# Chapter 3

# Agile Software Development

Compiled by Yohannes S.



Ian Sommerville, *Software Engineering*, 10th Edition
Pearson Education, Addison-Wesley

## Topics covered

- Agile methods
- Plan-driven and agile development
- Agile development techniques
- Agile project management
- Scrum
- Scaling up agile methods
- Exercise: Select suitable Agile project management tool

# The Problem(s) with Software Development

- Customers do not know exactly what they need or want
- Customers change their minds
- Customers' priorities change
- It is difficult to solve the needs of many stakeholders
- We can never *adequately* specify requirements
- We are lousy at estimating software tasks
- Unpredictable "stuff" happens
- It often takes several times to get "something" right

Users do not know what they want until they see it (but are good at telling us what they do not like)

## Rapid software development

✧Rapid development and delivery is now often the most important requirement for software systems

- Businesses operate in a fast–changing environment
- Software has to evolve quickly to reflect changing business needs

✧Rapid software development

- Specification, design and implementation are inter-leaved
- System is developed as a series of versions with stakeholders involved in version evaluation

# Agile Methods

✧Dissatisfaction with the overheads involved in software design methods led to the creation of *agile methods*. These methods:

- Focus on the code rather than the design
- Are based on an iterative approach to software development
- Are intended to deliver working software quickly and evolve this quickly to meet changing requirements

✧The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework

## 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:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

- http://agilemanifesto.org/

# Principles behind the Manifesto

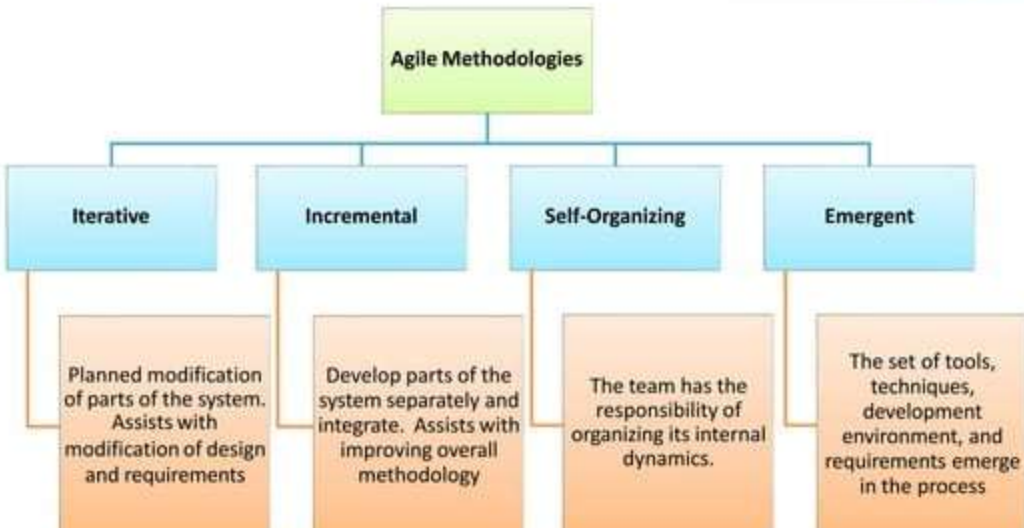
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of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

# Principles behind the Manifesto

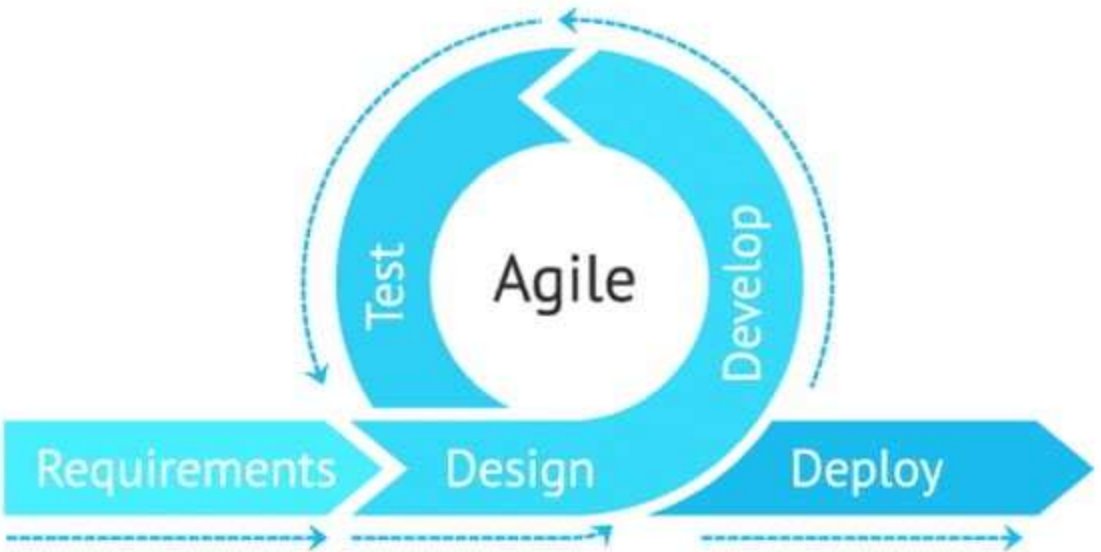
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the 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.

*Source:* *http://agilemanifesto.org/principles.html*

# Key Aspects of Agile Methodology



## Agile in Picture



## Agile method applicability

✧ Product development where a software company is developing a small or medium-sized product for sale

✧Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software

✧Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems

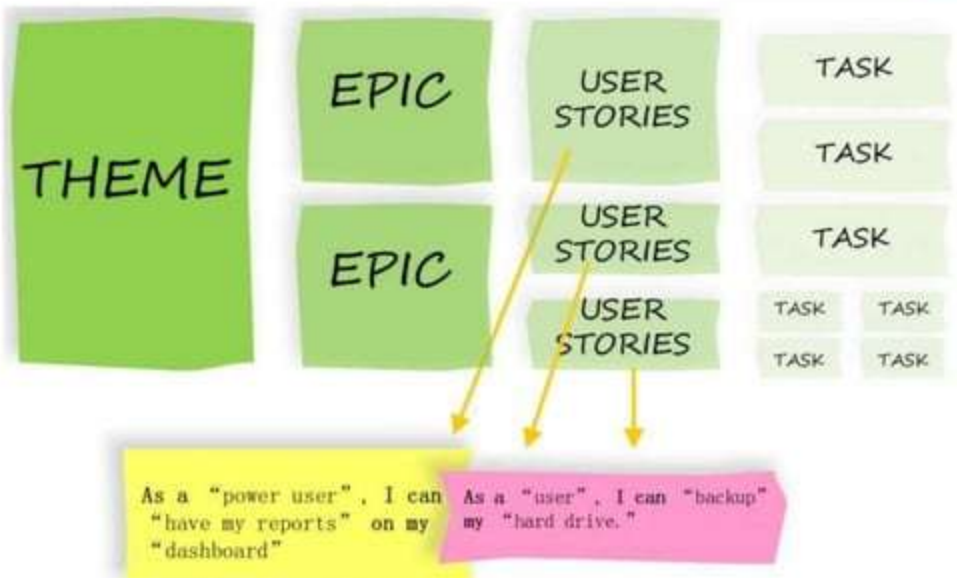
## Plan-driven and agile development

- Plan-driven development
  - Based around separate development stages with the outputs to be produced at each of these stages planned in advance
  - Not necessarily waterfall model – plan-driven, incremental development is also possible
  - Iteration occurs within activities
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process


# Waterfall vs. Agile

| Traditional | Agile |
|---|---|
| Detailed plan | Dynamic plan |
| Sequential phases conducted by documentation<br>Errors may be magnified | Iterative phases conducted by product delivery (by prioritizing)<br>Errors may be minimized |
| Functionality at the end<br><br>One Long period of time | Functionality after each iteration: Increments of the system every thirty days or less<br>Short periods of time |
| Project Manager | Team self management |
| Requirements Analysis | Customers working with development team (every iteration) |
| Reports (Docs) | Communication |

# Agile Terminologies: Theme, Epic, Story, Task



# Theme, Epic, Story, Task

## Stories – aka Product Backlog Item, Work Item

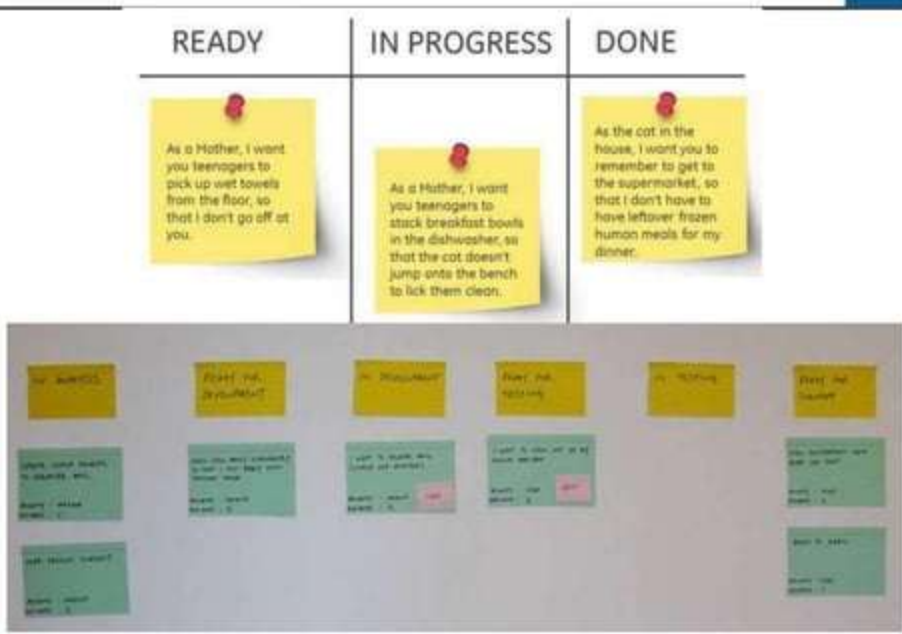
- A story is an individual feature or requirement that the client (and business) wants. It is something that is deliverable (i.e. production ready) within a single sprint.
- Stories are often written in a specific format:
  - As a [end user of the required feature]
  - I want [actual thing the user wants to be able to do once the feature is live – so often contains a verb]
  - So that [why they want this feature / the benefit this feature brings]
- *An example story:* ***As a customer, I want to be able to save a product in my wishlist so that I can view it again later.***

## Story Walls

- A story wall is a very common and effective information radiator that displays the status of each card in the current iteration or sprint.
- The team moves a card through each column as it gets completed.

# Story Walls


Software Engineering


READY
IN PROGRESS
DONE
As a Mother, I want you teenagers to pick up wet towels from the floor, so that I don't go off at you.
As a Mother, I want you teenagers to stack breakfast bowls in the dishwasher, so that the cat doesn't jump onto the bench to lick them clean.
As the cat in the house, I want you to remember to get to the supermarket, so that I don't have to have leftover frozen human meals for my dinner.

## Epic

- An epic is a big story. A requirement that is just too big to deliver in a single sprint.
- Epics need to be broken into smaller deliverables (stories).
- *An example epic: "As a customer, I want to be able to have wishlists so that I can come back to buy products later."*


Big, coarse-grained, sketchy

Small, detailed, specific

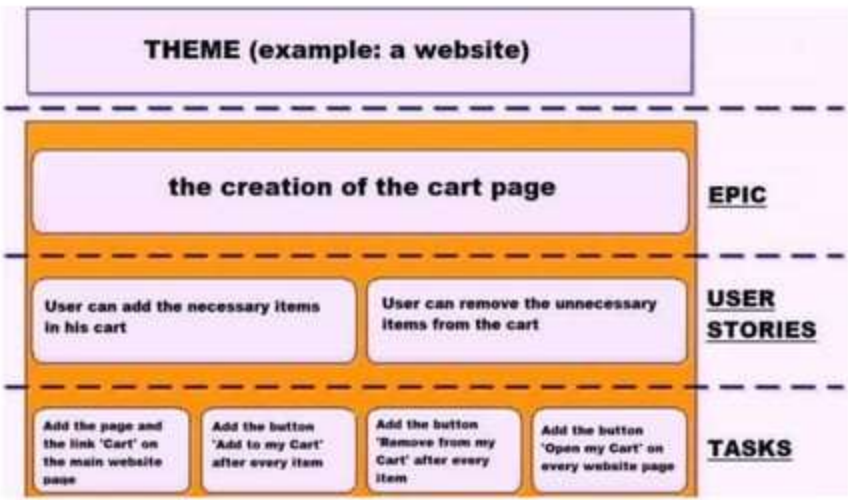
## Theme

- A collection of stories by category. A basket or bucket of stories. By its nature, an epic can also be a theme in itself.
- *An example theme: "Wishlist".*

## Tasks

- The elements of a story.
- Stepping stones to take the story to 'Done'.
- Tasks often follow the SMART acronym: specific, measurable, achievable, relevant, time-boxed.
- *An example task: "Put 'Add to wishlist' button on each product page."*

# Theme, Epic, User Stories, Tasks



## Agile Implementations

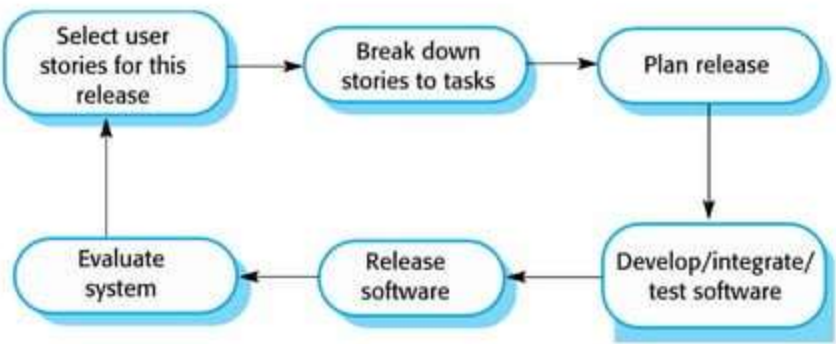
- Extreme Programming
- Scrum
- Kanban
- Lean and Six Sigma

Reading Assignment

## Extreme programming

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

# The extreme programming release cycle



# Extreme programming practices (a)

| Principle or practice | Description |
|---|---|
| Incremental planning | Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. |
| Small releases | The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release. |
| Simple design | Enough design is carried out to meet the current requirements and no more |
| Test-first development | An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented. |
| Refactoring | All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable. |

# Extreme programming practices (b)

| | |
|---|---|
| Pair programming | Developers work in pairs, checking each other's work and providing the support to always do a good job. |
| Collective ownership | The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything. |
| Continuous integration | As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass. |
| On-site customer | A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation. |

# Scrum

✧Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.

✧There are three phases in Scrum.

- The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
- This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
- The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

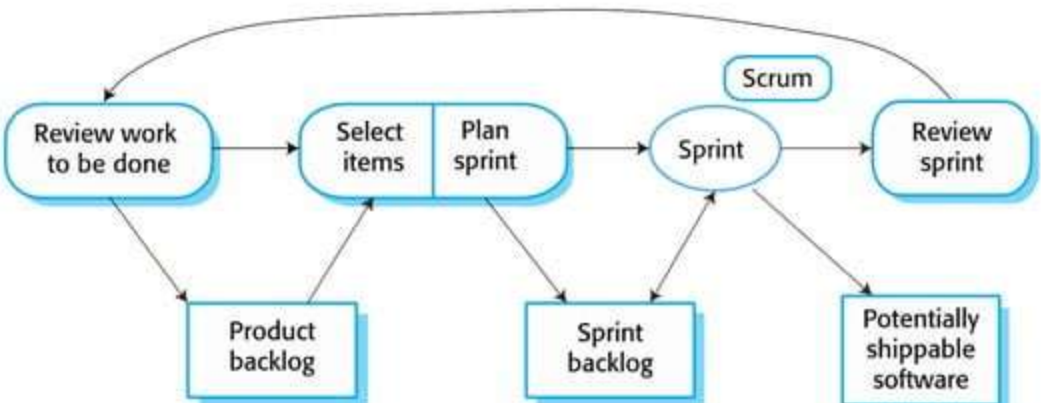
# Scrum terminology (a)

| Scrum term | Definition |
|---|---|
| Development team | A self-organizing group of software developers, which should be no more than 9 people. They are responsible for developing the software and other essential project documents. |
| Potentially shippable product increment | The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product. In practice, this is not always achievable. |
| Product backlog | This is a list of 'to do' items which the Scrum team must tackle. They may be feature definitions for the software, software requirements, user stories or descriptions of supplementary tasks that are needed, such as architecture definition or user documentation. |
| Product owner | An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical customer needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative. |

# Scrum terminology (b)

| Scrum term | Definition |
|---|---|
| Scrum | A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team. |
| ScrumMaster | The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference. The Scrum developers are adamant that the ScrumMaster should not be thought of as a project manager. Others, however, may not always find it easy to see the difference. |
| Sprint | A development iteration. Sprints are usually 2-4 weeks long. |
| Velocity | An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance. |

# Scrum sprint cycle



## The Scrum sprint cycle

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.

## The Sprint cycle

- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called 'Scrum master'.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

## Teamwork in Scrum

- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

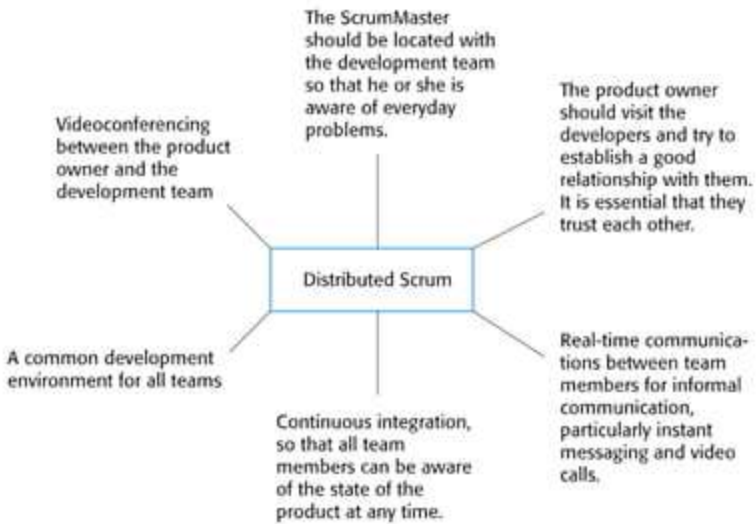
## Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

# Agile Stakeholders

| Product Manager | Product Owner | Team |
| --- | --- | --- |
| Market/Customer facing. Identifies market needs. Collocated with marketing/ business. | Solution, technology, and team facing. Collocated with team(s). | Customer/stakeholder facing. |
| Owns vision and roadmaps, program backlog, pricing, licensing, ROI. | Contributes to vision and program backlog. Owns team backlog and implementation. | Owns story estimates and implementation of value. |
| Drives PI objectives and release content via prioritized features and enablers. | Defines iterations and stories. Accepts iteration increments. | Contributes to intentional architecture. Owns emergent design. |
| Establishes feature acceptance criteria. | Drives iteration goals and iteration content via prioritized stories. | Contributes to backlog refinement and creation of stories. |
| | Establishes story acceptance criteria, accepts stories into the baseline. | Integrates with other teams. |

# Distributed Scrum



# Scrum vs. Extreme Programming

| Scrum | XP |
| --- | --- |
| • Changes in sprint are not allowed<br>• Once tasks for a certain sprint are set, the team determines the sequence in which they will develop the backlog items<br>• The Scrum Master is responsible for what is done in the sprint, including the code that is written<br>• The validation of the software is completed at the end of each sprint, at Sprint Review | • As long as the team hasn't started working on a particular feature, a new feature, of equivalent size can be swapped into the interation in exchange for an un-started feature<br>• Tasks are taken in a strict priority order<br>• Developers can modify or refactor parts of code as the need arises<br>• The software needs to be validated at all time, to the extent that tests are written prior to the actual software |

## Large systems development

✧Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones. E.g. development of ERP systems (read about this)

✧Large systems are 'brownfield systems', that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.

✧Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.

## Class Discussion: 5 marks

Greenfield vs. Brownfield System/Project ?

Greenfield: A brand new one, developed from scratch, and not based on any existing systems, code or infrastructure

Brownfield: Built upon already existing projects, code or other resources.

## Large systems development

✧Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed

✧Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.

✧Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

## Scaling out and scaling up

- Scaling up is concerned with using agile methods for developing large software systems that cannot be developed by a small team
- Scaling out is concerned with how agile methods can be introduced across a large organization with many years of software development experience
- When scaling agile methods it is essential to maintain agile fundamentals
  - Flexible planning, frequent system releases, continuous integration, test-driven development, and good team communications

## Scaling up to large systems

- Scaling up focuses on applying agile principles to develop large software systems.
- Why- these systems are too complex or large to be developed by a small team.
- Approach - instead of having just one agile team, multiple agile teams are coordinated to work on parts of the system.
- Goal - deliver large, cohesive software products without losing agility.

Example: For a large ERP system, scaling up might involve breaking the system into modules and assigning multiple agile teams to work on specific features, such as Finance, HR, and Inventory Management.

## Scaling out to large companies

- Scaling out focuses on spreading agile practices across a large organization.
- Key Challenge - many organizations are used to old, traditional methods (waterfall)
- Approach - introduce agile methods gradually, ensuring teams adopt agile values (flexibility, collaboration, iteration).
- Goal - promote a cultural shift where the entire organization operates with agile fundamentals.

Example: If your company has several departments (e.g., software, quality assurance, IT support), scaling out ensures all teams use agile methodologies like sprint planning, stand-ups, and frequent releases.

## Key Takeaway

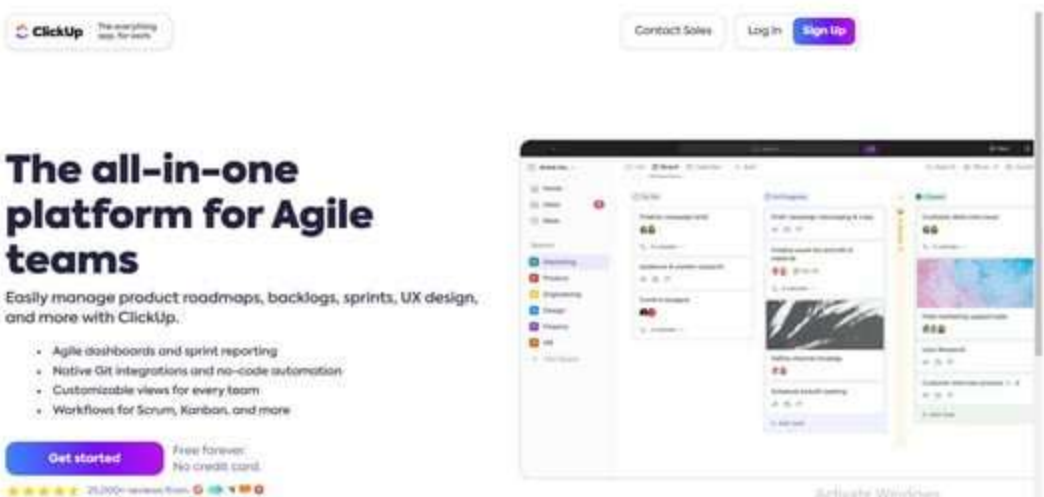
- **Scaling Up** ⟶ Focuses on managing large systems with multiple agile teams
- **Scaling Out** ⟶ Focuses on adopting agile practices across the organization
- Both approaches rely on **agile fundamentals** to succeed

# Agile project management

- Rol of Software Project Managers: Ensure software is delivered on time and within budget.
- Plan-Driven Approach: Managers create a detailed plan specifying
  - **What** will be delivered,
  - **When** it will be delivered, and
  - **Who** will develop the deliverables.
- Agile Approach: Focuses on incremental development and practices used in agile methods, requiring a more flexible and adaptive approach.

# Exercise

- Go through a list of free Agile Project Management tools
- Select a tool of your choice for your project. (Jira, Asana, Monday.com, ClickUp, IceScrum)



# Exercise/Project

- Form a team containing not more than five members
- Select project title
- Define objective/s, purposes
- Conduct feasibility study
- Write about the system's overview
- Select the process model best fits for applying to the development of your system
- Study software project management tools
- Select preferable tool
- Practice tool
- Apply scrum schemes (create product backlogs, create themes, create epics, write user stories, write tasks)
- Provide a light-weight requirement specification document (keeping its agility)
- Incorporate common system models that can represent the user stories you write.

## Project Deliverable

✓ Comprehensive User Story (Light-weight requirements specification document) contains:

- Descriptive title
- Objectives/purposes
- System overview
- Agile requirements specification: List of product backlogs, themes, epics, user stories and tasks
- Some requirements modeling (such as use case, sequence, activity, class) based on the specified user stories
- Scrum teams (product owner, development team and scrum master) with their roles and responsibilities. Assign the roles from your team members
- The name and number of sprints including the list of features implemented in each sprint with their duration.
- Practice with the aid of tools

# Thank You

Any question? !?