Agile Methodologies

Introduction

Agenda

- Agile
  - Manifesto
  - Principles
  - Design
- Scrum

What is Agile?

- A set of methods for software development
  - Iterative
  - Incremental
  - Assume changeability of requirements
- First appearance: 1957, IBM's Service Bureau Computation
- Consolidation and awareness: 2001, Agile Manifesto

Agile Manifesto

- Agile Values
  1. Individuals and interactions over processes and tools
  2. Working software over comprehensive documentation
  3. Customer collaboration over contract negotiation
  4. Responding to change over following a plan

Agile Manifesto

- Agile Values
  1. Individuals and interactions over processes and tools
     - People are the most important ingredient for success, even though a bad process can make great people ineffective
  2. Working software over comprehensive documentation
  3. Customer collaboration over contract negotiation
  4. Responding to change over following a plan

Agile Manifesto

- Agile Values
  1. Individuals and interactions over processes and tools
  2. Working software over comprehensive documentation
     - Software without documentation is a disaster, but that documentation needs to be human-readable, short, salient and high-level. Training people to lower level details is performed via close collaboration with trainees
  3. Customer collaboration over contract negotiation
  4. Responding to change over following a plan

Agile Manifesto

- Agile Values
  1. Individuals and interactions over processes and tools
  2. Working software over comprehensive documentation
  3. Customer collaboration over contract negotiation
     - Successful projects involve customer feedback on a regular and frequent basis
  4. Responding to change over following a plan

Agile Manifesto

- Agile Values
  1. Individuals and interactions over processes and tools
  2. Working software over comprehensive documentation
  3. Customer collaboration over contract negotiation
  4. Responding to change over following a plan
     - Once customers see the system start to function, they are likely to alter the requirements: make detailed plans for the next week (individual tasks), rough plans for the next 3 months (general requirements), and extremely crude plans beyond that (just an idea)

Agile Principles

- 12 principles, identified by 17 software developers* who met up at Snowbird, Utah
  1. Customer satisfaction by rapid delivery of useful software
  2. Welcome changing requirements, even late in development
  3. Working software is delivered frequently (weeks rather than months)
  4. Working software is the principal measure of progress
  5. Sustainable development, able to maintain a constant pace
  6. Close, daily cooperation between business people and developers
  7. Face-to-face conversation is the best form of communication (co-location)
  8. Projects are built around motivated individuals, who should be trusted
  9. Continuous attention to technical excellence and good design
  10. Simplicity - the art of maximizing the amount of work not done - is essential
  11. Self-organizing teams
  12. Regular adaptation to changing circumstances

* Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Stephen J. Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah
  1. **Customer satisfaction by rapid delivery of useful software**
  3. **Working software is delivered frequently**
  4. **Working software is the principal measure of progress**

- **Strong connection between quality, customer satisfaction and frequency of delivery**
- **Deliver every 2-4 weeks**
- **Deliver working software, not documentation**

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah

2. **Welcome changing requirements, even late in development**

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah

5. **Sustainable development, able to maintain a constant pace**
6. **Close, daily cooperation between business people and developers**
7. **Face-to-face conversation is the best form of communication**

- **Co-located/interacting teams**
- **No overtime or final rush**

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah
  - **Focus on quality of both team and work**
  - **Distribution of tasks across team members, according to their skills without degrading to silos of responsibility: everyone is responsible for everything**

8. **Projects are built around motivated individuals, who should be trusted**
9. **Continuous attention to technical excellence and good design**
11. **Self-organizing teams**

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah

10. **<u>Simplicity - the art of maximizing the amount of work not done - is essential</u>**

Agile Principles

- 12 principles, identified by 17 software developers who met up at Snowbird, Utah

- **When the environment changes, the team shall change accordingly**

12. **<u>Regular adaptation to changing circumstances</u>**

Agile Design

- Team improves the design of the system so that it is **as good as it can be** for the system as it is **now**
  - The team does not spend much time looking ahead to future requirements and needs
  - The team does not try to build today the infrastructure to support the features that may be needed tomorrow
- Software architecture is still crucial, but it has to continually evolve
  - During development, the current architecture shall be the *best* architecture for the current software
  - Eventually, the achieved architecture shall be the *best* architecture for the software as it will be at the end of the development

Agile Design

- SOLID - “First five principles”
  - **S**ingle-Responsibility Principle
  - **O**pen/Closed Principle
  - **L**iskov Substitution Principle
  - **I**nterface Segregation Principle
  - **D**ependency-Inversion Principle

Agile Design

- SOLID - “First five principles”
  - **Single-Responsibility Principle**
    - High cohesion of a component, i.e. high relatedness of its elements and behaviors
  - Open/Closed Principle
  - Liskov Substitution Principle
  - Interface Segregation Principle
  - Dependency-Inversion Principle

Agile Design

- SOLID - “First five principles”
  - Single-Responsibility Principle
  - **Open/Closed Principle**
    - Open to changes, close to modifications (no cascades of modifications)
  - Liskov Substitution Principle
  - Interface Segregation Principle
  - Dependency-Inversion Principle

Agile Design

- SOLID - “First five principles”
  - Single-Responsibility Principle
  - Open/Closed Principle
  - **Liskov Substitution Principle**
    - Subtypes shall be replaced by base types systematically, so that extensions do not cause avoidable changes
  - Interface Segregation Principle
  - Dependency-Inversion Principle

Agile Design

- SOLID - “First five principles”
  - Single-Responsibility Principle
  - Open/Closed Principle
  - Liskov Substitution Principle
  - **Interface Segregation Principle**
    - An interface shall offer an atomic set of behaviors, i.e. each entity using that interface shall use all of it
  - Dependency-Inversion Principle

Agile Design

- SOLID - “First five principles”
  - Single-Responsibility Principle
  - Open/Closed Principle
  - Liskov Substitution Principle
  - Interface Segregation Principle
  - **Dependency-Inversion Principle**
    - High-level modules shall not depend on low-level modules, but both should depend on abstractions, which should not depend on details

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- Immobility
- Viscosity
- Needless complexity
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- **Rigidity**
  - Tendency for software to be difficult to change, even in simple ways. A design is rigid if a single change causes a cascade of subsequent changes in dependent modules
- Fragility
- Immobility
- Viscosity
- Needless complexity
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- **Fragility**
  - Tendency of a program to break in many places when a single change is made
- Immobility
- Viscosity
- Needless complexity
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- **Immobility**
  - A design is immobile when it contains parts that could be useful in other systems, but the effort and risk involved with separating those parts from the original system are high
- Viscosity
- Needless complexity
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- Immobility
- **Viscosity**
  - When faced with a change, developers usually find more than one way to make that change. Some of the ways preserve the design, others do not (i.e., they are hacks). When the design-preserving methods are more difficult to use than the hacks, the viscosity of the design is high. It is easy to do the wrong thing but difficult to do the right thing
- Needless complexity
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- Immobility
- Viscosity
- **Needless complexity**
  - A design is needlessly complex when it contains elements that are not currently useful, e.g. because developers anticipate changes to the requirements and put facilities in the software to deal with those potential changes. This may seem like a good thing, but the design becomes littered with constructs that are never used
- Needless repetition
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- Immobility
- Viscosity
- Needless complexity
- **Needless repetition**
  - When the same code appears over and over again, in slightly different forms, the developers are missing an abstraction. When there is redundant code in the system, the job of changing the system can become arduous, e.g. when bugs are detected
- Opacity

Symptoms of Poor Agile Design

- Rigidity
- Fragility
- Immobility
- Viscosity
- Needless complexity
- Needless repetition
- **Opacity**
  - Tendency of a module to be difficult to understand. Code can be written in a clear and expressive manner, or it can be written in an opaque and convoluted manner. Code that evolves over time tends to become more and more opaque with age

Some Agile Methods

- Agile Unified Process (AUP)
- Crystal Clear
- Extreme Programming
- Feature-Driven Development
- Kanban
- Lean Software Development
- Scrum
- Test-Driven Development
- Dynamic System Development Method (DSDM)

Some Agile Methods

- Agile Unified Process (AUP)
- Crystal Clear
- Extreme Programming
- Feature-Driven Development
- Kanban
- Lean Software Development
- **Scrum**
- Test-Driven Development
- Dynamic System Development Method (DSDM)

Scrum

- Not a software development process, but a set of practices
- A framework to instantiate
- A software development process can leverage Scrum practices
- Based on an iterative, incremental and empirical approach
- Transparent

Scrum

3 Artifacts

- Product Backlog
- Sprint Backlog
- Increment

3 Roles

- Product Owner
- Scrum Master
- Development Team

4 Meetings

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Scrum

3 Artifacts

- Product Backlog
- Sprint Backlog
- Increment

3 Roles

- Product Owner
- Scrum Master
- Development Team

4 Meetings

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review
- Sprint Retrospective

Scrum Sprint

- Single iteration of fixed length
- Length < 1 month
- Development is performed during a Sprint
- During each Sprint, the same amount of work shall be done
  - The amount of work done is called velocity and represents the development pace
- A new Sprint starts immediately after the end of the last Sprint
- Each Sprint ends with an increment on the product (e.g. a new functionality)

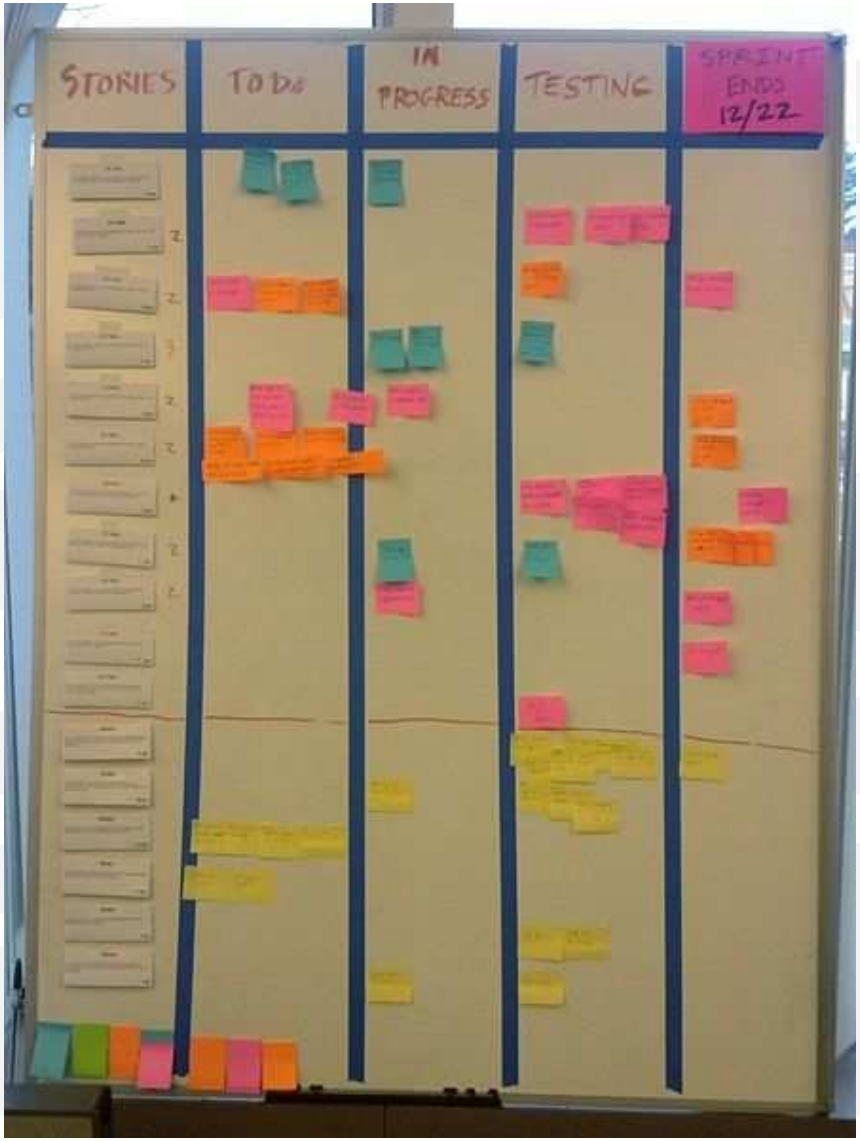
Product Backlog

- Ordered list of everything that might be needed in the product
  - Features, functions, requirements, enhancements, and fixes
- Single source of requirements for any changes to be made to the product
- Never complete
- Product Backlog items have attributes: description, order, estimate and value
- Product Backlog refinement is the act of adding details, estimates and order to items in the Product Backlog
  - Only items on top of the product backlog are detailed at fine-grained level
- Backlog items can be grouped by custom criteria

Sprint Backlog

- Selection of Product Backlog items to develop during a sprint
- Items are small and detailed enough so that they can be understood and developed within the sprint by the Development Team
- Items are detailed and modified as the Sprint proceeds
- No new items can be added from the Product Backlog during the Sprint

Scrum Board



Definition of Done

- A backlog item can be “Done” or not
- The meaning of Done must be shared among stakeholders
- No prescriptive definition
  - The team shall define one
- E.g. define a checklist of actions to perform before declaring an item as “done”

Increment

- Sum of “Done” Product Backlog Items
- Shipped at the end of the Sprint
- Not necessarily a release

Scrum Team and Roles

- Typically 5-11 members
- Three roles
  - Product Owner
  - Development Team
  - Scrum Master

Scrum Team and Roles

- Typically 5-11 members
- Three roles
  - **Product Owner**
    - Exclusive management and control of the product backlog
      - This includes assigning priorities to backlog items
    - Represents the voice of the customers
    - Responsible for the value of the work done
    - A single person, not a committee
  - Development Team
  - Scrum Master

Scrum Team and Roles

- Typically 5-11 members
- Three roles
  - Product Owner
  - **Development Team**
    - Developers who turn the backlog items into shippable software
    - Solely responsible on how to develop the system
    - Responsible for estimating the backlog items
    - Cross-functional
    - Self-organizing
    - All “developers”, no one is over the others
  - Scrum Master

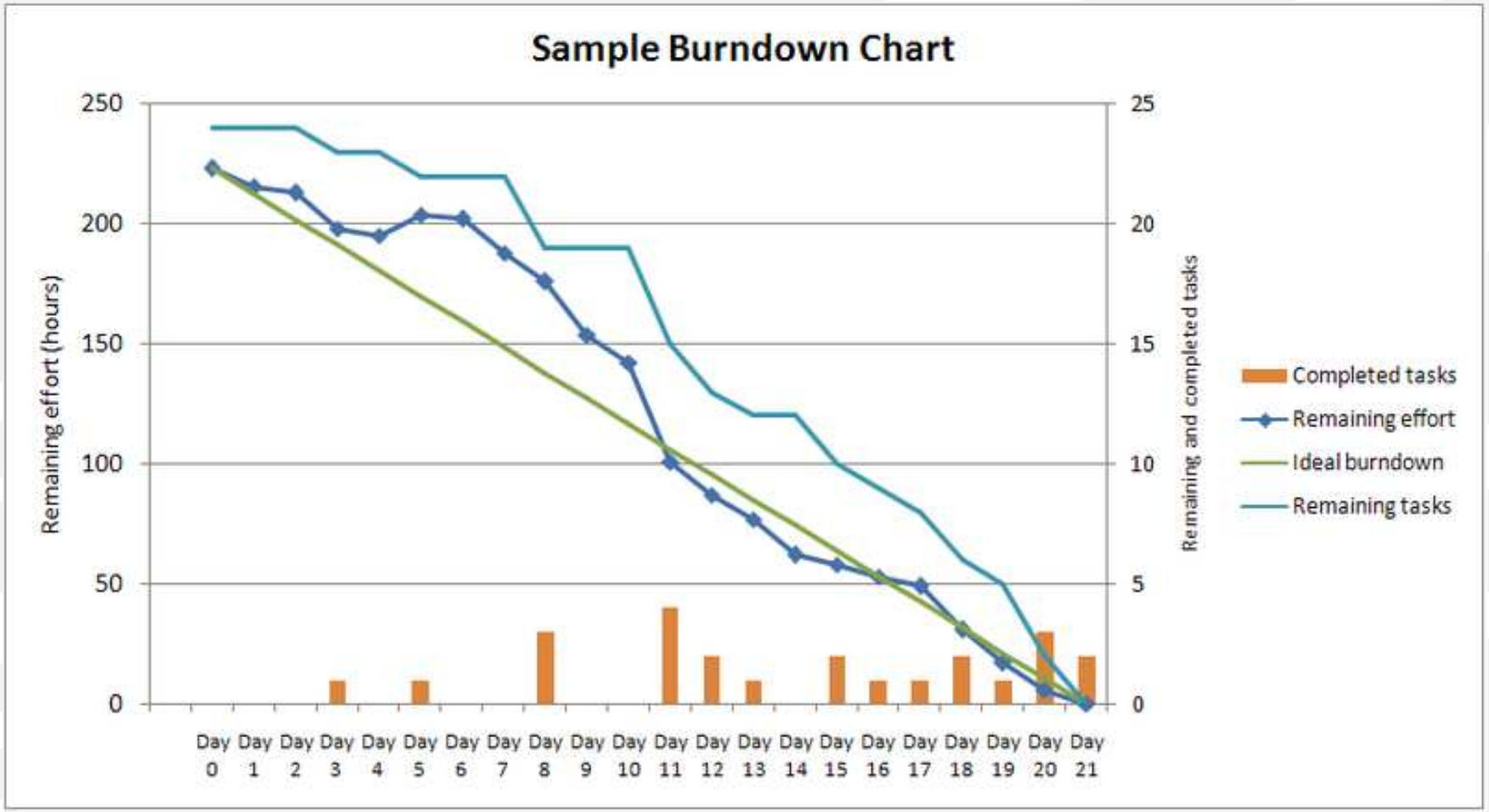
Scrum Team and Roles

- Typically 5-11 members
- Three roles
  - Product Owner
  - Development Team
  - **Scrum Master**
    - Supports the Product Owner to effectively manage the product backlog
    - Supports the Development Team by coaching on agile and removing impediments
    - Supports the organization in planning and adopting Scrum

Sprint Planning Meeting

- Length ≤ 8 hours at the beginning of a Sprint
- The entire Scrum Team participates
- Product Owner presents the top priorities of the next Sprint
- Development Team
  - Picks what can be developed within the next Sprint
  - Defines a Sprint Goal, i.e. the objective to meet by the end of the Sprint
  - Finally, defines a plan to achieve the goal

Scrum: Burndown Chart



Daily Scrum Meeting

- ~15 minutes at the beginning of the day
- Development Team and Scrum Master participate
- Synchronization of the work to be done during the day
- Each team members shares
  - Yesterday progress
  - Today plans
  - Encountered problems

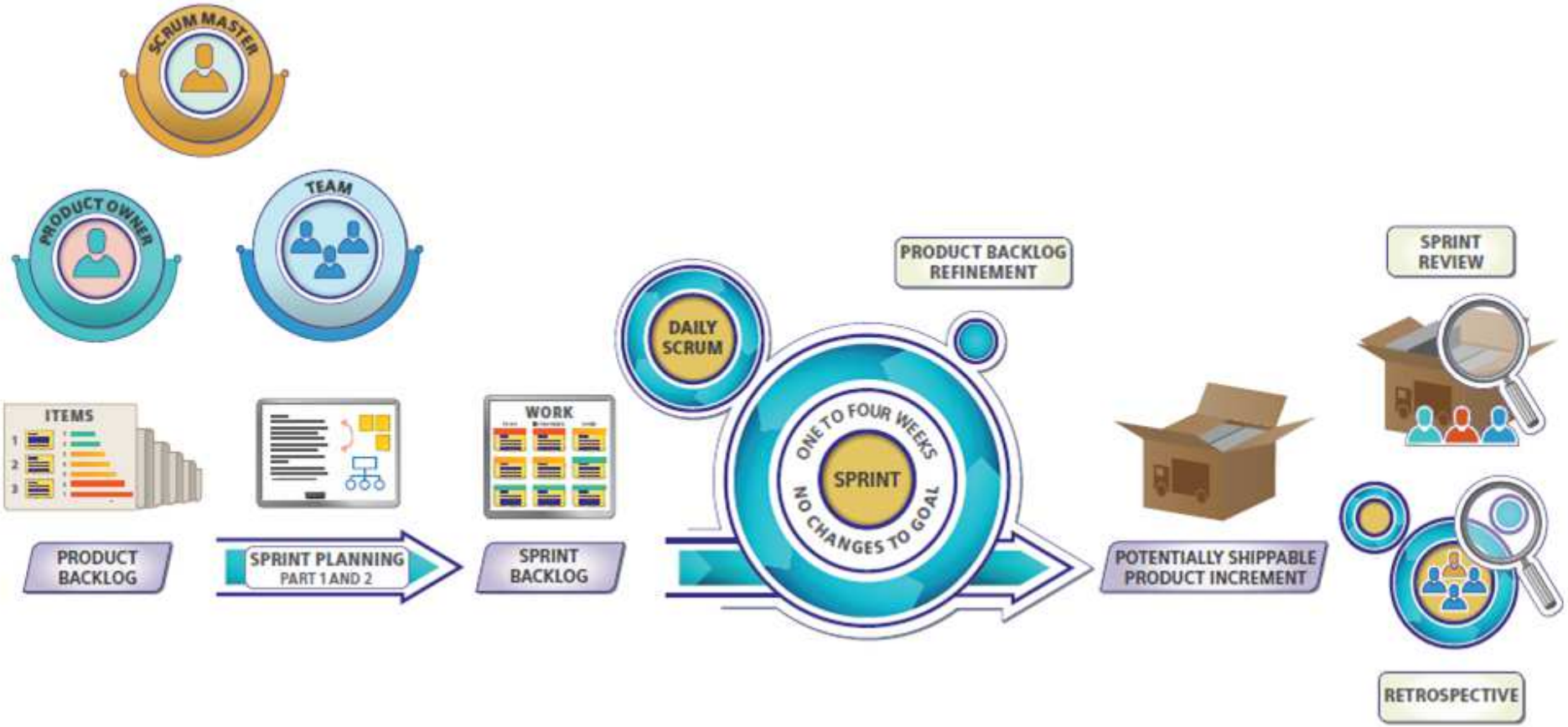
Sprint Review Meeting

- Length ≤ 4 hours at the end of a Sprint
- Scrum teams and all stakeholders participate
- Informal meeting, not a status meeting
- Discussion on the last Sprint
- The Product Owner reports on the achievement of the Sprint goal
- The Team reports on encountered issues and shows the new delivered increment
  - Incomplete work is not shown
- Product Backlog is discussed

Retrospective Meeting

- Length ≤ 3 hours at the end of the Sprint
- The entire Scrum team participates
- Assessment of team dynamics and tools
- Adjustment of process, team and interactions to make the next Sprint more efficient and productive
- Plan on how to implement the improvement

Scrum Sprint: Synthesis



Source: Scrum Primer

References

- **Agile Manifesto**
  - http://agilemanifesto.org/
- **Agile Alliance**:
  - http://www.agilealliance.org/
- **Scrum**
  - http://www.scrum.org/
- **Scrum Guide**
  - http://www.scrumguides.org/
- **The Scrum Primer**: A Lightweight Guide to the Theory and Practice of Scrum
  - http://www.scrumprimer.org/