

CSC 171 – Module 8

Agile Practices

Definition of Ready

- The team makes explicit and visible criteria (generally based on the INVEST matrix) that a user story must meet prior to being accepted into the upcoming iteration.
- Expected Benefits
 - Avoids beginning work on features that do not have clearly defined completion criteria
 - Provides the team with an explicit agreement allowing it to “push back” on accepting ill-defined features to work on
- The stories at the top of the Product Backlog that will be pulled into the Sprint Backlog or Kanban board must be Ready.
 - The Product Owner is responsible for putting the features and stories in the product backlog
 - The team must work with the Product Owner during Backlog Refinement to get the stories into actionable shape

Give an example backlog item you have worked on that is not ready.

Definition of Done

- Definition of Done (DoD)
 - The team agrees on a list of criteria which must be met before a product increment (or a user story) is considered “done”.
- "Done" means releasable
- Expected benefits
 - Provides a checklist that guides pre-implementation activities such as discussion, estimation, and design
 - Limits the cost of rework once a feature has been accepted as “done”
 - Limits the risk of misunderstanding and conflict between the development team and the customer or product owner
- Consists of three components
 - Functional requirements
 - Non-functional requirements
 - e.g., Availability, Maintainability, Performance, Reliability, Scalability, Security, Usability
 - Quality standards
 - e.g., Defined coding standards, Test- Driven Development, Unit Test Coverage, No Defects/Known Defects, Technical Debt, Design Principles etc.

Give an example of DoD you used.

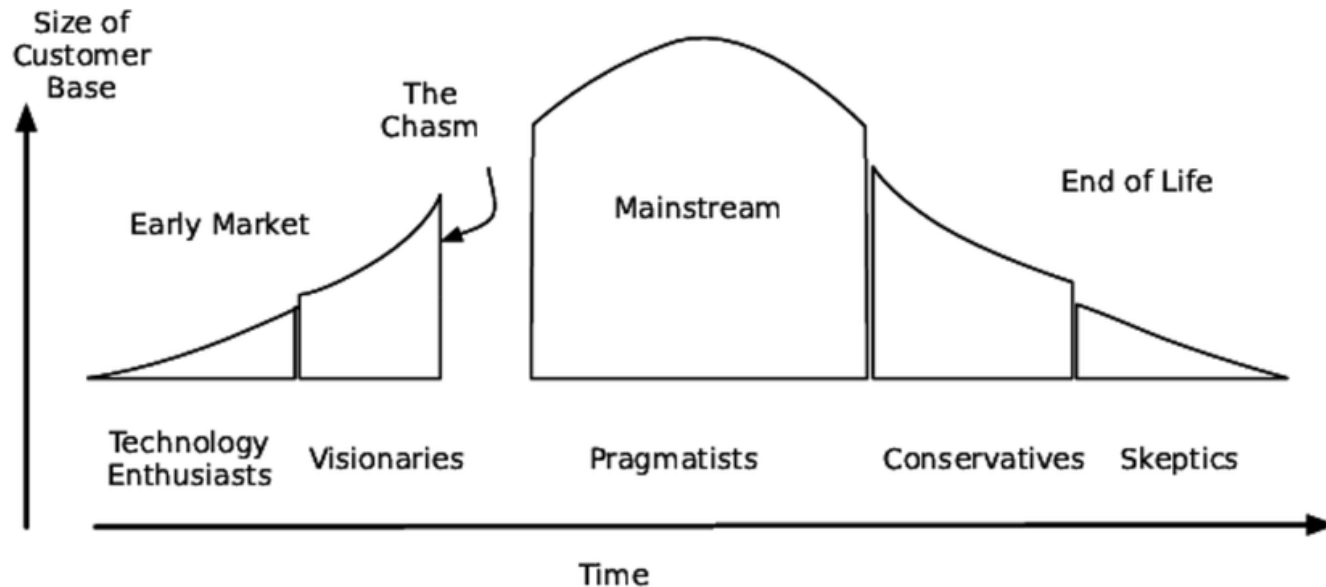
DoD Example

- All the code is checked in.
- There are automated unit tests, and they are checked in.
- There are automated system tests at the API level, and they are checked in.
- The checked in code and tests have been reviewed and approved by others.
- All the tests pass.
- The documentation for the feature is complete.
- The story meets all functional requirements.
- The story meets all non-functional requirements.
- There is no increased technical debt.
- The product owner has accepted the user story.

Given-When-Then

- A template to guide the writing of acceptance tests for a user story
 - Acceptance tests are from the user's point of view
- Given-When-Then
 - (Given) some context
 - (When) some action is carried out
 - (Then) you should see some set of observable consequences
- Examples
 - Test 1
 - Given a Zoom meeting is configured to mute participants upon joining
 - When a participant joins the meeting
 - Then that participant is automatically on mute
 - Test 2
 - Given a Zoom meeting is **not** configured to mute participants upon joining
 - When a participant joins the meeting with his/her audio on
 - Then that participant is not on mute
- Too many tests indicate story likely too large

How Much Quality is Needed?



What Customers Care About at Different Times:

Enthusiasts:	Visionaries:	Pragmatists:	Conservatives:	Skeptics:
<ol style="list-style-type: none"> 1. Time to Release 2. Low Defects 3. More Features 	<ol style="list-style-type: none"> 1. Time to Release 2. More Features 3. Low Defects 	<ol style="list-style-type: none"> 1. Low Defects 2. Time to Release 3. More Features 	<ol style="list-style-type: none"> 1. Low Defects 2. More Features 3. Time to Release 	<ol style="list-style-type: none"> 1. Low Defects 2. More Features 3. Time to Release

For a product you care about, which phase is it in?

How do you order time to release, low defects, and more features?

Refactoring

- Improving the internal structure of an existing program's source code, while preserving its external behavior.
- Refactoring in the absence of safeguards (e.g., automated regression testing suite) against introducing defects is risky.
- Refactoring does not mean
 - Rewriting code
 - Fixing bugs
 - Improve observable aspects of software (e.g., its interface)
- Expected benefits
 - Improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance
 - Helps code understanding
 - Favors the emergence of reusable design elements (such as design patterns) and code modules

What kind of refactoring have you done in your projects?

Test at All Levels

- Types of testing
 - Automated unit testing
 - Automated and exploratory feature (story) testing
 - Automated feature set testing
 - Automated smoke testing
 - Automated system testing from API
 - System testing from GUI, possibly with some automation
 - Exploratory testing at the system level
- Practices
 - Automate as you proceed
 - Test-driven development (TDD) is when a developer writes a test that fails (red), creates just enough code to make it pass (green), and, when it's time to add to that code, refactors to clean any messes.
 - Behavior-driven development ([BDD](#)) is for specification by example
 - Extension of TDD, tests of any unit of software should be specified in terms of the desired behavior of the unit.
 - Behavior specification, often uses given-when-then template
 - Ubiquitous language, a (semi-)formal language that is shared by all members of a software development team
 - Acceptance test-driven development (ATDD) involves team members with different perspectives (customer, development, testing) collaborating to write acceptance tests in advance of implementing the corresponding functionality.
 - Extension of TDD
 - Three amigos, writing acceptance tests before developers begin coding
 - Often uses given-when-then template

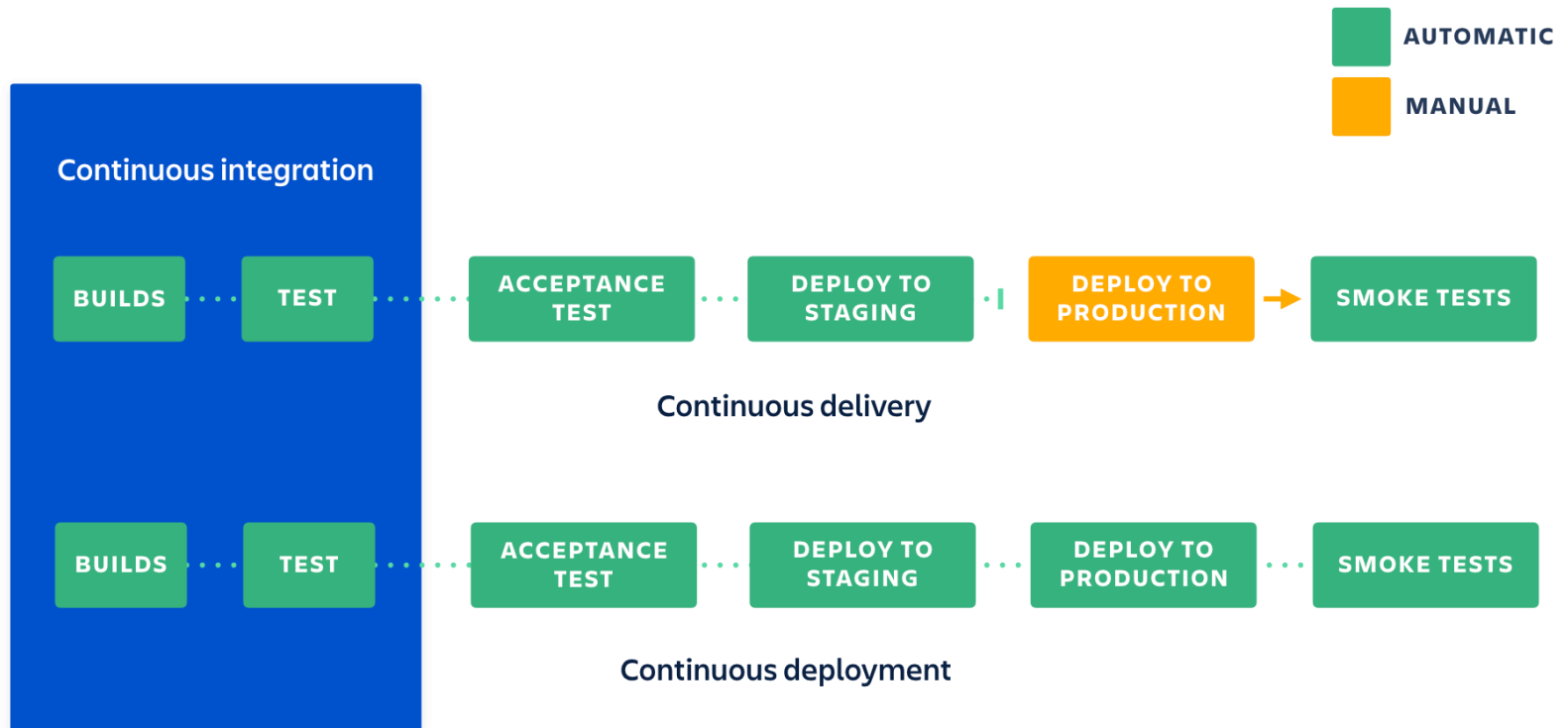
Continuous Integration

- A software development practice where members of a team integrate their work frequently (e.g., at least once per day per person). Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.
- Benefits
 - Reduced integration problems and risks
 - Easier to discover and fix defects
 - Faster product development
- Signs of use
 - Use of a version control tool
 - An automated build and product release process
 - Instrumentation of the build process to trigger automated tests “every time any change is published to version control”
 - In the event of even a single test failing, alerting the team of a “broken build”

Continuous Deployment

- An extension of continuous integration. Product deployment steps are automated. Once new product increment meets the release criteria, product is deployed automatically.
- Benefits
 - Reduced lead time
 - Early return on investment
 - Early feedback from customers
- Potential costs
 - Infrastructure to ensure product quality
 - Infrastructure to easily roll back new features

CI & CD [5]



Frequent Releases

- An agile team frequently releases its product into the hands of end users, listening to feedback, whether critical or appreciative.
 - Release frequency varies according to the technical and business aspects of the context
 - Continuous deployment can be used for totally digital products
- Expected benefits
 - It mitigates the well-known planning failure mode of discovering delays very late
 - It validates the product's fit to its market earlier
 - It provides earlier information about the quality and stability of the product
 - It allows for a quicker return on the economic investment into the product
- How to release partially completed feature sets
 - Consider flags for enabling features
 - Consider the ability to roll back any changes
- Types of releases
 - Internal releases, which are used internally to understand the product direction and for demonstrations
 - Incrementally grown releases, which are used to bring new customers onboard
 - "Real" releases, which are taken by all customers

Pairing, Swarming, Mobbing

- Pairing
 - Two people work together on one item on one machine with one keyboard.
 - The person typing is called the driver. The person looking is the navigator. Often, the two people change places on a frequent basis (e.g., every 15 minutes).
- Swarming
 - The team works together on one story, but each person contributes based on his or her expertise.
 - The team has a cadence: work for a defined duration (e.g., an hour) check in with each other, scatter to work for another duration, check in, repeat until the item is done.
 - If one person is done before the others, that person offers his or her services—possibly as a reviewer—to the rest of the team.
 - The team's WIP limit is 1.
- Mobbing
 - The entire team works together on one keyboard. The team changes drivers on a frequent basis.
 - The team has a WIP limit of 1.
- Benefits of pairing, swarming, and mobbing
 - The team limits its WIP.
 - The team can learn together.
 - The team has multiple eyes on small chunks of work, so it gets the benefit of continuous review.
 - The team collaborates, so it reinforces its teamwork.
- Pairing, swarming, and mobbing all use the ideas of flow efficiency to finish the team's work.

Sustainable Pace


- The team aims for a work pace that they would be able to sustain indefinitely.
- One of the agile principles
- "I have never seen teams deliver under sustained pressure." [1]
- When a team is pressured to deliver instead of think, the team creates defects.
- Overtime tends to mask schedule, management, or quality deficiencies.

User Story Practices

- Role-feature-benefit template
 - As a (who wants to accomplish something)
 - I want to (what they want to accomplish)
 - So that (why they want to accomplish that thing)
- Three C's
 - Components of a user story
 - Card, Conversation, and Confirmation
- INVEST
 - A set of criteria to assess the quality of a user story
 - Independent (of all others)
 - Negotiable (not a specific contract for features)
 - Valuable (or vertical)
 - Estimable (to a good approximation)
 - Small (to fit within an iteration)
 - Testable (in principle, even if there isn't a test for it yet)

Personas

- "A designer who tries to please everybody ends up pleasing nobody, because too many compromises kill the product's integrity." [3]
- Persona defines the person who is going to use the product and includes
 - Demographics: name, age, photo, ...
 - Goals: What are they trying to do?
 - Motivations: Why are they trying to do it?
 - ...
- Personas are not static, and they develop in sync with the product
- An example [4]

		
Daughter / Mom Marla		
Age 40-55		
Key Tasks		
<ul style="list-style-type: none">• Capture moments, take pictures• Share pictures with family and friends via social media and email• View pictures that are shared by family and friends• Occasionally print pictures and put them on display at home or office		
Motivations	Frustrations	Opportunities
Remember meaningful event Relive events Evoke warm memories	Can't find pictures when I want to Too many pictures, no organization	Create organization of pictures as a by product of everyday events

Story Splitting

- When to split a user story?
 - It is too large to fit within an iteration
 - It is small enough, but the planned iteration does not have too much room left
 - More accurate estimate is needed
- How to split a user story?
 - Split across data boundaries
 - Split along operational boundaries
 - Split to remove cross-cutting concerns
 - Split to separate the functional and nonfunctional aspects
 - Split stories of mixed priority
 - Split by persona
 - Break out a spike
- What to avoid?
 - Split along implementation layers
 - Add related changes to an appropriately sized feature

Story Splitting Guidelines

- Split across data boundaries
 - Example: As a user, I can enter grades for a student, so that...
 - As a user I can enter input attendance grade for a student, so that...
 - As a user, I can enter assignment grades for a student, so that...
 - As a user, I can enter team project grades for a student, so that...
 - As a user, I can enter mid-term exam grade for a student, so that...
 - As a user, I can enter final exam grade for a student, so that...

The screenshot shows a 'Grade Calculator' application window. It contains a table with the following data:

Assignment Group	Group Weight	Points Earned	Points Possible	Percent of Points Earned
Attendance	1%	0	0	0%
Assignments	19%	0	0	0%
Team Project	25%	0	0	0%
Mid-Term Exam	25%	0	0	0%
Final Exam	30%	0	0	0%

Below the table are five buttons: 'Input Attendance', 'Input Assignment', 'Input Team Project', 'Input Mid-Term Exam', and 'Input Final Exam'. At the bottom, it displays 'Total Grade: 100%'.

Story Splitting Guidelines – Cont.

- Split along operational boundaries
 - Example: Splitting across the CRUD operations (Create, Read, Update, Delete)
 - As a student, I want to be able to post my assignment submission, so that ...
 - As a student, I want to be able to view my assignment submission, so that ...
 - As a student, I want to be able to update my assignment submission, so that ...
 - As a student, I want to be able to delete my assignment submission, so that ...
 - Example: The story of building the following UI can be split into
 - Multiple data input user stories
 - Data display grid
 - Data calculation

The screenshot shows a 'Grade Calculator' application window. It contains a table with the following data:

Assignment Group	Group Weight	Points Earned	Points Possible	Percent of Points Earned
Attendance	1%	0	0	0%
Assignments	19%	0	0	0%
Team Project	25%	0	0	0%
Mid-Term Exam	25%	0	0	0%
Final Exam	30%	0	0	0%

Below the table, there are five input buttons: 'Input Attendance', 'Input Assignment', 'Input Team Project', 'Input Mid-Term Exam', and 'Input Final Exam'. At the bottom, the 'Total Grade' is displayed as '100%' in green text.

Story Splitting Guidelines – Cont.

- Split to remove cross-cutting concerns (e.g., error-handling, logging, security etc.)
 - Example: screen with access control
 - Develop the screen without access control
 - Add access control to the screen
- Split to separate the functional and nonfunctional aspects
 - Example: “Make it work, then make it faster.”
 - Example: Build a simple UI first, then enhance it with better usability
- Split stories of mixed priority
 - Example: As a user, I am required to log into the system.
 - If the user enters a valid username and password, she is granted access. (high priority)
 - If the user enters an invalid password three times in a row, she is denied access until she calls customer service. (high priority)
 - If the user is denied access, she is sent an email stating that an attempt was made to use her account. (low priority)

Story Splitting Guidelines – Cont.

- Split by persona
 - Example: As a user, I can log in LMS, so that...
 - As an administrator, I can log in LMS, so that...
 - As an instructor, I can log in LMS, so that...
 - As a student, I can log in LMS, so that...
 - ...
- Break out a spike
 - A story may be large because it is poorly understood
 - Example: As user, I can zoom in and out on the map, so that ...
 - Investigate zooming functionality provided in third party UI library
 - Implement zooming product functionality
- What to avoid?
 - Splitting a user story along implementation layers
 - Example: user interface, business logic, database user stories
 - Adding related changes to an appropriately sized feature
 - Example: fix a defect that is not part of the user story in the same code base

References

- [1] Create Your Successful Agile Project, Johanna Rothman, Pragmatic Programmers LLC, 2017. ISBN:9781680502602
- [2] Agile Estimating and Planning 1st Edition; Author: Mike Cohn; ISBN-13: 978-0131479418; ISBN-10: 9780131479418
- [3] Personas, Agile Alliance,
<https://www.agilealliance.org/glossary/personas>
- [4] Successful User Experience: Strategies and Roadmaps, Elizabeth Rosenzweig, Morgan Kaufmann, 2015
- [5] Continuous Integration vs. Continuous Delivery vs. Continuous Deployment, <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>