motion - handovers, finding answers to questions
  7. Defects

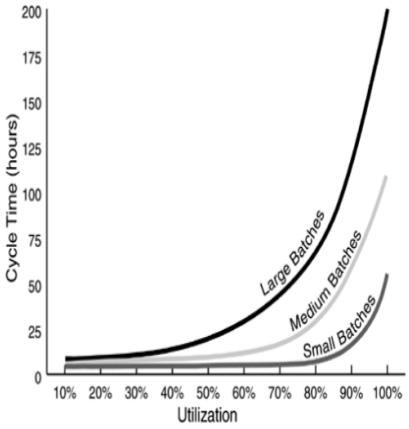
  Tool: Value Stream Mapping



- Amplify learning - Incorporate feedback from dev, tests, retrospectives, etc. into development process
  Tool: Feedback
  Tool: Short iterations; increase control, help sync devs and customer, force decisions to be made
  Tool: Synchronization: Daily scrum, build, system test, weekly push to master, weekly meetings (scrum-of-scrum)
- Decide as late as possible
  Keep options open as long as is practical, allows you to survey landscape, delay detailed decisions, breadth-first development allows managing risk
  Tool: Last responsible moment - that in which failing to make a decision eliminates an important alternative
- Deliver as fast as possible: Limit WIP by delivering fast, faster delivery increases customers' business flexibility, prevents them changing their minds, keeps resources from getting tied up in WIP. The faster you deliver, the longer you can delay decisions
  Tool: Pull system; every team member knows how to contribute best, everyone sees what's going on, what needs to be done, what problems exists, what progress is being made
  Tool: Queuing theory; Cycle time is the measurement of a queue, minimize cycle time. Consider arrival rate, service rate, resource load

Bach size impact on cycle time

This works just like a traffic jam at rush hour - go above 85 percent capacity and gridlock is inevitable.



How queues work?

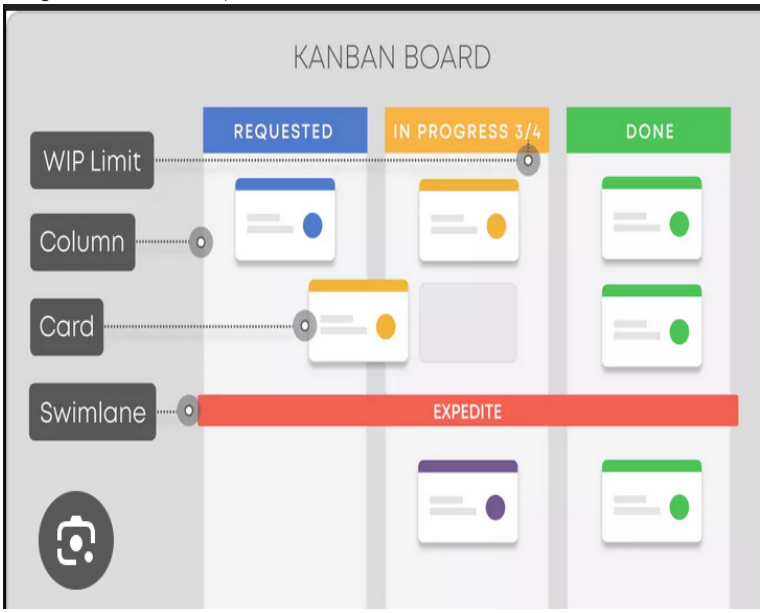
- To increase throughput, fix the current bottleneck that is slowing things down.
- As variability (in arrival time or processing time) increases, cycle time will increase.
- As batch size increases, variability in arrival and processing time increases, and therefore cycle time will increase.
- As utilization increases, cycle time will increase nonlinearly.
- Continuous flow requires a reduction in variability.
- Variability may be reduced by an even arrival of demand, small batches, an even rate of processing, and parallel processing.
- Decreasing variability early in the process has larger impact than decreasing variability late in the process.

Tool: Cost of delay; from an economic model, show how revenue and market share affect profits in relationship to delays

- Empower the team: Move decisions to the lowest level possible while developing the capacity of those people to make decisions wisely
  Tool: Leadership
- See the whole: Software systems are not just the sum of their parts, but the product of their interactions. Avoid local sub-optimization
  Tool: Contracts

Lean process:

1. Identify value
2. Map value streams, visualize with Kanban board



3. Create flow, limiting WIP
4. Establish pull from common prioritized backlog, clear rules for moving tasks in Kanban board
5. Seek perfection, constantly remove waste

Agile focuses on local-level communication, Lean advocates holistic view of delivery change, XP advocates 'decide now, change later', Lean defers commitment Global optimization
Value Stream Mapping + Know how to improve the efficiency of a given development process
'Last responsible moment' for decision making
Queuing theory

- Arrival rate
- Service rate
- Resource load

How small vs large batches pass through the process, diagram showing how batch size affects cycle time.
Difference between Lean, Agile, Kanban, XP, etc.
Kanban

- Pull system that visualizes workflow and uses WIP to facilitate flow
- The Kanban board
- Kanban phases

Kanban principles

- Start with what you do now
- Agree to pursue incremental evolutionary change
- Respect current process, roles, responsibility, titles
- Leadership at all levels

Kanban Practices

- Visualize workflow to understand & improve
- Limit WIP
- Manage flow
- Make policies explicit
- Implement feedback loops
- Improve collaboratively, evolve experimentally - whole team shares theory on why small change helps

Scrumban, which aspects of each can be integrated in what scenarios. Read through Lecture 4 table on SCrum vs Kanban
Lean focuses on frequent deliveries and quality, XP deals with these aspects indirectly
Kaizen - Continuous improvement
Agile leadership, the triangle of FUNCTIONALITY, TIME, COST (+ QUALITY)
A manager should provide for their teams:

- Autonomy
- Master
- Purpose

Concept of less is more, one example is not focusing on elements of little value, pick off the 'low-hanging fruit' that offers the most instant value rather than prototyping and planning endlessly
Be able to decide & justify what features you would promise to a customer given some tasks, velocities, etc.
forming, storming, norming, performing
Servant leader vs regular leader. Encourages ideas rather than giving answers, builds trust, is people-focused. Encourages team members to find own answers to questions, enables them and provides required resources.
Levels of authority

1.

Look at the video on the submarine captain, reflect on leadership styles, what we can gain from each of them. Products should be busy, rather than team members
Concept of the sphere of influence, don't focus on things you have little impact on
Cynefin framework. Categorizes problems into domains

- Obvious - identify issue, apply a proven solution
- Complicated - analyze problem, consult experts, find best solution
- Complex - probe, sense, respond: experiment, gather feedback, adapt
- Chaotic - Act, sense, respond: Take immediate action to stabilize, then find solutions
- Disorder - Break problem down into the other four domains

Types of reporting, state vs. prognosis. How would you report a status to your manager?
Change curve
How commitment impacts predictability and productivity. Why teams should not overcommit (Agile advises working at a sustainable pace, no death marches, avoid burnout, etc.)
5 dysfunctions of a team - Will not be asked I can't be arsed
Random stuff about guest lecture: How would you work/what tools would you use if you worked in a team like the lecturer's?
Unit, Integration, System, Acceptance testing

- Unit testing refers to tests for specific elements/methods/functions of an interface
- Integration testing refers to testing that involves bringing together 2 or more parts of a system and testing that they work together correctly
- System testing involves building a software application as a whole and testing it against a set of requirements
- Acceptance/end-user testing refers to determining whether an application meets end-users' approval
- Software in the Loop testing refers to testing software in a simulated environment before integrating with real hardware
- Hardware in the Loop testing refers to hardware testing in a real-time simulated environment

A regression test suite is the selection of test cases used to ensure added code/features do not cause a regression (error/loss of functionality) A regression test then is a test in the regression test suite
Principles of continuous integration

-

Single source of truth/information for the software product, defined pulling and merging procedures
Version vs revision vs branch vs baseline

-

Test prioritization

- Usually done statistically based on the set of tests that commonly fail in relation to modifications in the given component of the product.

Delivery vs deployment

- Deployment refers to ensuring the product is functional & can be downloaded/accessed instantly 'with one click', delivery refers to the actual ct of the customer receiveing and installing software

Continuous Delivery vs Continuous Deployment

-

Deployment practices (Name at least 3)

1. Dark launching
2. Feature flag

3. Rollouts

DevOps, conflicts, why prod and dev. environments should not be different

-

SAFe (Scaled Agile Framework) Principles

1.

SAFe
Implementation roadmap
Group manager vs Team manager

-

Product Increment (PI) Planning

-

Agile Release Train

-

Develop on a cadence, release on demand
Solution train engineers, solution managers, system architects

-

Architectural runway/runaway

- The scope of features/functionalities that can be built on the product on the current architecture (the runway)

Conflict between Product and Solution management (PM and System Architect)

-

Decentralizing decisions between interfaces

-

Shared services & system team outside ART

-

Backlogs and roles on different levels of the framework. Epics are broken down into Capabilities, Capabilities into features, features into stories
Define 'the visions' of a company given its product (what kind of snake oil salesman shit is this)
product refers to an ongoing offering/service that provides value to userse, it is continuously improved, sold, maintained.
A project refers to a temporary effort to create a specific deliverable within a timeline, scope, budget