جامعة الأمير سطام بن عبد العزيز
Prince Sattam Bin Abdulaziz University

Chapter 5

Requirements Specification for Agile Methods

Dr. Mariem Haoues
Dr. Taher Labidi

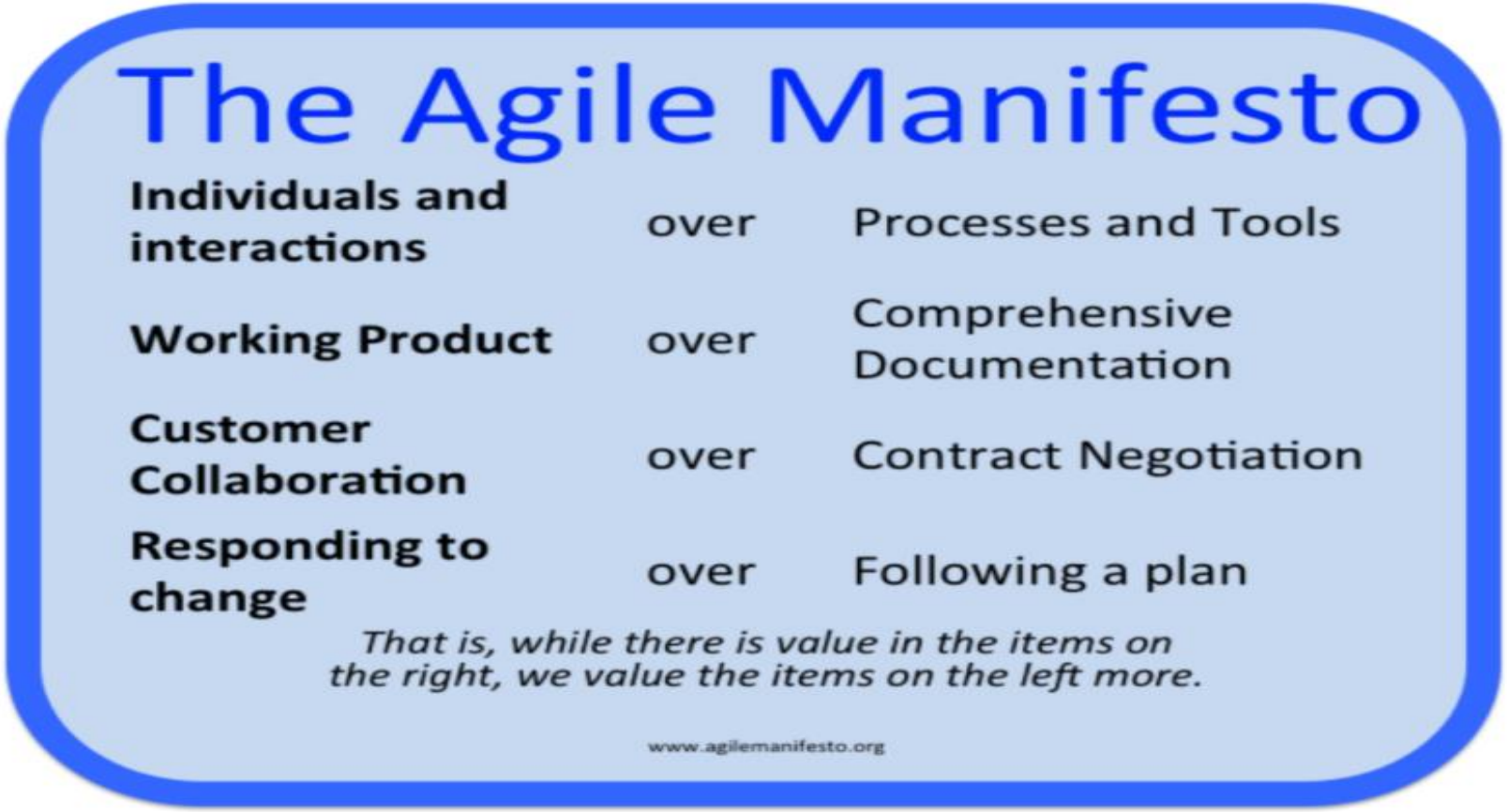
Department: Software Engineering

Academic Year
2022-2023

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:



Agile Principles (1)

01 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

02 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

03 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

04 Business people and developers must work together daily throughout the project.

05 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

06 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

07 Working software is the primary measure of progress.

08 The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

09 Continuous attention to technical excellence and good design enhances agility.

10 Simplicity—the art of maximizing the amount of work not done—is essential.

11 The best architectures, requirements, and designs emerge from self-organizing teams.

12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agile Principles (2)

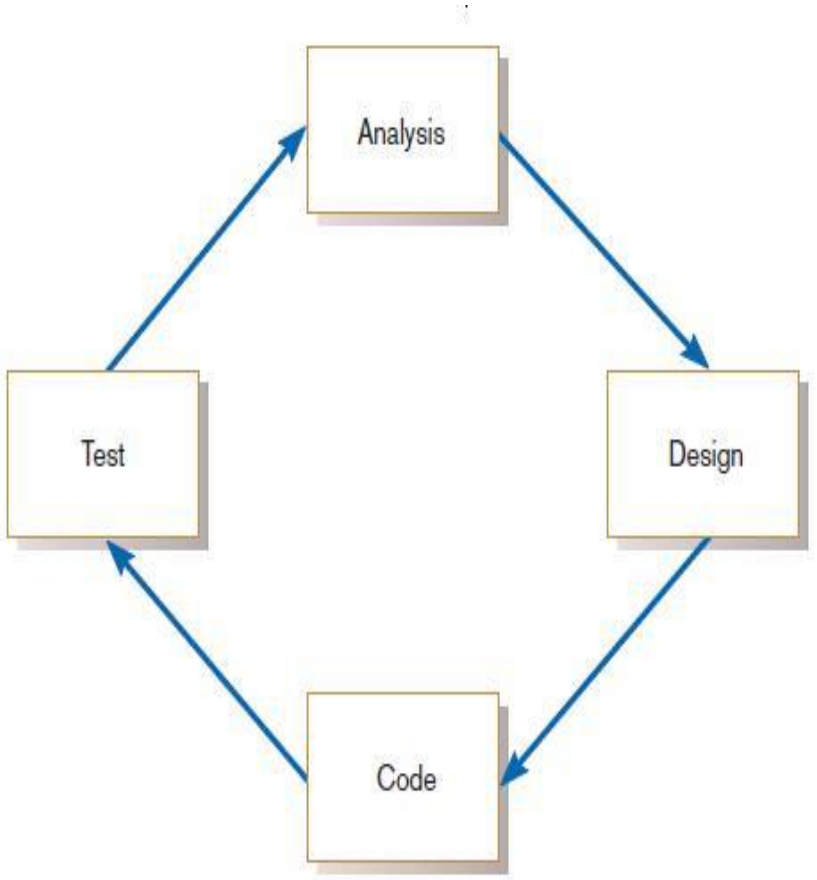
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Business people and developers must work together daily throughout the project
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Agile Principles (3)

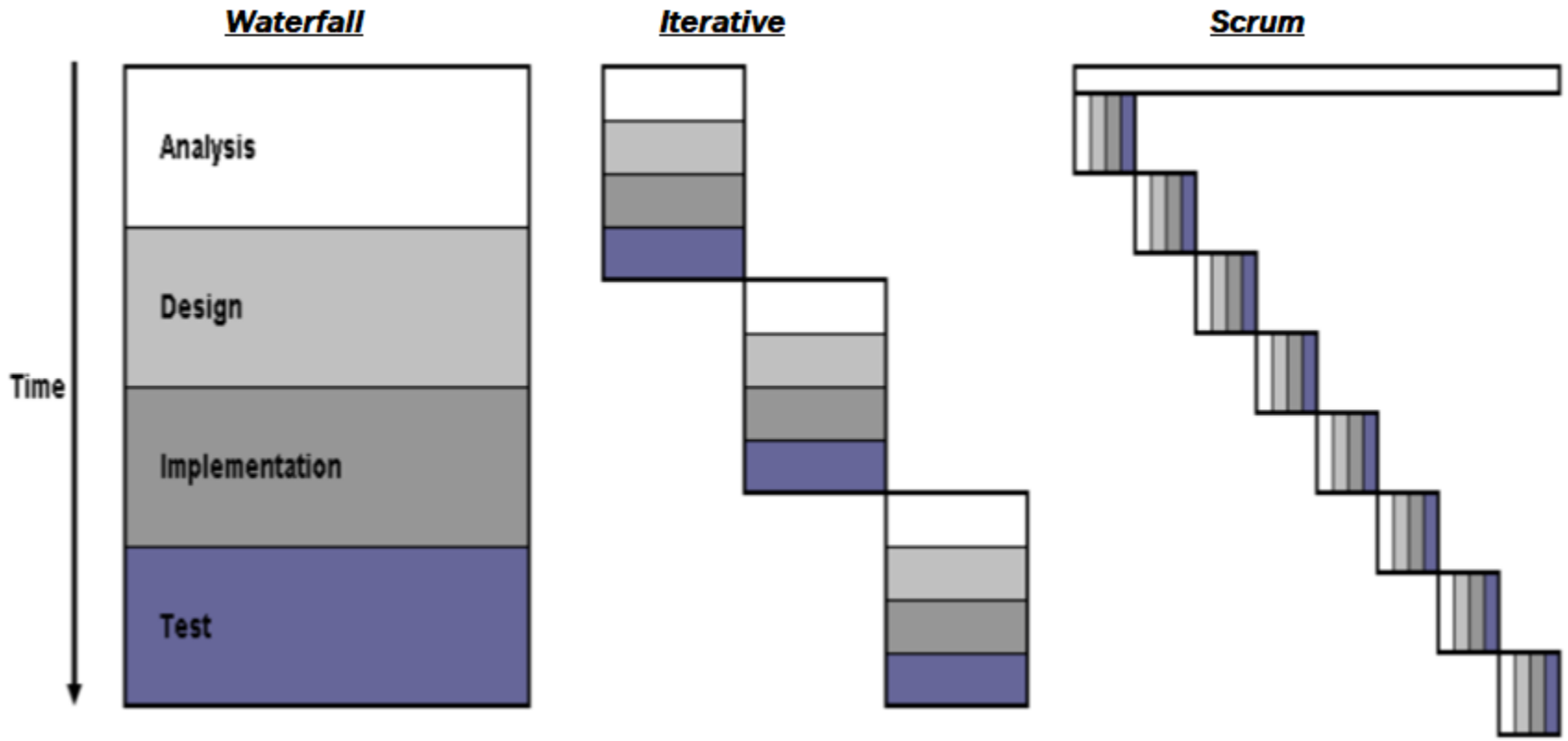
- Working software is the primary measure of progress
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity--the art of maximizing the amount of work not done--is essential. “Do the simplest thing that could possibly work.”
- 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 Development Life Cycle (SDLC)

- Analysis – an organization's needs are identified, analyzed, prioritized, and arranged and the requirements are structured
- Design – a description of the recommended solution is converted into system specifications
- Implementation – the system is coded
- Test and Maintenance – the system is tested, systematically repaired and improved, installed and supported in the organization



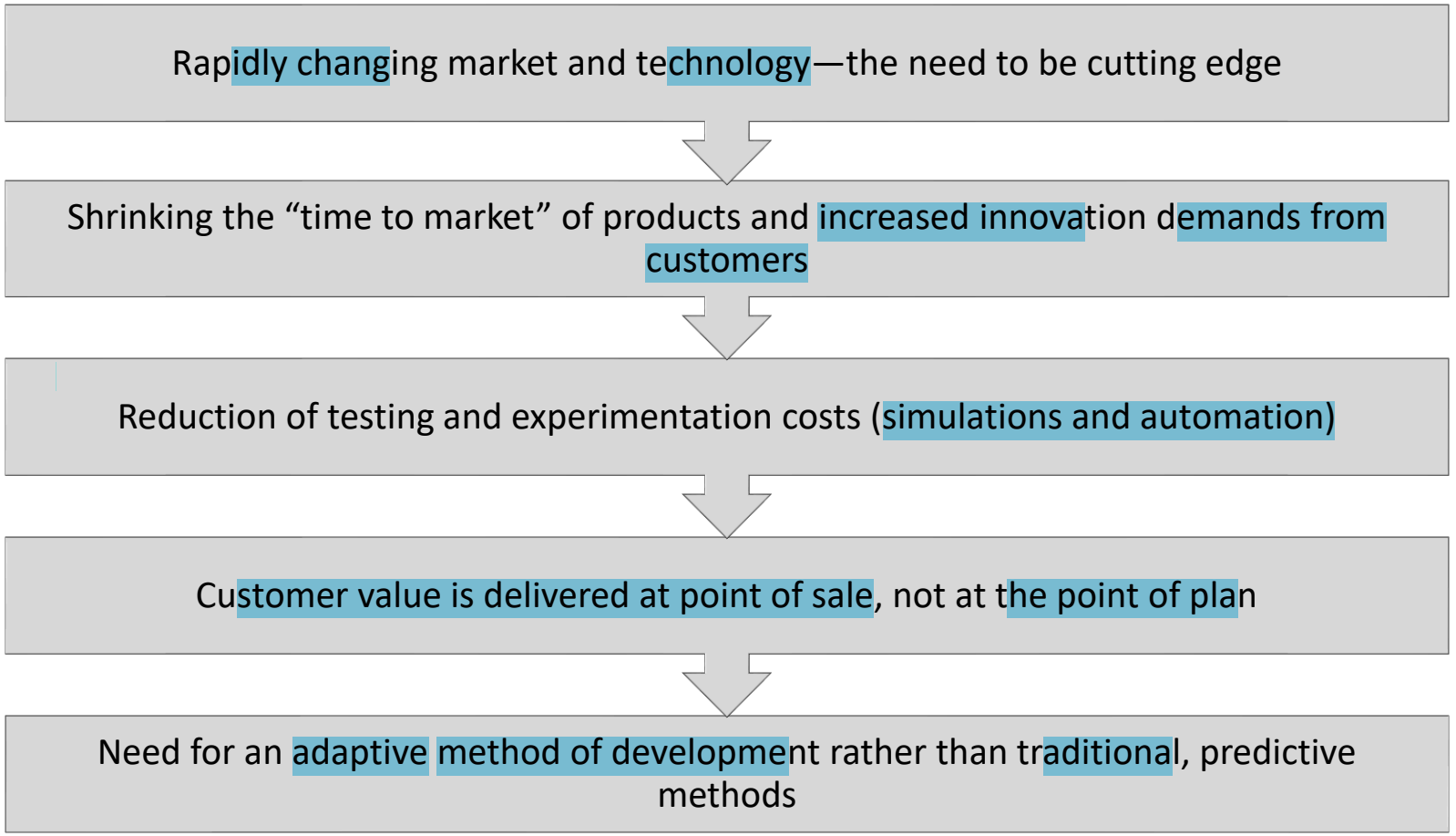
Comparison of Waterfall, Iterative, and Iterative/incremental (Scrum) Development Cycle



Traditional SDLC - Waterfall

- Problems with Waterfall Approach:
  - Feedback ignored, milestones lock in design specs even when conditions change
  - Limited user involvement (only in requirements phase)
  - Too much focus on milestone deadlines of SDLC phases to the detriment of sound development practices
  - Ineffective with unpredictable or dynamic requirements

The need for Agile ?

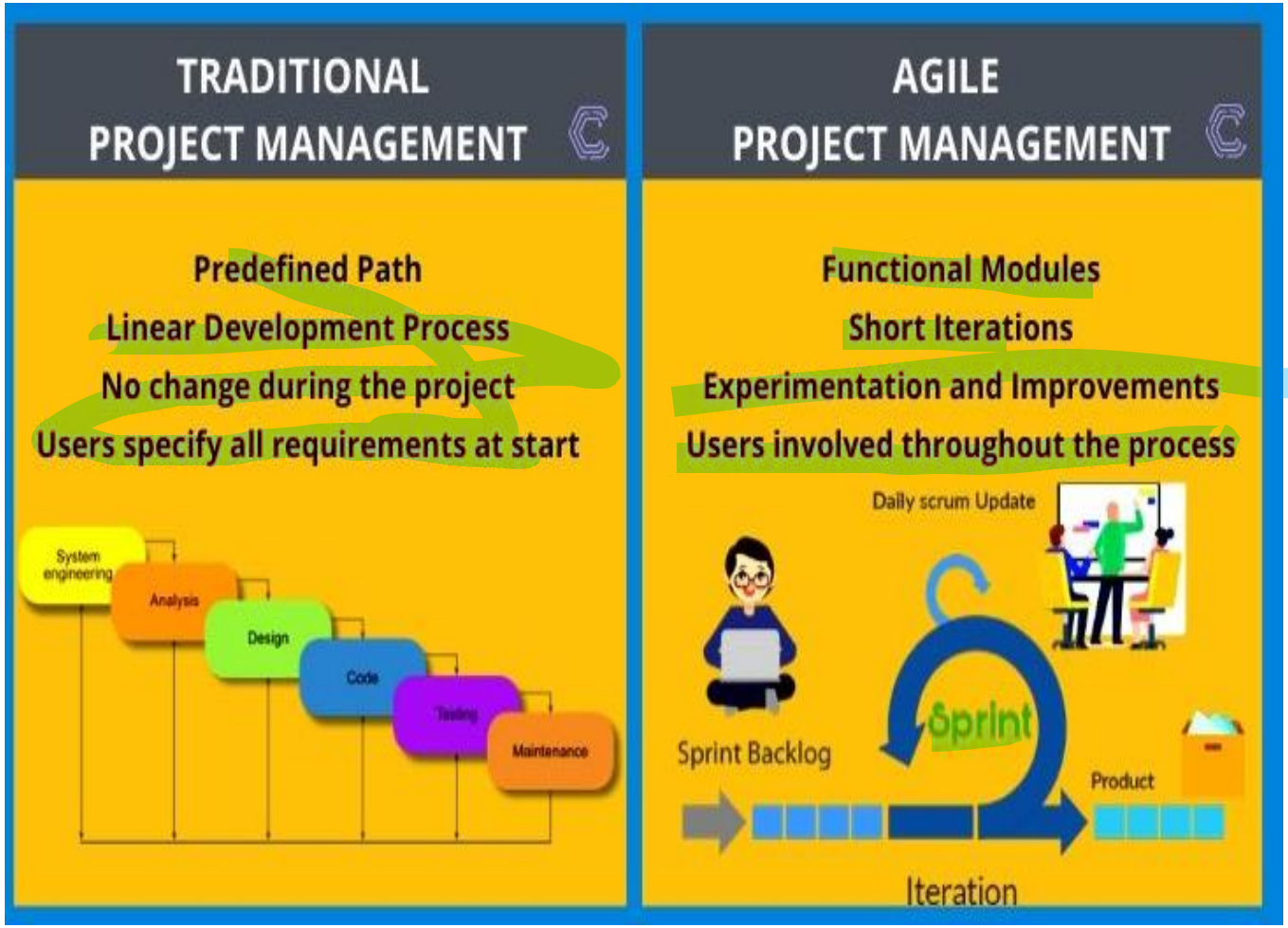


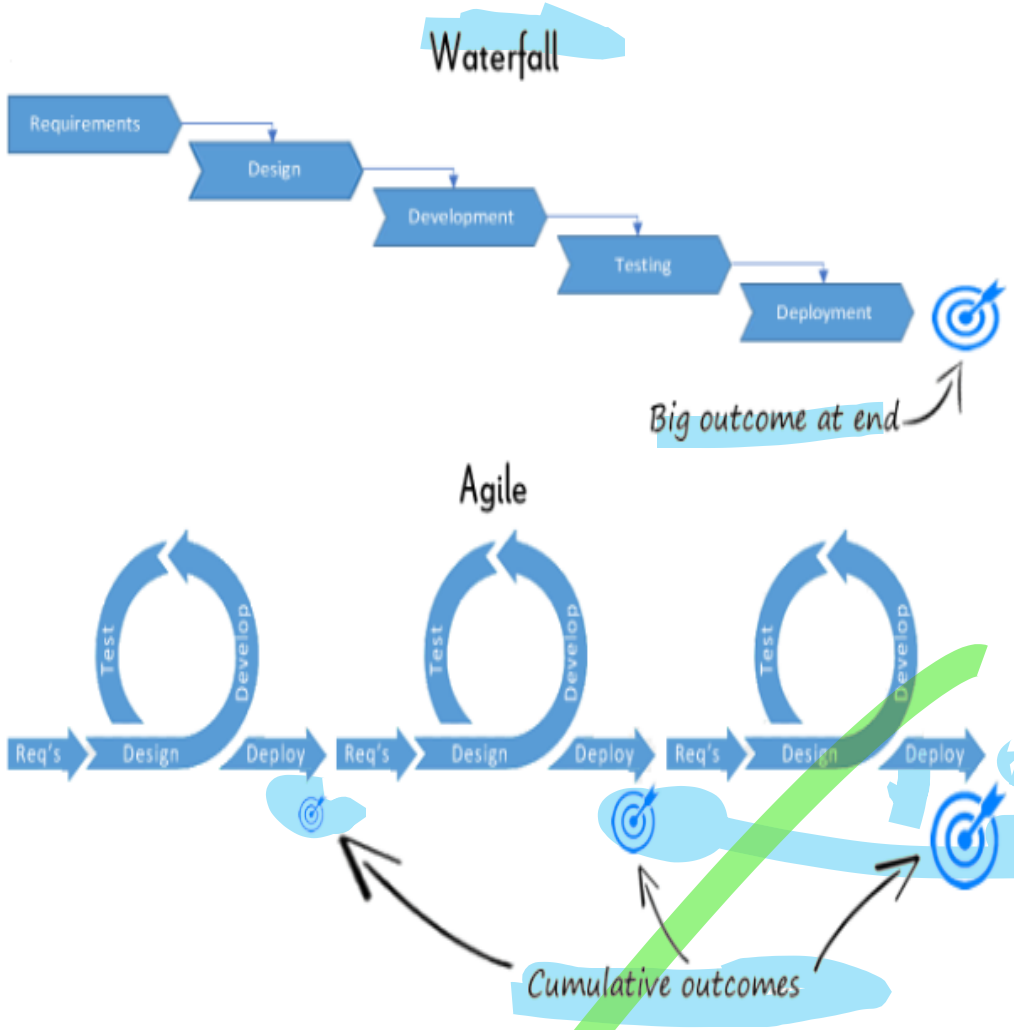
Requirements engineering in agile methodologies

- A major difference between traditional and agile methodologies is in the gathering of requirements
- Requirements engineering approaches for agile methodologies tend to be much more informal
- Some proponents of agile methodologies use requirements engineering practices as a selling point for agile

Requirements engineering: Agile versus Traditional

	Traditional	Agile
When requirements gathered?	Usually at the front end of the process	Always ongoing
Who gathers the requirements?	One or a few requirements engineers	All developers
Customer interaction	Limited to requirements engineers	All developers
Feedback to customers on requirements implementation	Sporadic, mostly at the end during acceptance testing	Ongoing
Customer involvement	Low	High
Vulnerability to changing requirements	High	Low


TRADITIONAL
PROJECT MANAGEMENT
Predefined Path
Linear Development Process
No change during the project
Users specify all requirements at start
System engineering
Analysis
Design
Code
Testing
Maintenance
AGILE
PROJECT MANAGEMENT
Functional Modules
Short Iterations
Experimentation and Improvements
Users involved throughout the process
Daily scrum Update
Sprint
Sprint Backlog
Product
Iteration


Waterfall
Requirements
Design
Development
Testing
Deployment
Big outcome at end
Agile
Test
Develop
Req's
Design
Deploy
Cumulative outcomes

Some agile Framework

- Scrum
- eXtreme Programming (XP)
- Kanban
- Lean Development
- Crystal
- Dynamic Systems Development Method (DSDM)
- Feature Driven Development (FDD)

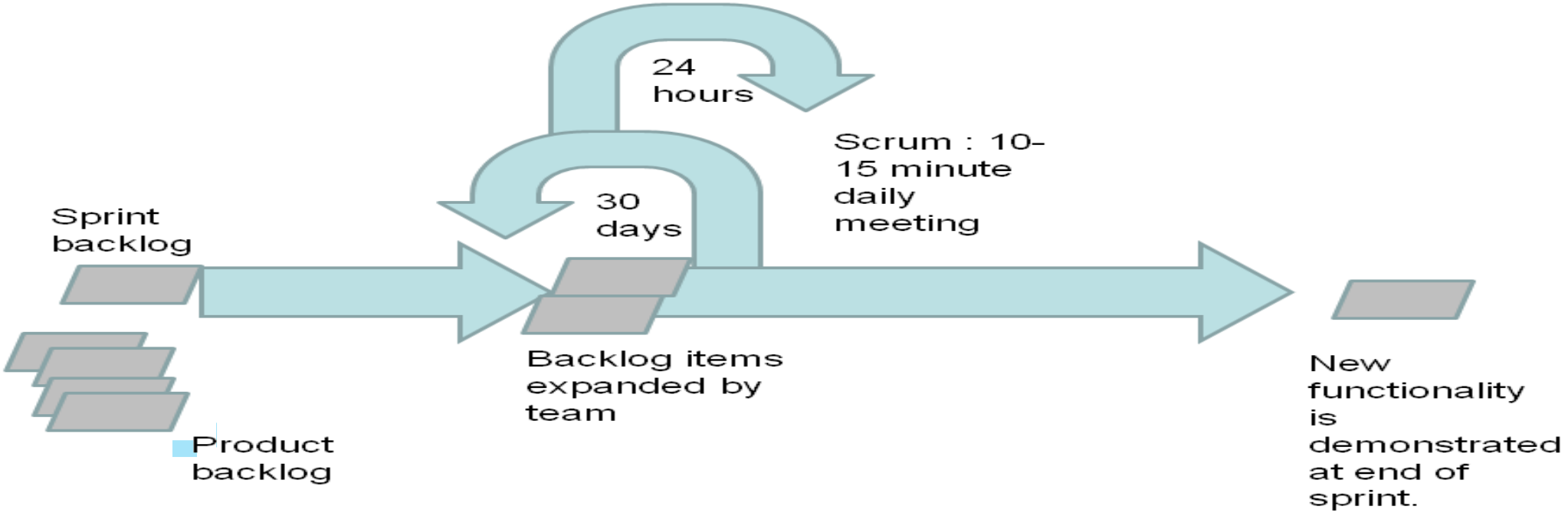
Scrum

- Relies on self-directed teams and dispenses with much advanced planning, task definition and management reporting
- A ScrumMaster, whose primary job is to remove impediments to the ability of the team to deliver the sprint goal
- The ScrumMaster is not the leader of the team (as they are self-organizing)
- A brief planning session in which the backlog items for the sprint will be defined
- A brief heartbeat retrospective, at which all team members reflect about the past sprint

Scrum

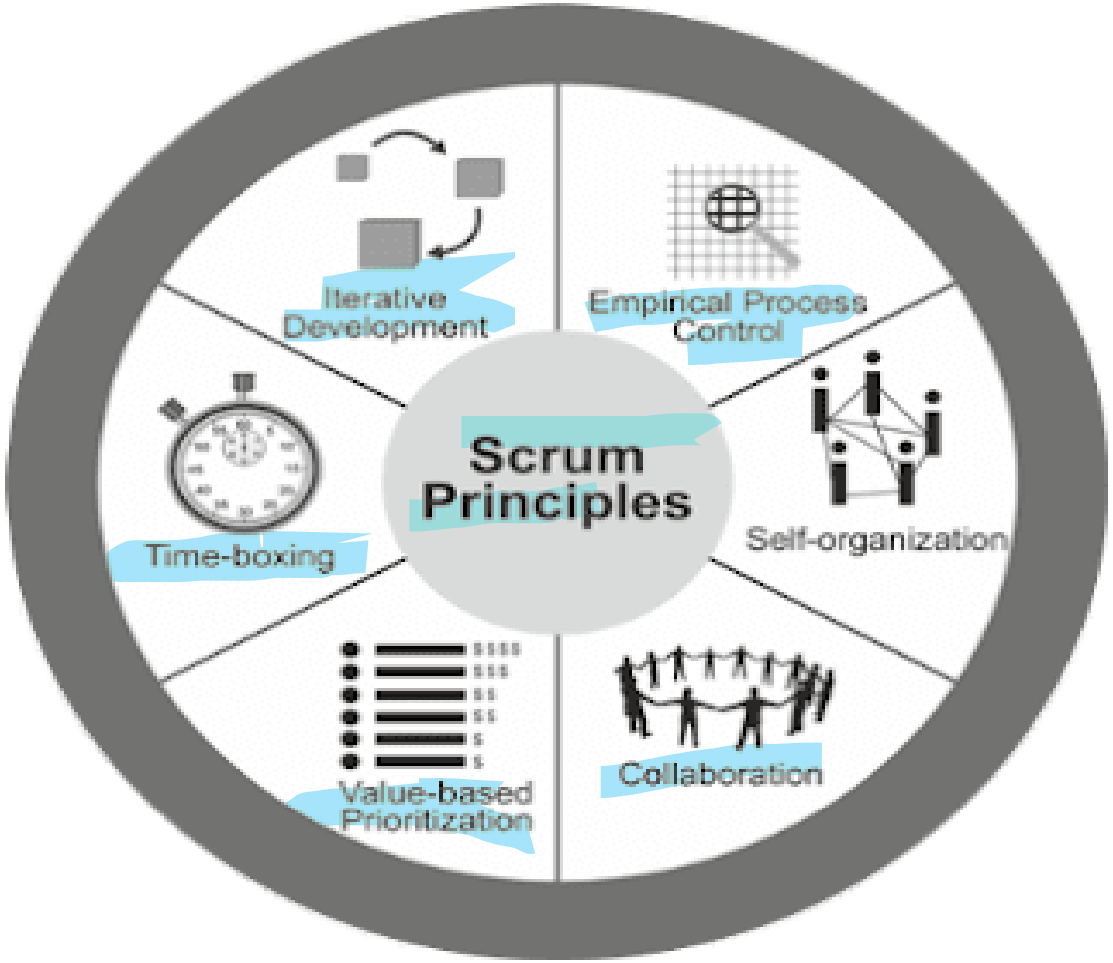
- A living backlog of prioritized work (requirements) is created
- Completion of a largely fixed set of backlog items (requirements) in a series of short (30-day) iterations or sprints
- A brief (15 minute) daily meeting (scrum), where progress is explained, upcoming work is described, and impediments are discussed
- A brief planning session in which the backlog items for the sprint will be defined

Scrum

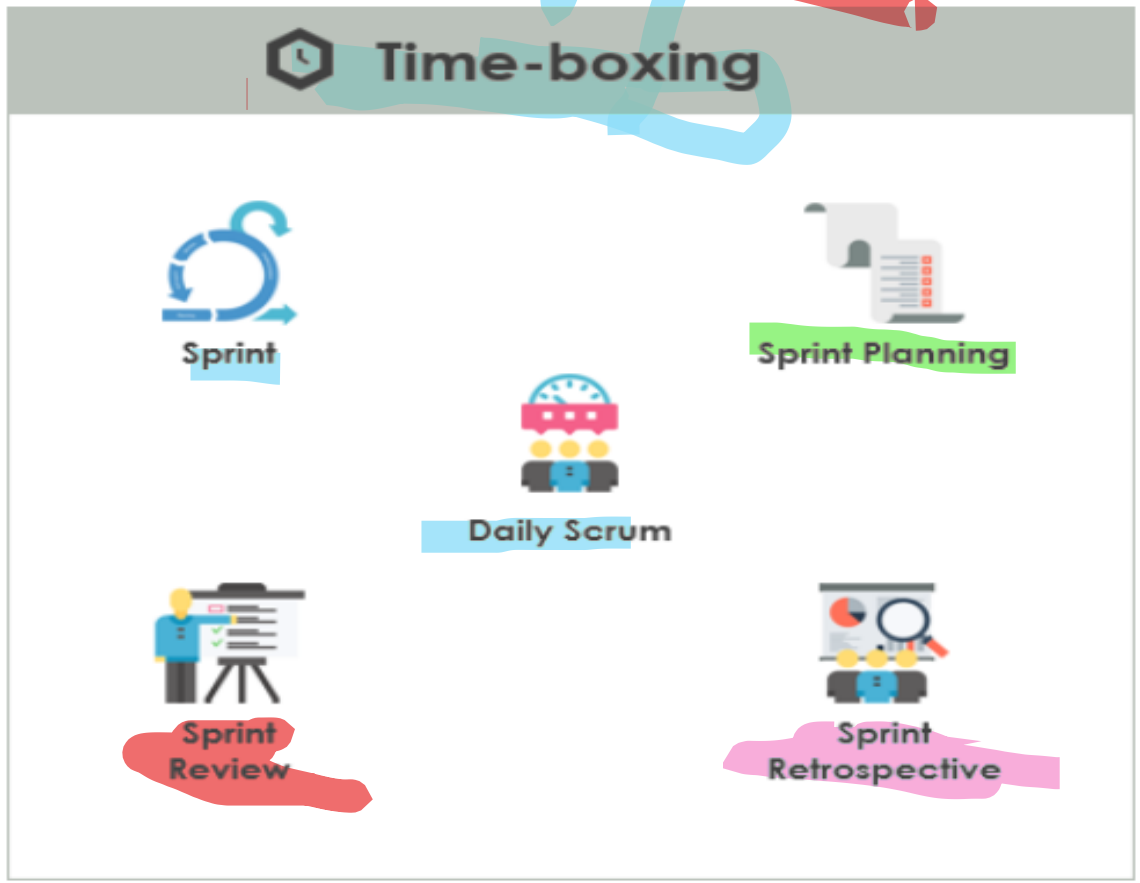

24
hours
Scrum : 10–
15 minute
daily
meeting
30
days
Sprint
backlog
Backlog items
expanded by
team
New
functionality
is
demonstrated
at end of
sprint.
Product
backlog

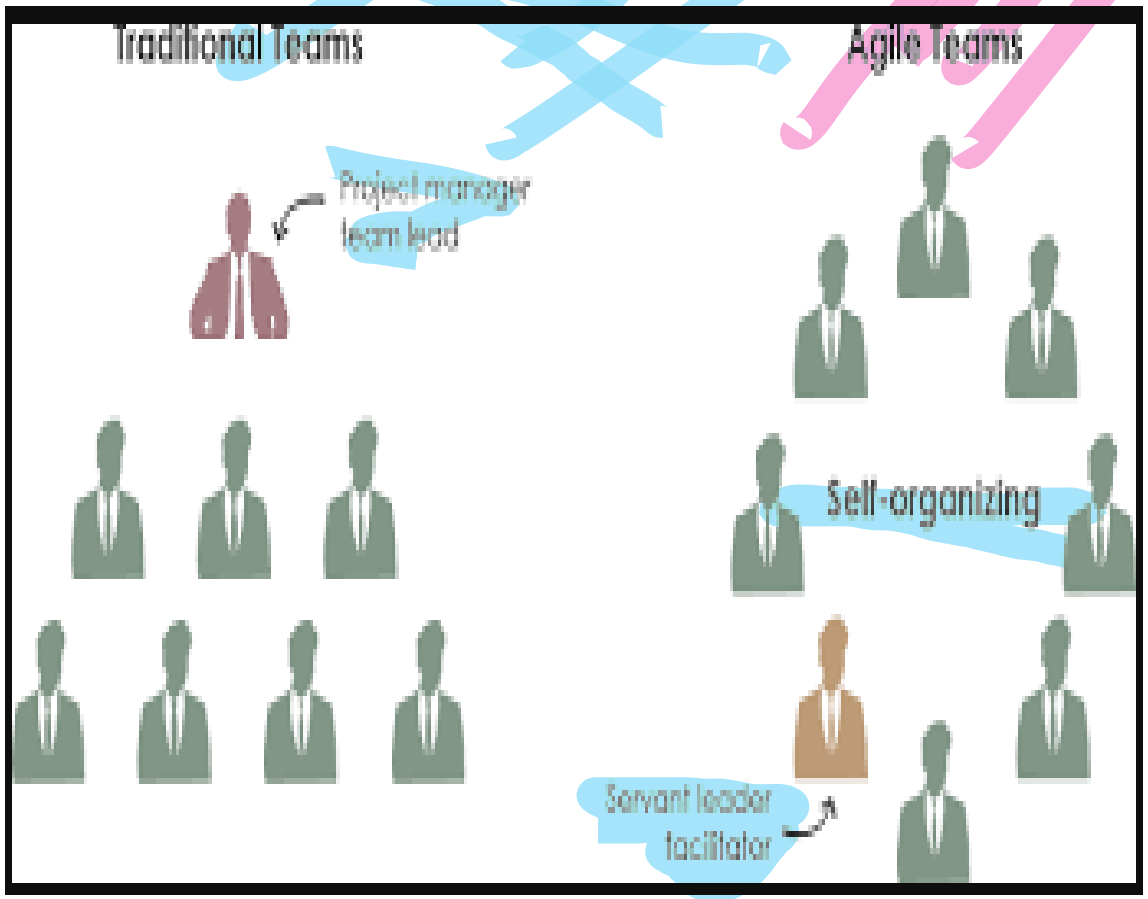
Scrum Principles

- Empirical Process Control
- Self-organization
- Collaboration
- Value-based Prioritization
- Time Boxing
- Iterative development

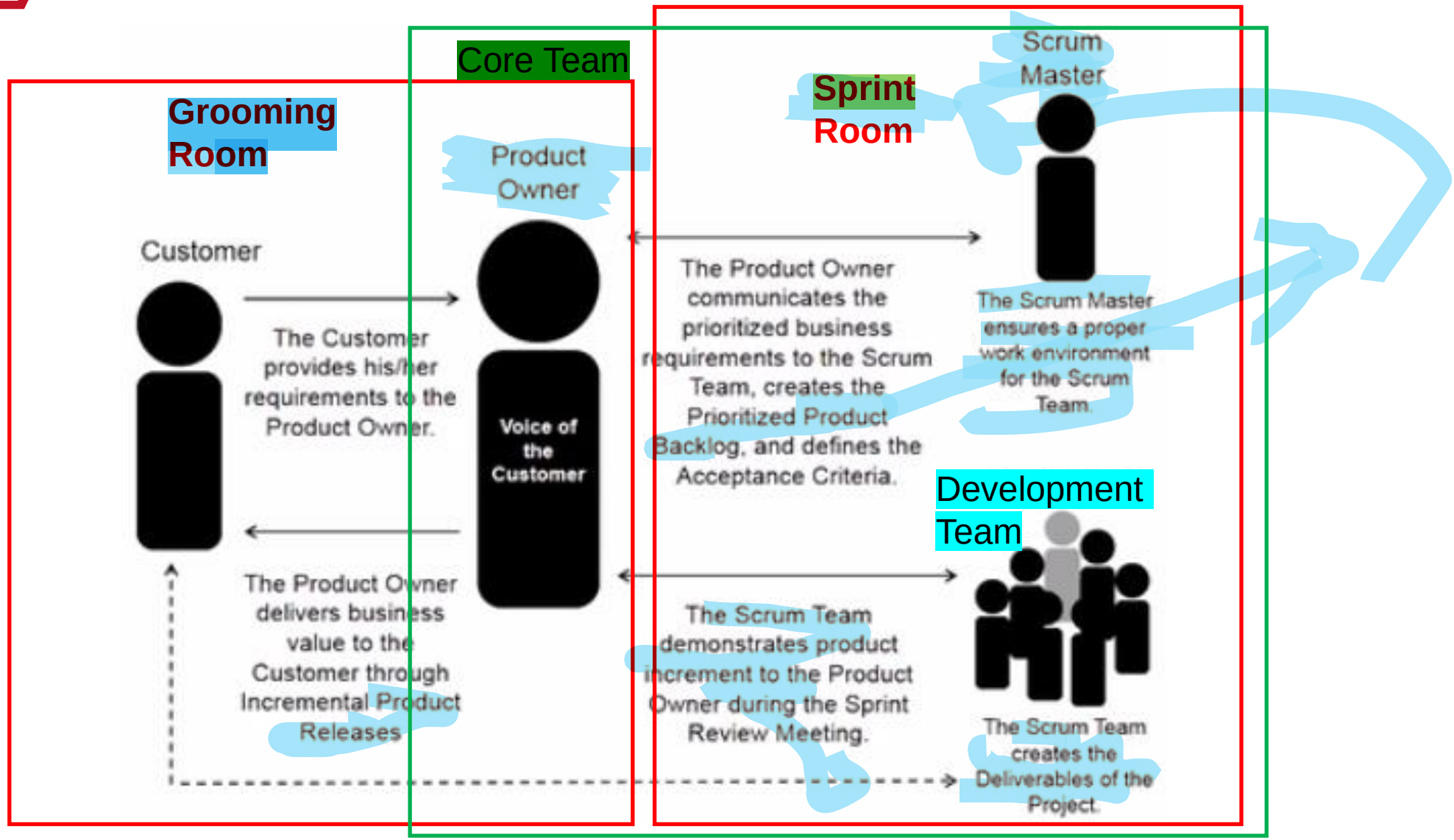


Scrum Principles

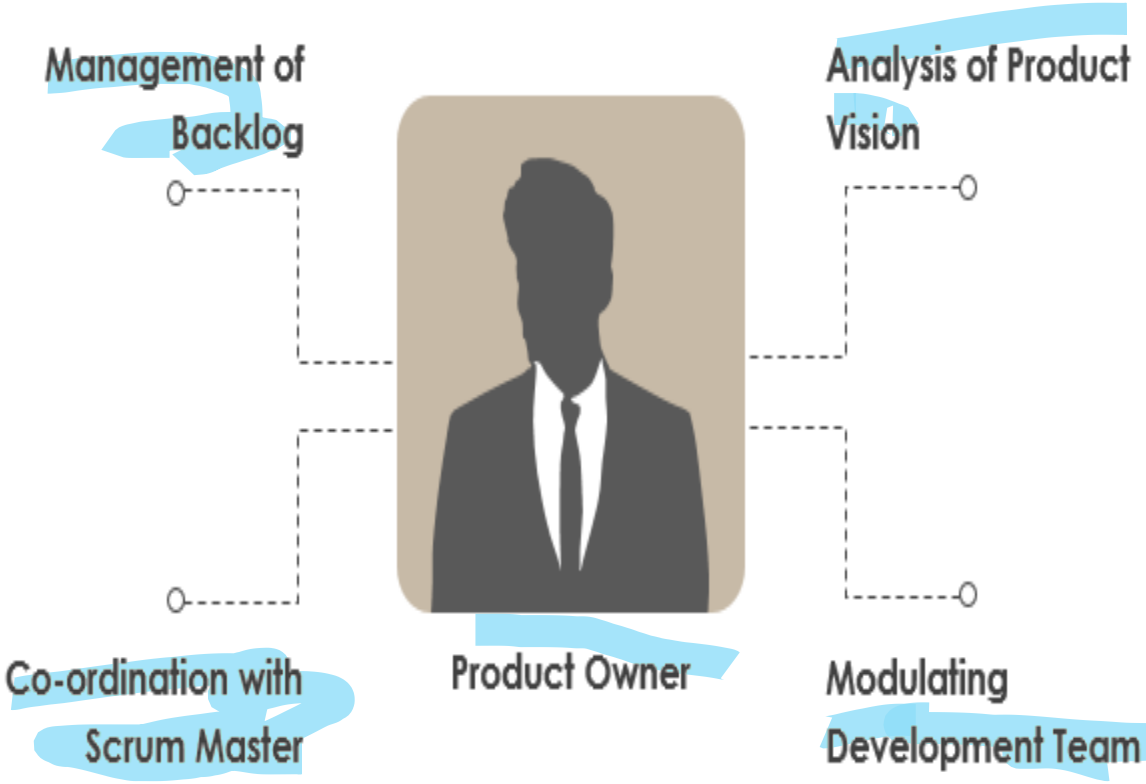




Scrum Room


Grooming Room
Core Team
Sprint Room
Product Owner
Scrum Master
Customer
The Customer provides his/her requirements to the Product Owner.
Voice of the Customer
The Product Owner communicates the prioritized business requirements to the Scrum Team, creates the Prioritized Product Backlog, and defines the Acceptance Criteria.
The Scrum Master ensures a proper work environment for the Scrum Team.
Development Team
The Product Owner delivers business value to the Customer through Incremental Product Releases
The Scrum Team demonstrates product increment to the Product Owner during the Sprint Review Meeting.
The Scrum Team creates the Deliverables of the Project.

Product Owner vs Scrum Master



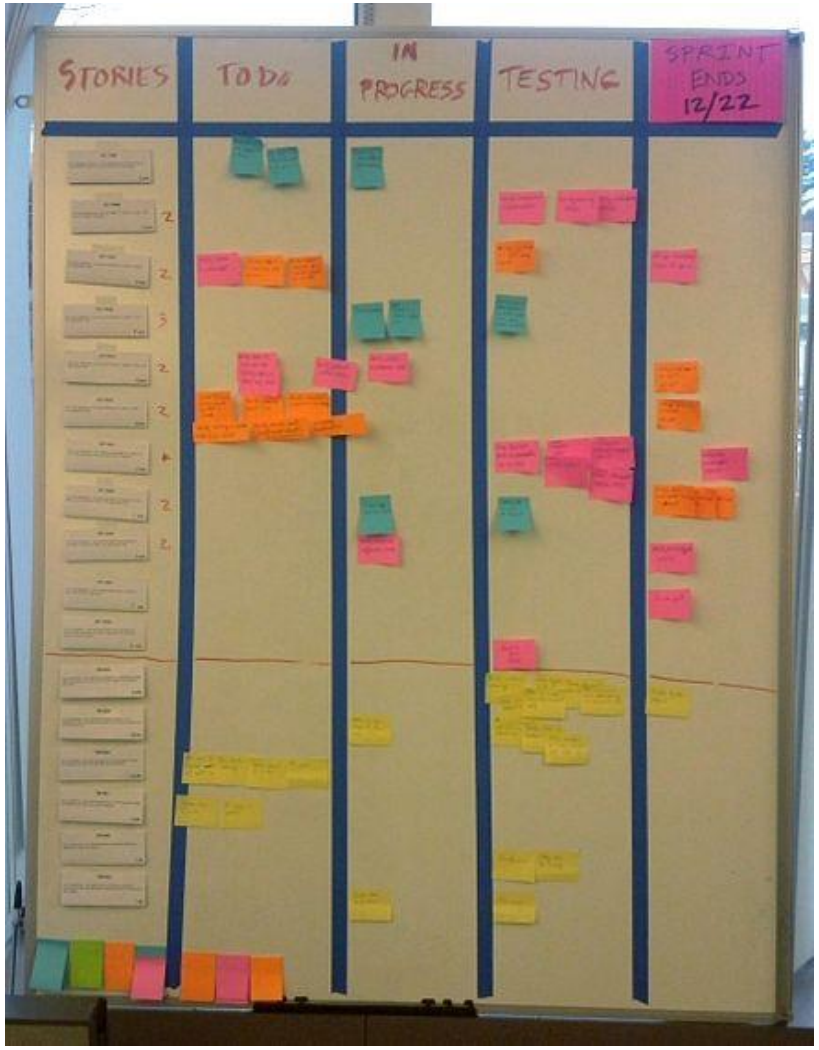
Scrum Master

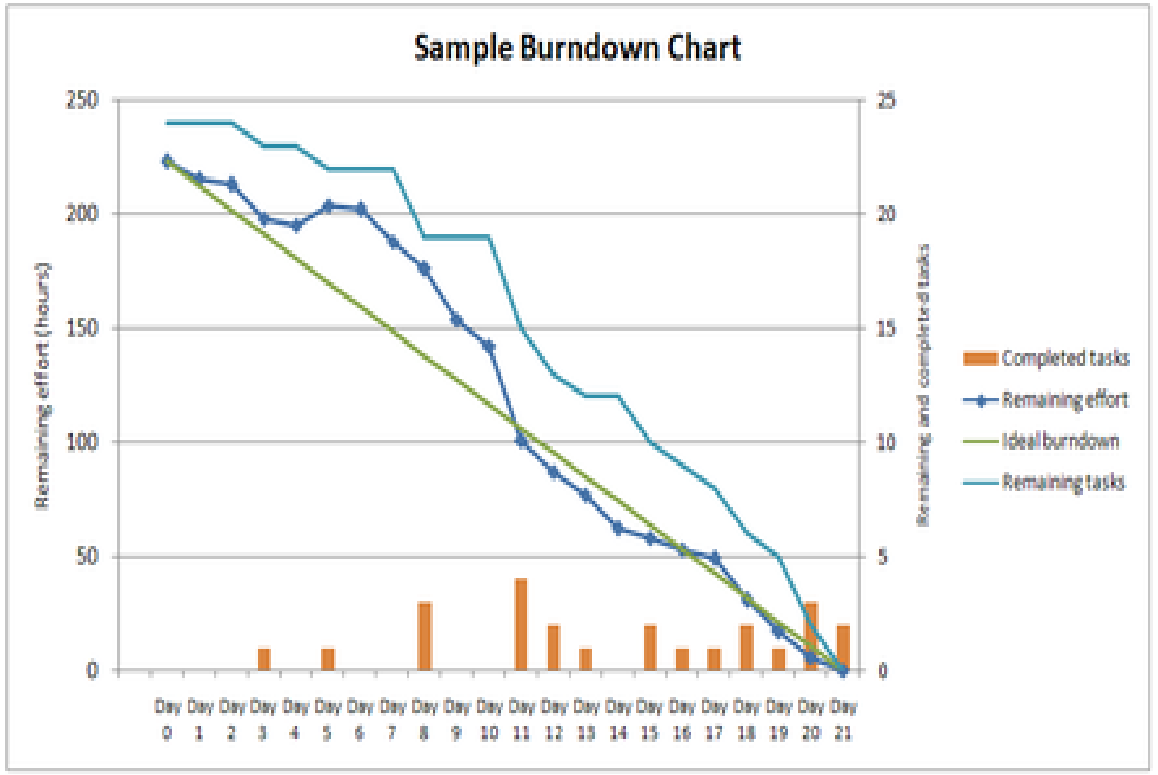
- Facilitator who encourages self-organization from development team
- Enables close cooperation across roles and functions
- Addresses resource issues
- Protect the team
- Remove impediments

RE in Scrum

- Requirements stack is living backlog
- Requirements are frozen each iteration for development stability
- One person is final authority for requirement prioritization (product owner)
- sprint-long iterations (1, 2 or 4 weeks)

RE in Scrum


STORIES
TO DO
IN PROGRESS
TESTING
SPRINT ENDS 12/22


Sample Burndown Chart
Remaining effort (hours)
Remaining and completed tasks
250
200
150
100
50
0
25
20
15
10
5
0
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Day 8
Day 9
Day 10
Day 11
Day 12
Day 13
Day 14
Day 15
Day 16
Day 17
Day 18
Day 19
Day 20
Day 21
Completed tasks
Remaining effort
Ideal burndown
Remaining tasks

Sprint Backlog

Three backlog (requirements) types

-Product

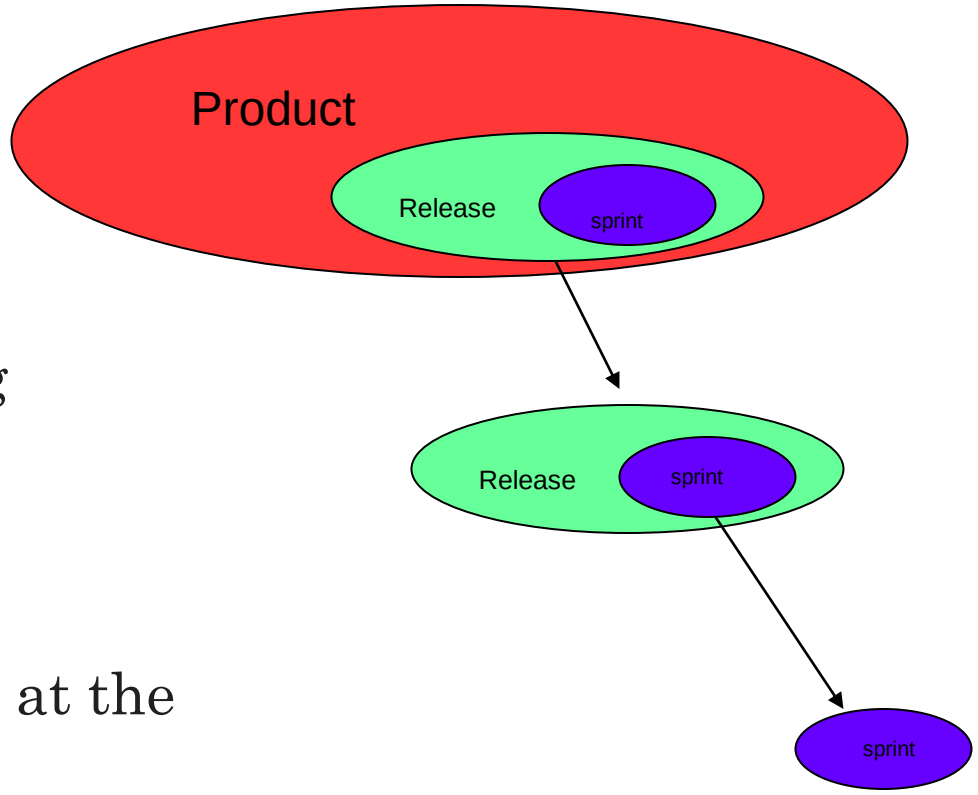
- Acts as a repository for requirements targeted for release at some point
- Typically high level requirements

-Release

- A prioritized set of items from product backlog
- Contains more details and low level estimates

-Sprint

- Set of release requirements that the team will complete (fully coded, tested and documented) at the end of the sprint
- Set can be accomplished typically in 8 - 16 hours



Extreme Programming (XP)

- Promotes a set of 12 core practices that help developers to respond to and embrace inevitable change

- The practices can be grouped according to four practice areas

1. Planning
2. Coding
3. Designing
4. Testing

Extreme Programming (XP)

1. Planning Practices

- ✓Writing user stories
- ✓Release planning
- ✓Making frequent small releases
- ✓Measuring project velocity
- ✓Dividing the project into iterations
- ✓Iteration planning
- ✓Moving people around
- ✓Stand-up meetings
- ✓Adapting XP when needed

2. Coding Practices

- ✓ Having the customer always available
- ✓ Writing code to agreed standards
- ✓ Coding the unit test cases first
- ✓ Pair-programming all production code
- ✓ Integrating code units one at a time
- ✓ Integrating often
- ✓ Using collective code ownership
- ✓ Leaving optimization until the end
- ✓ Allowing no overtime

Extreme Programming (XP)

3. Designing Practices

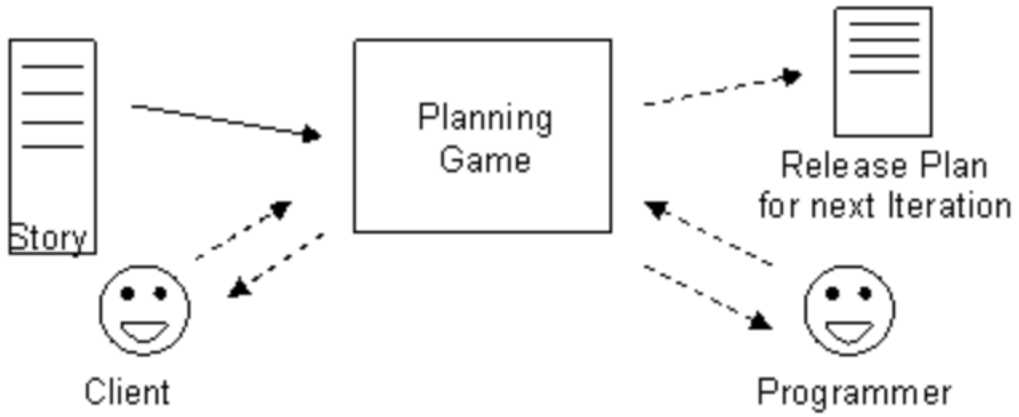
- ✓ Seeking a design that is simple
- ✓ Choosing a system metaphor (unified point of view)
- ✓ Employing Class, Responsibilities, and Collaboration (CRC) cards for design sessions
- ✓ Creating spike solutions (simple prototypes) to reduce risk
- ✓ Avoiding speculative generality
- ✓ Refactoring wherever possible

4. Testing Practices

- ✓ Having unit tests for all code
- ✓ Only releasing code after the unit tests have been passed
- ✓ Creating tests when bugs are found (to prevent them from “coming back”)
- ✓ Running acceptance tests and publishing the results

RE in XP

- ✓ Requirements is a dialog, not a document
- ✓ Stack of user stories represent requirements
- ✓ User stories are managed and implemented as code via the “planning game ”
  - ➢ Release planning
  - ➢ Iteration planning



Some Others Agile Framework

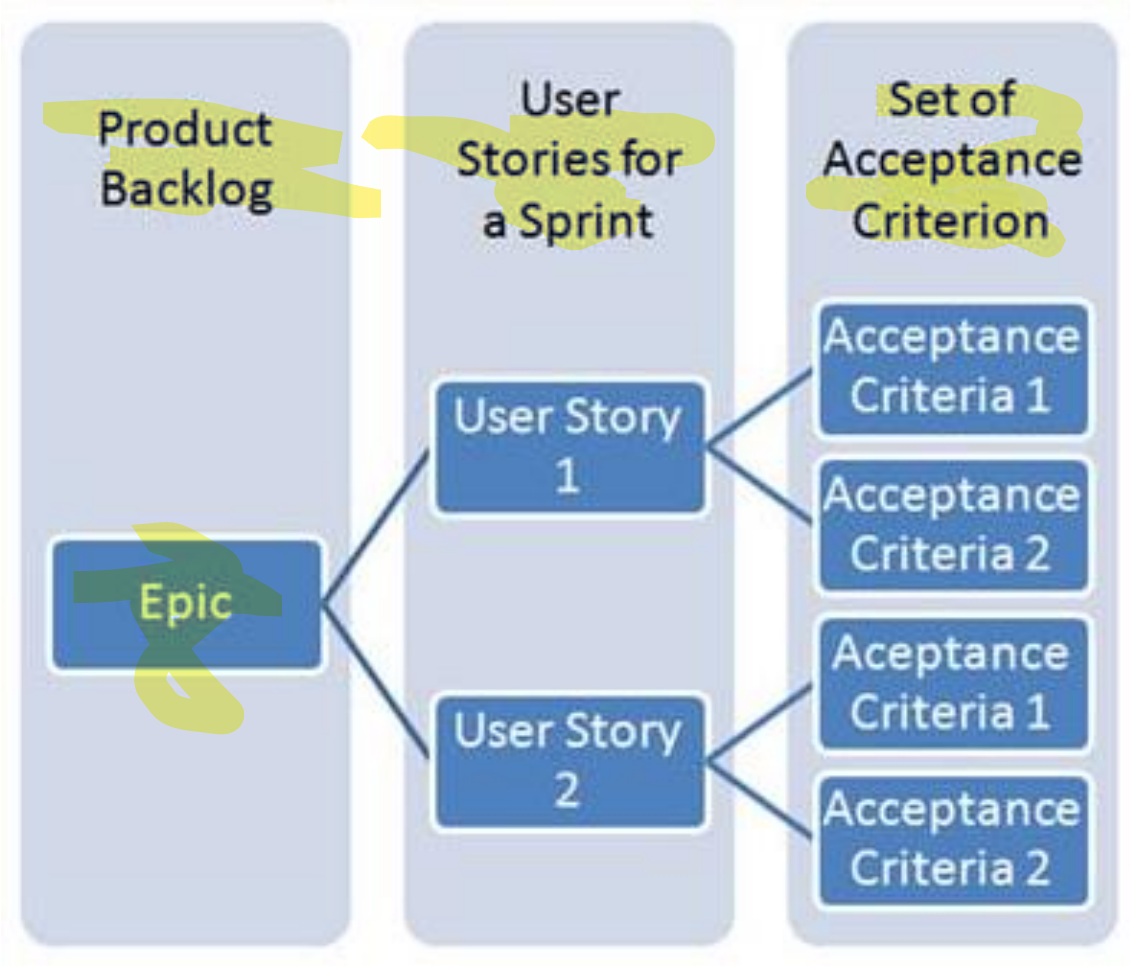
- Crystal -- empowers the development team to define the development process and refine it in subsequent iterations until it is stable

- Dynamic Systems Development Method (DSDM) -- conceived as a methodology for rapid application development. Relies on a set of principles that include empowered teams, frequent deliverables, incremental development and integrated testing

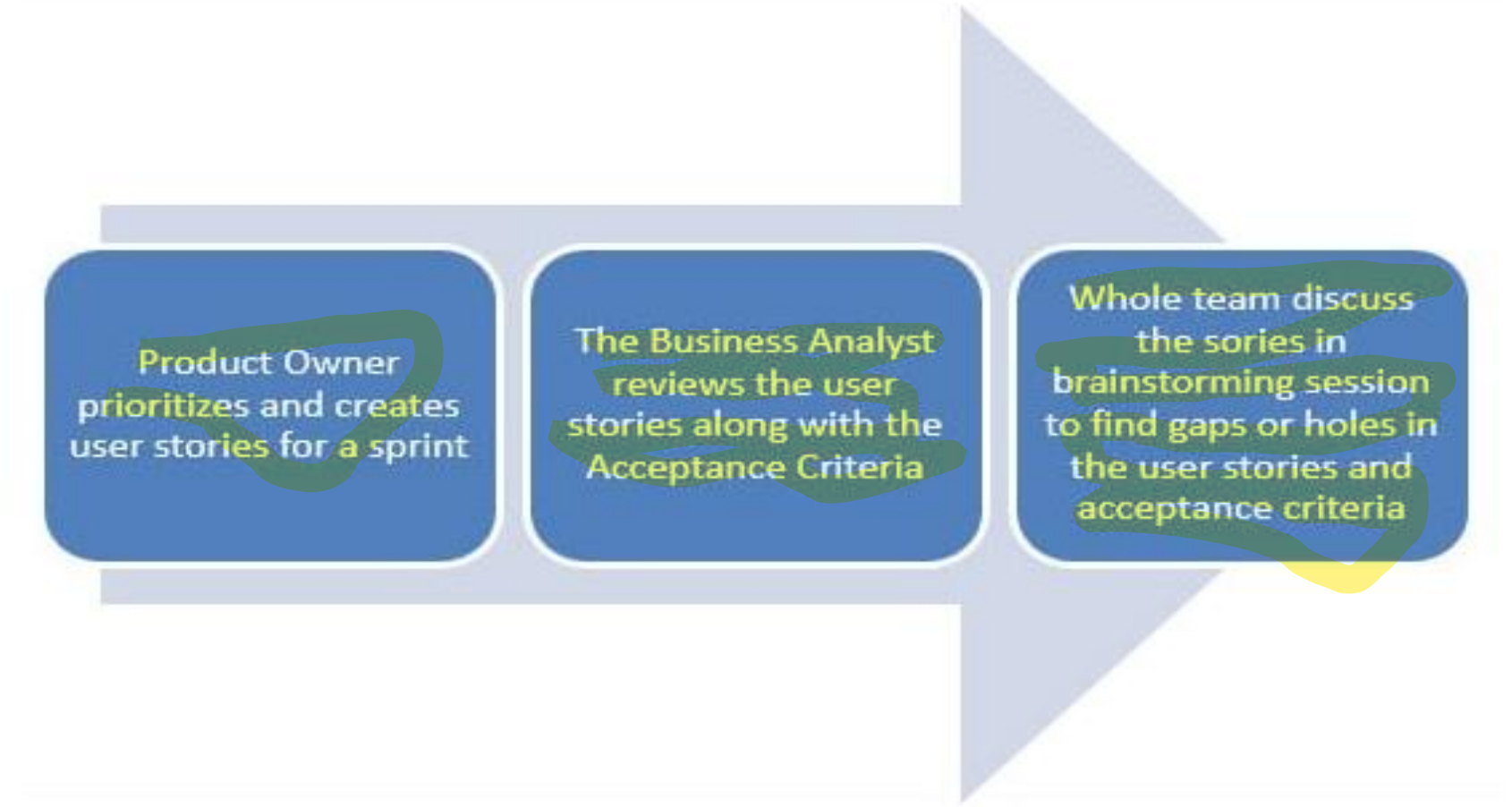
- Feature-Driven Development -- a model-driven, short-iteration methodology built around the feature, a unit of work that has meaning for the client and developer and is small enough to be completed quickly

Epic, User Story and Acceptance Criteria

Epic, User Story and Acceptance Criteria



Epic, User Story and Acceptance Criteria



Epic

- An agile epic is a large body of work that can be broken down into specific tasks (called user stories) based on the needs/requests of customers or end-users

- An epic will typically require development work covering several sprints. Hence, a user story can be completed within the timeframe of an agile sprint

- A product owner is responsible for writing Agile epics. They will liaise with key stakeholders, such as clients and investors, to ensure it satisfies the required needs

User Stories

- User stories are the most basic unit of requirements in most agile methodologies
- Each user story represents a feature desired by the customer
- User stories are written by the customer on index cards (though they can be automated via wikis or other tools)
- Formal requirements and use cases are derived from the user stories by the software engineering team as needed
- User stories are used for effort and cost estimation

User Stories

- Initial user stories are usually gathered in small offsite meetings
- Stories can be generated either through goal-oriented (e.g. “let’s discuss how a customer makes a purchase”) approaches or through interactive (stream-of-consciousness) approaches
- Developing user stories is an “iterative and interactive” process
- Testability of each story is considered by the development team
- The development team also manages the size of stories for uniformity (e.g. too large – split, too small – combine)

User Story Template

- A user story often follows the following equation:

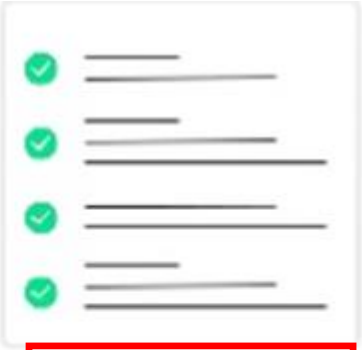
As a <type of user>, **I want** <some feature> **so that** <reason>

Example: As an online shopper, I want to add an item to my cart, so that I can purchase them

Who are we building it for? Who is the user ?	As a <type of user>
What are we building? What is the intention?	I want <some goal or objective>
Why are we building it? What is the value for the customer?	So that <benefit/value>

User Story checklist

- [x] Keep them short
- [x] Keep them simple
- [x] Write from the perspective of the user
- [x] Make the value/benefit of the story clear - what is the reason for the story?
- [x] Describe **one** piece of functionality. If you have to write **and** break it into 2 stories
- [x] Write stories as a team
- [x] Use acceptance criteria to show a MVP



Business Justification

User Story Components

- Title – a short handle for the story. Present tense verb in active voice is desirable
- Acceptance test – a unique identifier that will be the name of a method to test the story
- Priority – based on the prioritization scheme adopted
- Story points – estimated time to implement
- Description – one to three sentences describing the story

Criteria for User Story

- Stories should be understandable to the customers
- Each story should add value
- Developers do not write user stories
- Stories need to be small enough that several can be completed per iteration
- Stories should be independent (as much as possible)
- Stories must be testable – like any requirement, if it cannot be tested, it's not a requirement!

Acceptance Criteria

- User Story: As an online banking customer, I want a strong password, So that my credit card information is secure

- Acceptance Criteria:

  - The password must be at least 8 characters
  - The password must contain at least 1 character from each of the following groups: lower case alphabet, upper case alphabet, numeric, special characters (!, @, #, %, &, *)

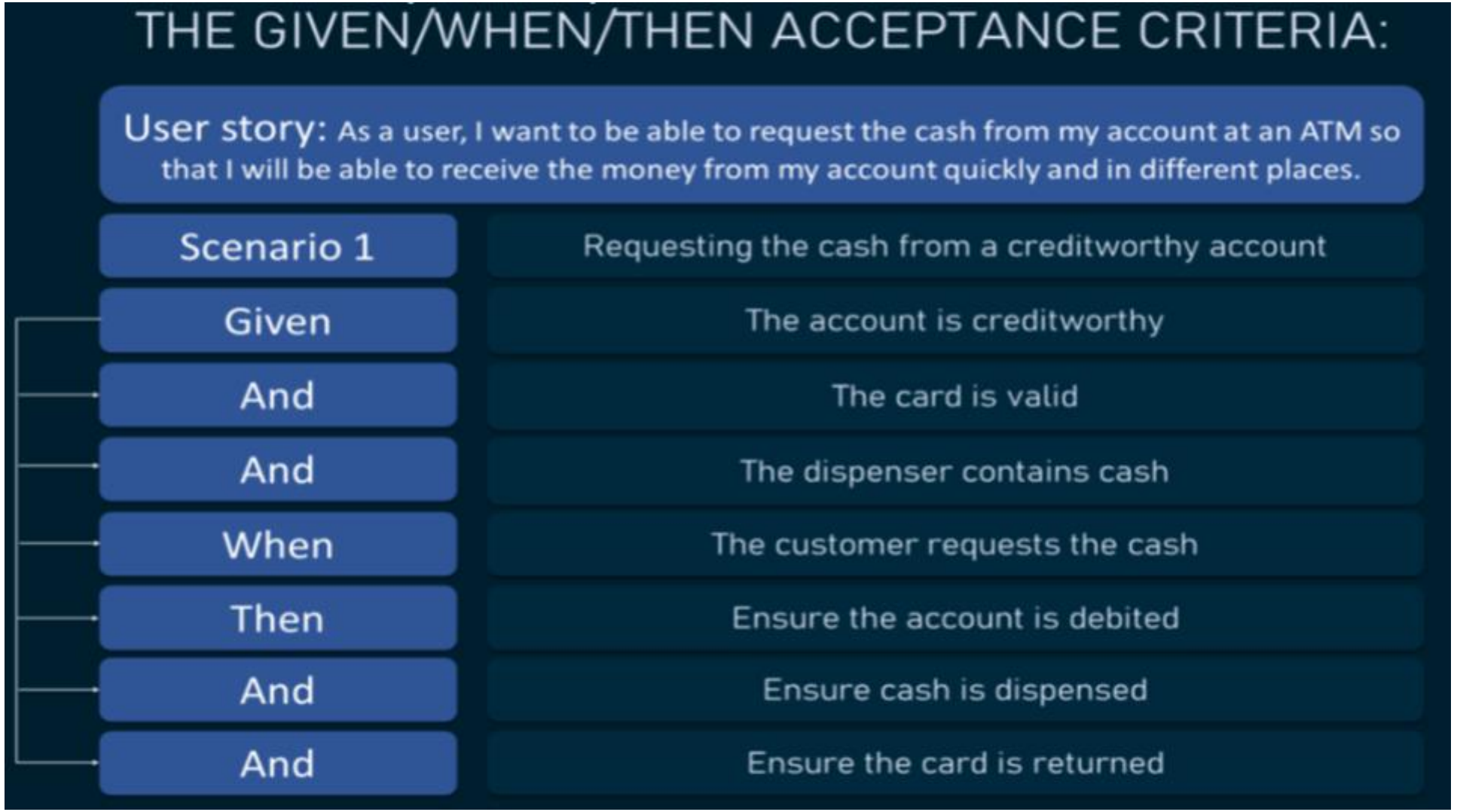
Acceptance Criteria

User story: *As a user, I want to be able to register online, so that I can start shopping online.*

Acceptance criteria:

- [ ] User can only submit a form by filling in all required fields
- [ ] The email user provided must not be a free email
- [ ] Submission from same IP can only be made three times within 30 minutes
- [ ] User can only submit a form by filling in all required fields
- [ ] User will receive a notification email after successfully registration

Acceptance Criteria Template



Acceptance Criteria

acceptance criteria should include

- Negative scenarios of the functionality
- Functional and non-functional use cases
- Performance concerns and guidelines
- What system/feature intends to do
- End-to-user flow
- The impact of a user story to other features
- UX concerns

Acceptance Criteria

acceptance criteria should NOT include

- [x] Code review was done
- [x] Non-blocker or major issues
- [x] Performance testing performed
- [x] Acceptance and functional testing done

why?

Your acceptance criteria should not include any of the above, because your team should already have a clear understanding of what your Definition of Done (DoD) means. This could mean:

- *unit/integrated tested*
- *ready for acceptance test*
- *deployed on demo server*
- *releasable*

Agile Requirements Best Practices (1)

- Stakeholders actively participate
- Implement requirements, do not only document them
- Create platform independent requirements to a point
- Smaller is better
- Question traceability
- Explain the techniques
- Adopt stakeholder terminology

Agile Requirements Best Practices (2)

- Obtain management support
- Turn stakeholders into developers
- Treat requirements like a prioritized stack
- Frequent, personal interaction
- Frequent delivery of software
- Express requirements as features
- Keep it fun

Challenges in Agile RE

- Dealing with non-functional requirements - Because they are not always apparent when dealing with requirements functionality only
- Customer interaction is strong but mostly through prototyping - Using various interviewing techniques would be desirable
- Validation is strong through testing, but verification is weak - Using formal methods could strengthened agile RE
- Requirements management is built into the process (e.g. Scrum and XP) but it is mostly focused on the code level - Document changes to the requirements comprehensibly

THANK YOU

Dr. Taher Labidi
Dr. Mariem Haoues

DEPARTMENT: Software Engineering