Agile Methodologies

Introduction

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
     - o People are the most important ingredient for success, even though a bad process can make great people ineffective
  2. Working software over comprehensive documentation
     - o 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
     - o Successful projects involve customer feedback on a regular and frequent basis
  4. Responding to change over following a plan
     - o 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**
  2. Welcome changing requirements, even late in development
  3. **Working software is delivered frequently**
  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

- **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

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

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. **<u>Sustainable development, able to maintain a constant pace</u>**
6. **<u>Close, daily cooperation between business people and developers</u>**
7. **<u>Face-to-face conversation is the best form of communication</u>**
8. Continuous attention to technical excellence and good design
9. Simplicity - the art of maximizing the amount of work not done - is essential
10. Self-organizing teams
11. Regular adaptation to changing circumstances

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

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. **<u>Projects are built around motivated individuals, who should be trusted</u>**
9. **<u>Continuous attention to technical excellence and good design</u>**
10. Simplicity - the art of maximizing the amount of work not done - is essential
11. **<u>Self-organizing teams</u>**
12. Regular adaptation to changing circumstances

- **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**

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

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

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

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
  - Eventually, the achieved architecture shall be the *best* architecture for the eventual software
  - At runtime, the current architecture shall be the *best* architecture for the current software

Causes of Poor Agile Design

- Violation of the following design principles
  - Single-Responsibility Principle
    - High cohesion of a component, i.e. high relatedness of its elements and behaviors
  - Open/Closed Principle
    - Open to changes, close to modifications (no cascades of modifications)
  - Liskov Substitution Principle
    - Subtypes shall be replaced by base types systematically, so that extensions do not cause avoidable changes
  - 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
    - 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 too great
- 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
- **<u>Opacity</u>**
  - 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

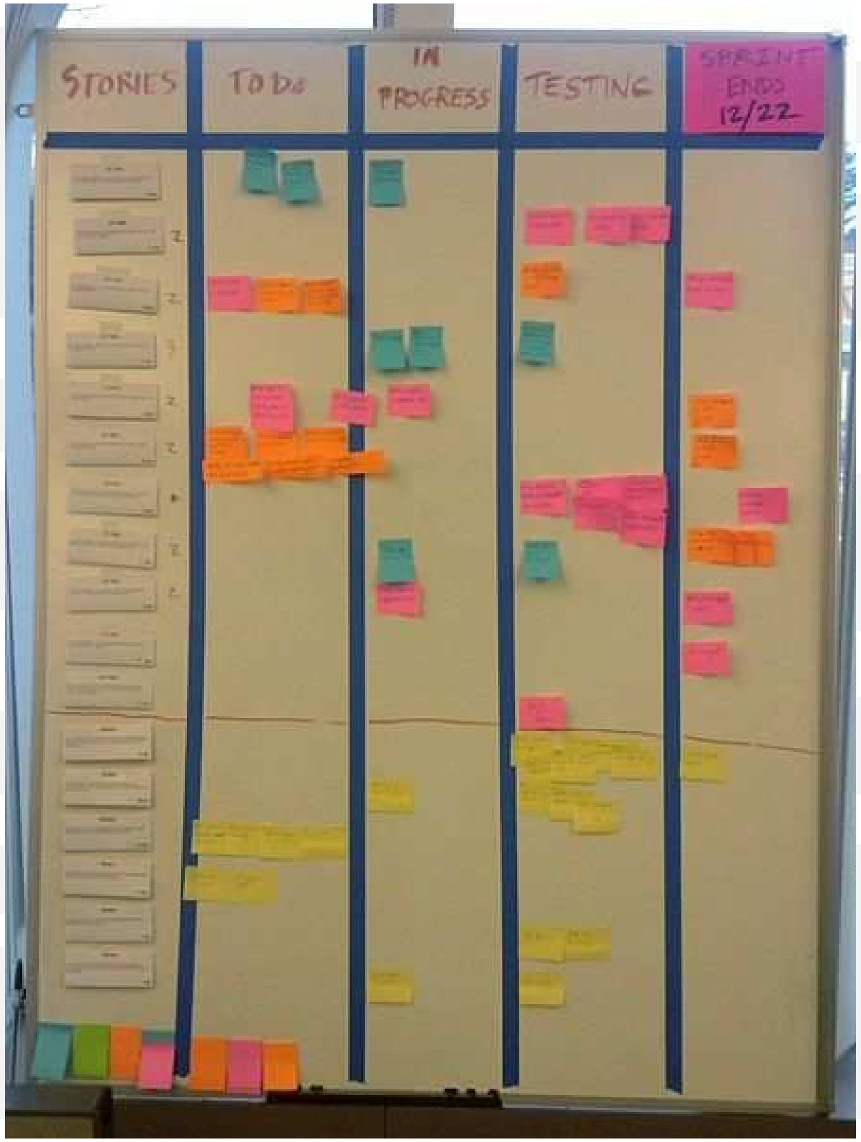
Scrum: Sprint

- Based on the concept of *Sprint*
  - Single iteration of fixed length throughout the development
  - Each Sprint ends with an increment on the product (e.g. a new functionality)
  - A new Sprint starts immediately after the end of the last Sprint

Scrum: Product Backlog and Story Points

- The product backlog is the current list of customer’s requirements – generally “user stories” –, i.e. the desired functionality, changes or enhancements
- Each user-story is assigned an amount of “story points”, i.e. a number *somehow* representing the estimated
- Product backlog is continuously in evolution
- Items in the product backlog are sorted by priority
  - Priority is assigned based on risk, value and necessity
- Only items on top of the product backlog are detailed at fine-grained level

Scrum Board



Scrum: Team and Roles

- The Scrum team consists of 7±2 members
- Three roles
  - Scrum Master
    - Team support and process monitoring, control, and coaching
    - Not the team leader, but responsible for removing impediments for achieving the Sprint goals
  - Product Owner
    - Manage and control the product backlog
    - Responsible for the value of the work done and for assigning priorities
    - Represents the voice of the customer
  - Development Team
    - Developers who turn the product owner's requests into shippable software

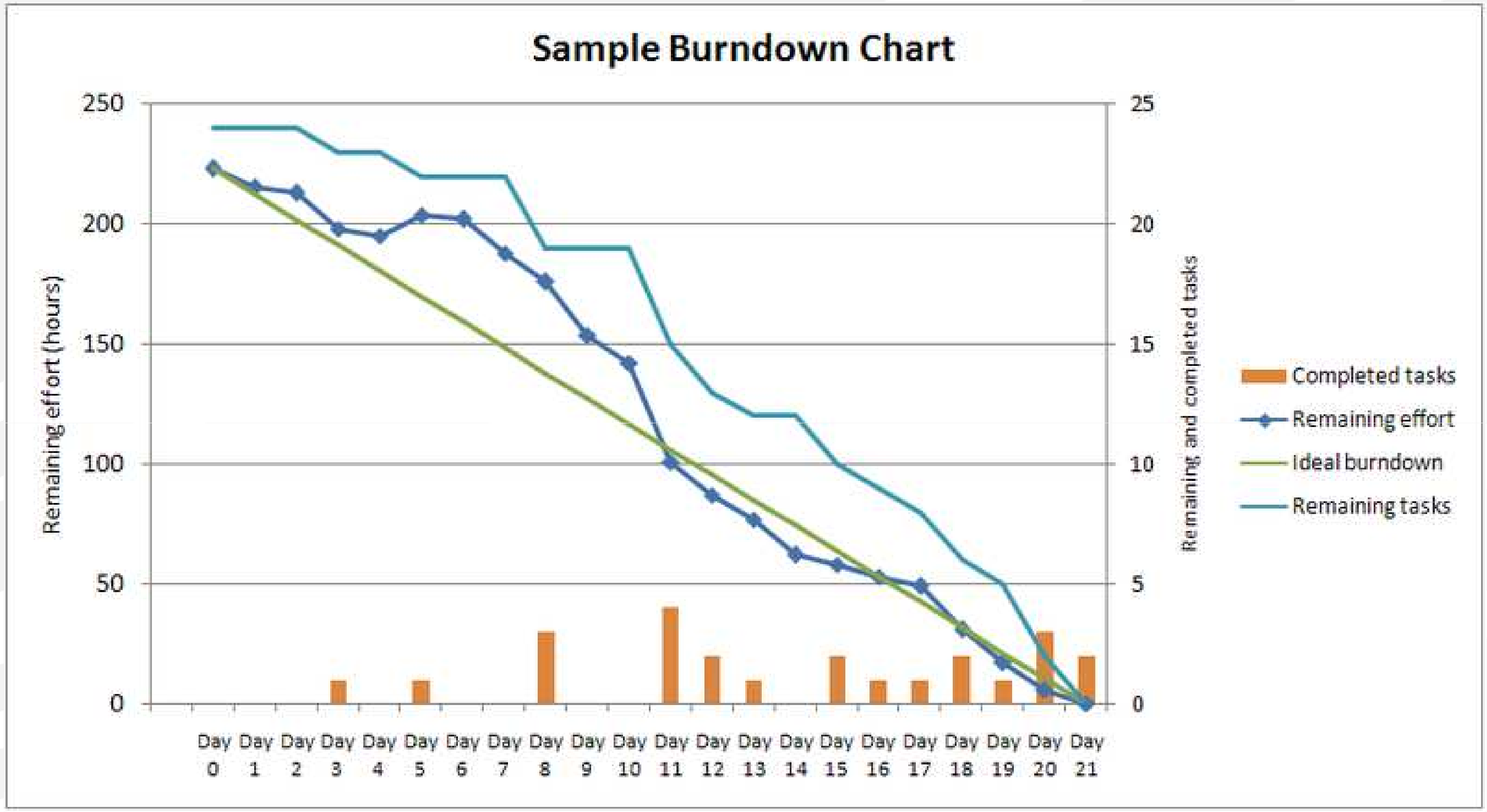
Scrum: Meetings

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

Scrum: Meetings

- Sprint Planning Meeting
  - ~4-8h
  - Product Owner presents the top priorities of the next Sprint
  - Decision on what stories from the product backlog are planned for the next Sprint
  - <u>During each Sprint, the same amount of story points shall be burnt</u>
    - The amount of story points burnt is called velocity and represents the development pace (velocity)
  - The Development Team decomposes stories into tasks to perform
    - Each task should require less than 1 day
    - Such tasks form the Sprint Backlog
    - Sprint backlog can change during the Sprint, according to the new knowledge acquired by the team
- Daily Scrum Meeting
- Sprint Review Meeting
- Retrospective Meeting

Scrum: Burndown Chart



Scrum: Meetings

- Sprint Planning Meeting
- Daily Scrum Meeting
  - ~15 minutes
  - Each team members shares
    - Yesterday progress
    - Today plans
    - Encountered problems
- Sprint Review Meeting
- Retrospective Meeting

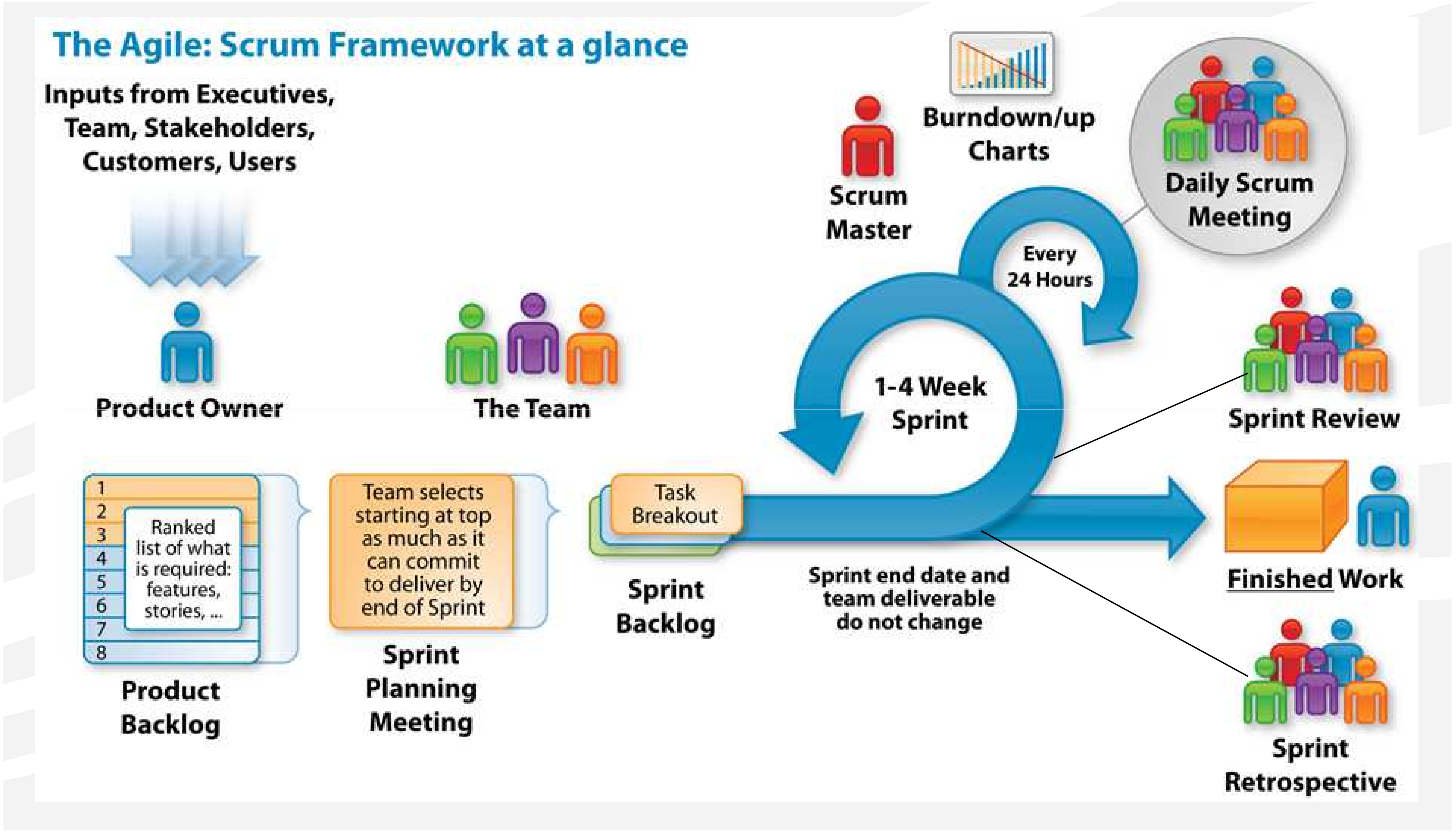
Scrum: Meetings

- Sprint Planning Meeting
- Daily Scrum Meeting
- Sprint Review Meeting
  - Informal meeting
  - ~2-4 hours
  - Discussion on the last Sprint with all stakeholders
  - The Product Owner reports on the achievement of the Sprint goals
  - The Team reports on encountered issues and shows the new delivered increment
    - Incomplete work is not shown
- Retrospective Meeting

Scrum: Meetings

- Sprint Planning Meeting
- Daily Scrum Meeting
- Sprint Review Meeting
- Retrospective Meeting
  - ~2-4 hours
  - Assessment of team dynamics and tools
  - Adjustment of process, team and interactions to make the next Sprint more efficient and productive

Scrum Process


The Agile: Scrum Framework at a glance
Inputs from Executives, Team, Stakeholders, Customers, Users
Product Owner
The Team
Scrum Master
Burndown/up Charts
Daily Scrum Meeting
Every 24 Hours
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Ranked list of what is required: features, stories, ...
Product Backlog
Team selects starting at top as much as it can commit to deliver by end of Sprint
Sprint Planning Meeting
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Sprint Retrospective

Source: http://www.softwaysolutions.com/

Scrum: Clustering Teams

- When more teams are working on the same project, synchronization is required
- A new meeting is introduced: Scrum of Scrums
  - Each day after the Daily Scrum
  - Same agenda, plus questions at team-level
    - Team’s yesterday progress
    - Team’s today plans
    - Problems that are slowing down the team
    - Possible impacts on other teams’ development path