Concepts, Principles, Phases and Characteristics of the Agile Development

Differences between Agile and Traditional Development

Vassilis C. Gerogiannis
(vgerogian@uth.gr)

Learning outcomes

- At the end of this presentation, you will be able to:
  - Describe the basic concepts of the agile model.
  - Identify (at least) three advantages of following an agile approach.
  - Indicate (at least) three differences between agile SDLC and traditional SDLC approaches (e.g., Waterfall).

Aim and objectives

- The aim of this presentation is to introduce the learners to the basic concepts of agile software development.
- The presentation describes the basic concepts, principles, phases and characteristics of the agile SDLC model.
- The presentation emphasizes on the differences between agile software development and traditional software development approaches (e.g., Waterfall).

Terms and keywords

- Agile Software Development Life-Cycle (SDLC) Model
  - Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. An agile SDLC model defines a software process which follows the Agile Principles and applies an Agile Software Development Method.
- Agile Manifesto (Agile Principles)
  - The Agile Manifesto is the foundation of most Agile Software Development methods. It has 4 core values supplemented by 12 Principles. The document was developed in 2001 by a group of 17 pioneer software engineers (www.agilemanifesto.org).
- Agile Software Development Methods
  - Agile Software Development Methods (or Agile Methods) is a group of methodologies for developing software based on the guidelines of the Agile SDLC model and the principles of the Agile manifesto. Representative methods are: Extreme Programming, SCRUM, KANBAN, DSDM, FDD, Crystal Family of methods etc.

Table of contents

Introduction to Agile Software Development

Traditional Software Development (SD)

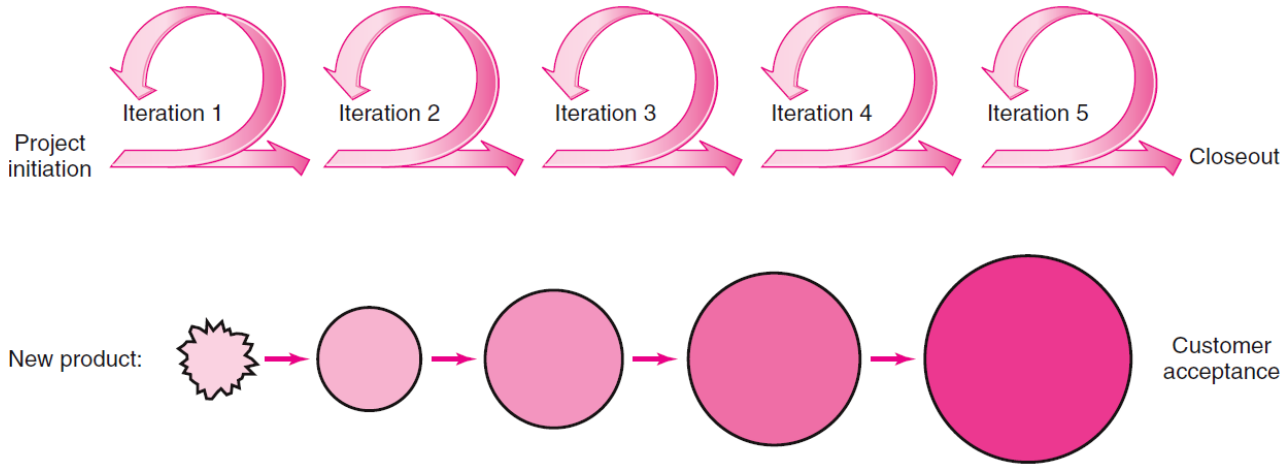
- Traditional software development life-cycle (SDLC) models (e.g., the Waterfall method or variations such as the V-Model) are rigid and heavyweight:
  - They follow a sequence of pre-determined stages (requirements definition, design, implementation, testing and deployment).
  - They require a stable and fully known set of requirements at the very beginning of a project.
  - They require detailed documentation to guide all development activities.
  - Handling changes during the development lifecycle can be difficult.
- In traditional SDLC models, the project success relies on knowing all software requirements before design and implementation begin.

Agile Software Development (SD)

- Agile software development (SD) is a **group of software development methods** which adopt **iterative and incremental development** to encounter the limitations of traditional SD methods.
- Agile SD methods utilize and further extend the idea/use of iterative/incremental development adopted by evolutionary prototyping, incremental development and Spiral.
- Iterative/Incremental development starts with an initial planning and results in the final system deployment, though applying iterations in the between, each one delivering an evolutionary increment of a software artifact.

Phases in Agile SD (1/2)

- Phases within an agile SD life-cycle are repeatedly executed and continuously revisited.
- Development activities iteratively enhance the functionality of the software by considering customer feedback.
- The development life-cycle is divided into small parts, called "**increments**" or "**iterations**".



Phases in Agile SD (2/2)

- Iterations are applied to each of the conventional tasks of traditional SD (requirements definition, design, programming, testing).
- Each iteration incrementally evolves the software product by developing, almost on a daily basis, additional user requirements (also called **user stories**), implemented in the form of software functionalities, which are called **software features**.
- A series of iterations develops a software **release**, i.e., a version of the system that can be successfully deployed.

Basic Concepts in Agile SD (1/2)

- In Agile SD requirements and their implementation solutions (software components) evolve through **collaboration between:**

  self-organizing and **cross-functional teams**

  by exploiting **continuous involvement of the users**.

- Agile SD adopts:

  adaptive planning and **evolutionary development/delivery** of software

  to achieve **end-user/customer satisfaction**.

Basic Concepts in Agile SD (2/2)

- Agile SD follows a "**time-boxed" iterative approach** to provide rapid and flexible/adaptive response to changes.
- Agile SD methods are '**lightweight**' and '**human-centric**' with the aim to **create business value**.
- Agile SD can be regarded as a conceptual framework that promotes foreseen interactions among stakeholders throughout the SD life-cycle.
- Agile SD life-cycle model is a software process which follows the **agile principles** and applies an **Agile SD method**.

Representative Agile SD Methods

- SCRUM
- Extreme Programming (XP)
- KANBAN
- Crystal methods
- Dynamic Software Development Method (DSDM)
- Feature-Driven Development (ADD)
- etc.

Characteristics of Agile methods

- Emphasis on code rather than design.
- Follow an iterative approach to SD.
- Target to deliver working software quickly.
- Evolve quickly to meet changing requirements.
- Assume active customer involvement.
- People-based rather than Plan-based development.

The 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.

Source: The Agile Manifesto (2001). *Manifesto for agile software development*. http://www.agilemanifesto.org

Twelve Agile Principles based on the Agile Manifesto (1/3)

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.

Source: The Agile Manifesto (2001). *Principles behind the Agile Manifesto*. http://www.agilemanifesto.org/principles.html

Twelve Agile Principles based on the Agile Manifesto (2/3)

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.
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.

Source: The Agile Manifesto (2001). *Principles behind the Agile Manifesto*.
http://www.agilemanifesto.org/principles.html

Twelve Agile Principles based on the Agile Manifesto (3/3)

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: The Agile Manifesto (2001). *Principles behind the Agile Manifesto*. http://www.agilemanifesto.org/principles.html

Advantages of using the Agile Methods

- Adaptability to changing market/customer needs
- Better cost efficiencies and fastest time-to-market
- Improved quality, customer/developer satisfaction, and project success

Limitations of Agile Methods

- May do not satisfy top management’s needs for tight budget, scope, and schedule control.
- Self-organization and intensive collaboration can be incompatible with the culture of some corporates.
- Agile methods appear to be effective for small/medium-scale projects that require only five-nine dedicated team members to complete the work.
- Active customer involvement/cooperation is not always possible.

Characteristics of Agile Software Development

Agile Characteristics based on the Agile Principles

- Adaptive planning
- Evolutionary development with focus on customer value/satisfaction
- Continuous user involvement
- Self - organized / Cross - functional teams
- Lightweight Process
- Human-centric Approach

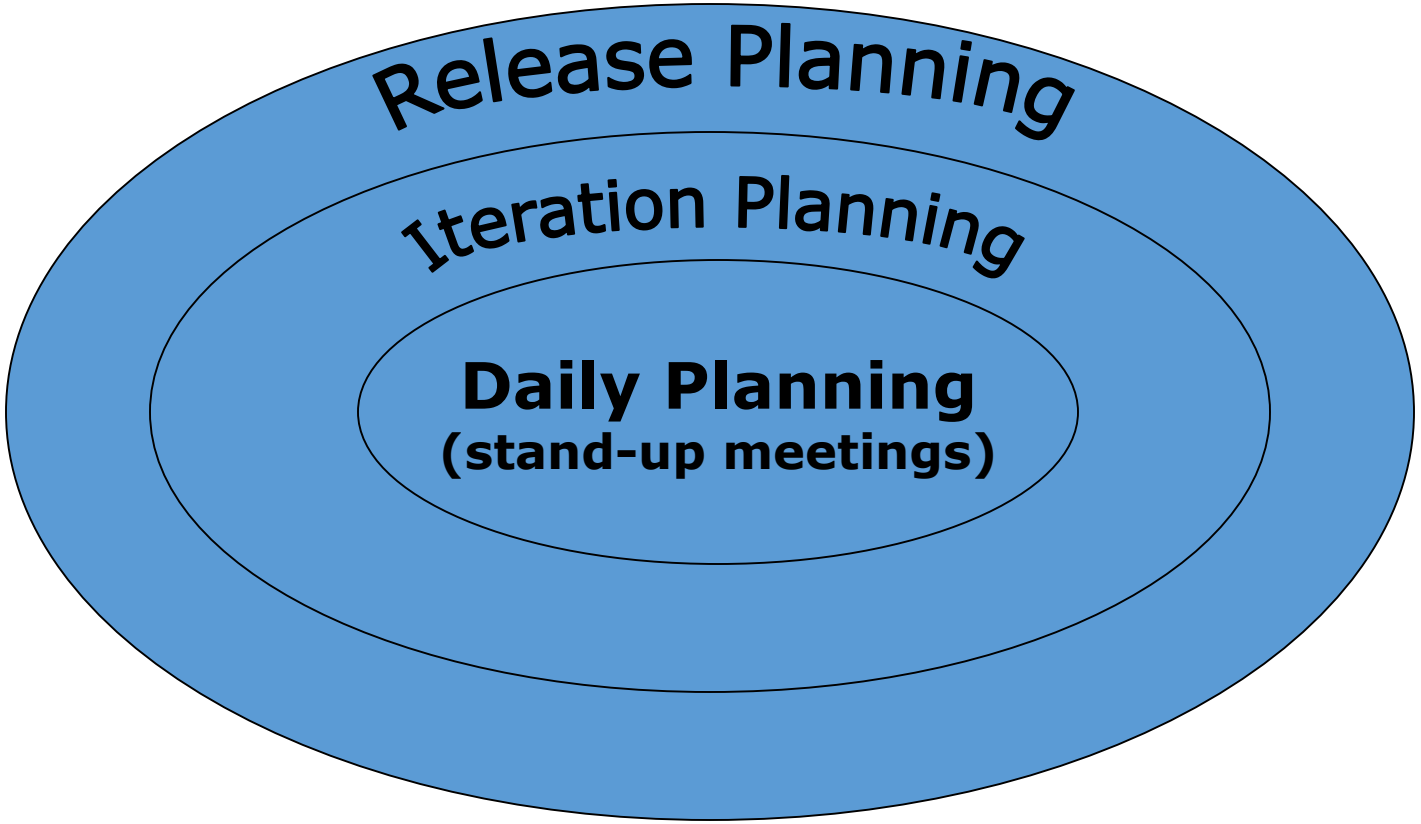
Adaptive Planning: High-level principles (1/2)

1. **Planning at multiple levels**: the amount of information considered at each planning level (release, iteration and daily planning) results in continuously delivering a high value product.
2. **Two-way communication**: Both team members and customers are involved in planning.
3. **Transparency**: Frequent deliveries and progress demonstration keep all stakeholders updated and informed.
4. **Handing uncertainty**: the process is adapted to fit with different types of project risks by **minimizing uncertainties, risk and need for rework**.
5. Consideration of the product backlog (i.e., the set of the candidate software features to be delivered) makes iteration planning adaptable and flexible: a specific person (e.g., the Product Owner in terms of the SCRUM method) is responsible for maximizing the product value resulting from the development work. The product owner considers how changes influence the release plan. Depending on the project needs and features' priorities, **the plan is updated continuously**.

Adaptive Planning: High-level principles (2/2)

6. Estimating risks of a project is not limited to the project schedule and cost. **Other risks are also considered** (e.g., the team members velocity to implement the features, their availability, any distractions that may occur etc.)
7. **Estimate ranges** are used when assuming uncertainties.
8. Project work projection is **based on the actual work progress** rather than a predefined theoretical plan.
9. All possibilities about resource work are considered (e.g., ideal working time, unproductive time, team member absences etc.) to **handle unexpected circumstances**.

Adaptive Planning: Planning levels



Adaptive Planning: Planning levels

- Release Planning (on a monthly basis)
  - Planning goals: overall project scope, external communication
  - Planning the implementation of the features
  - Decided by the customers and the product owner
- Iteration Planning (on 1-2 weeks basis)
  - Planning goals: scope of the next iteration, internal communication
  - Planning work tasks within an iteration
  - Driven by the development team members who prioritize software features based on their value, cost of implementation, risk etc.
- Daily Planning (daily standup meeting)
  - Primary goals: internal communication
  - Planning of individual work and coordination
  - Avoid unproductive time and duplicated effort
  - Not easy to “go dark” (difficult for a developer to try avoid working)

Adaptive Planning: Time-boxing

- In agile SD the speed of delivering software to customers is critical and the cost of delays is great.
- Agile Manifesto principle:
  - "*deliver working software frequently, ideally in a couple of weeks*"
- Time-boxing is a way to achieve this principle.
  - A time-box is the fixed time period during which development team members agree to complete a project activity.
  - Any iteration resulting in the delivery of an increment is time-boxed, by allocating a fixed period of time to the iteration development activities.
  - Any undone work is moved to the next iteration or into subsequent iterations.
- The ultimate objective is to establish a development rhythm for the software artifacts produced and delivered to customers.

Evolutionary SW Development (1/2)

- “Adaptation to change” is a key value in the Agile Manifesto.
- Evolutionary software development:
  - early, incremental and continuous delivery of working software.
  - The software is developed gradually with small portion at a time.
- Adaptive and evolutionary refinement of plans, requirements, designs and software features, at each iteration.
- Instead of defining and documenting all requirements upfront and refusing changes when implementing them, agile teams decide on requirements and features as late as possible.
- Requirements are gathered and implemented iteratively.

Evolutionary SW Development (2/2)

- For every iteration software components are presented to customers for review.
- These components are not integrated as a full system: and rework or additional features would not marginally increase the project cost.
- Developers are able to include features needed by the customers, and system integration will only occur when customers have no additional requirements.

The goal is to satisfy customers with a complete system containing all functions they desire

Continuous User Involvement

- Agile methodologies emphasize in early and continuous communication/involvement with the customers.
- In every iteration, the development team and the customers (or customer representatives) hold a meeting, where:
  - team members communicate and report their work done in the current iteration
  - customers provide feedback on the delivered software to refine current features and propose additional features to be developed in the next iteration.
- Frequent software releases allow customers to:
  - acquire sufficient knowledge on the current release of the system
  - provide feedback to refine the requirements provided in the current release
- Frequent software releases allow development team members to:
  - be well-informed and competent to consider needed adjustments that emerge during the development life-cycle

Self-organizing teams

- Agile teams are self-organizing and self-managing.
- Agile team members are accountable:
  - to deliver results that meet customer needs.
  - to each other and to other external teams for delivering quality work on time.
  - to keep a continuous flow of development in every iteration.

Cross-functional teams

- Cross-functional teams are groups of people from different functional areas:
  - Groups are formed not only with technical specialists (e.g., designers, developers, quality assurance engineers/testers, etc.) but also consist of persons like business analysts, marketing specialists or other people taking an active part in the project.
  - A cross-functional team possesses all required domain knowledge and skills, without the need to rely on others outside of the team.

Lightweight process

- Instead of “heavy” documentation, agile emphasizes on customer feedback and teams communication.
- Change is embraced: iterate often, design and redesign, code and test frequently, keep the customer involved and informed.
- Deliver software to the customer in short iterations (e.g., every 2 weeks).
- Identify and eliminate defects early, thus reducing costs.

Human-centric approach (1/2)

- Reliance on people and the interaction between them.
- Development is not process-centric, but human-centric.
- Frequent and effective personal communication and collaboration between the project stakeholders.
- Intensive teamwork and high task interdependence.
- Avoid ‘burnout’ of team members (allow a sustainable rhythm of work).

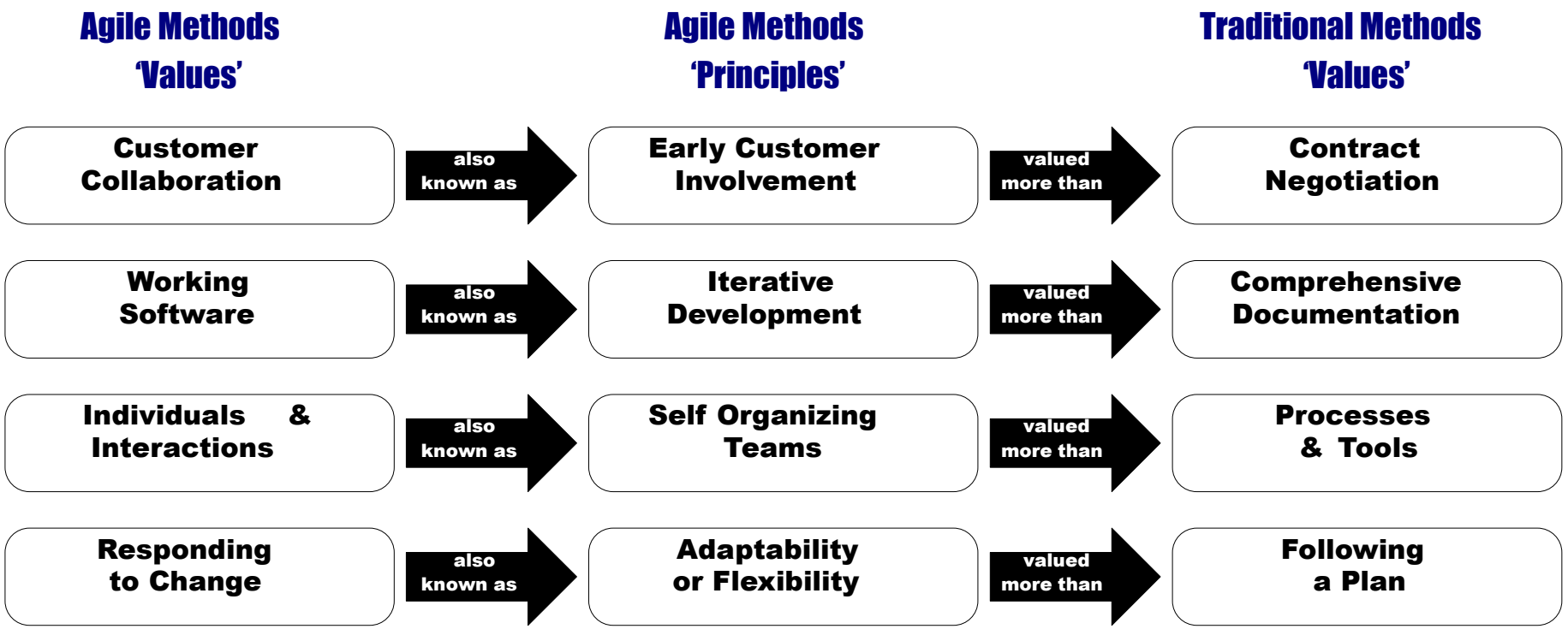
Human-centric approach (2/2)

- Co-located teams, where possible.
- Daily “stand-up” meetings, facilitated release planning workshops & retrospective meetings.
- “Agile” physical design of the workspace and other communication tools:
  - Open-plan workspace and offices where everyone (including developers, testers, business experts, etc.) sits and interacts
  - Effective use of physical tools for communication and collaboration (e.g., visible white boards, task boards, status boards)
  - Use of post-it notes to denote the software features and their status of implementation.
  - Use of cards (e.g., user story cards) to specify the requirements.

Agile vs. Traditional Software Development

Agile vs Traditional (Plan-based) Software Development

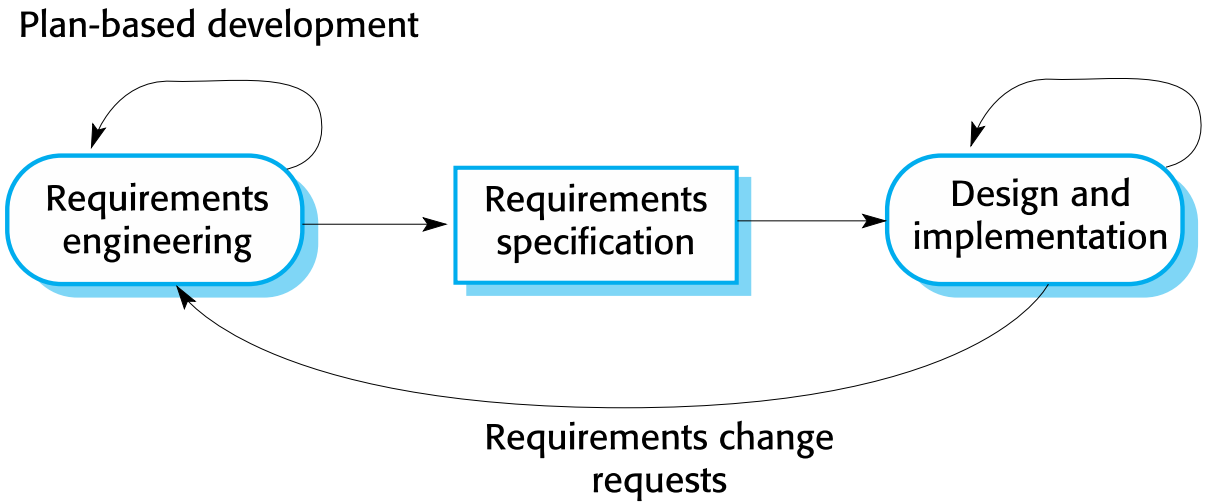
Differences in Values

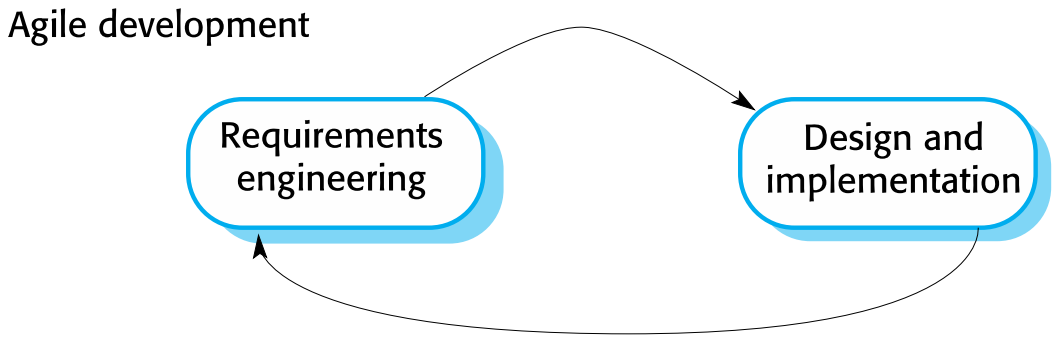


Source: The Agile Manifesto (2001). *Manifesto for agile software development*. http://www.agilemanifesto.org

Agile vs Traditional (Plan-based) Software Development

Differences in Process Flow





Comparison between Agile and Traditional Software Development (1/2)

Traditional methods:

- Concentrate on thorough, upfront planning of the entire project. Fixed project scope.
- Require ‘up-front’ requirement gathering.
- Design ‘up-front’.
- Freeze requirements and design as early as possible.
- Sequential execution. The development teams build the system in “one-shot” fashion.

Agile methods:

- Rely on incremental, iterative development cycles. Flexible project scope.
- Avoid ‘up-front’ requirement gathering. Agile has the ability to successfully deliver results quickly and inexpensively on complex projects with ill-defined requirements.
- Continuous design.
- Freeze requirements and design as late as possible.
- Iterative/incremental and evolutionary/adaptable execution.

Comparison between Agile and Traditional Software Development (2/2)

Traditional methods:

- Require a high degree of predictability to be effective. Good for projects with low uncertainty.
- The cost of change is high.
- Emphasis on contracts, plans, processes, documents, and tools.
- Conventional project teams dedicated to specific tasks. Team members having specific technical expertise.
- Low and late customer involvement.
- Coherence to plan is the key priority.

Agile methods:

- Ideal for less-predictable and exploratory projects in which requirements need to be discovered and new technology to be tested. Good for projects with high uncertainty.
- The cost of change is low.
- Emphasis on teams, working software, customer collaboration, and responding to change. Focus on active collaboration between the project team and customer representatives.
- Self-organized and cross-functional project teams. Team members having both technical and soft skills.
- High, early and continuous customer involvement.
- Customer satisfaction and business return on investment (ROI) are the key priorities.

Summary of differences (1/3)

	Traditional development	Agile development
Underlying principle	Software systems are specifiable and predictable. They can be developed through following detailed planning.	Software systems are adaptive to changes. High quality software is developed by small teams that use the iterative/continuous improvement based on customer feedback and changes.
Management style	Dictated by command and control.	Dictated by leadership and collaboration.
Communication model	Formal based on documents.	Informal based on productive collaboration.
Development model	Sequential.	Iterative/incremental/evolutionary.
Organizational structure	Bureaucratic, highly formalized, suited for large organizations.	Flexible, participative, encourages social cooperation, suited for small and medium organizations.

Summary of differences (2/3)

	Traditional development	Agile development
Testing and quality control activities	Sequential execution of validation/verification activities. Testing is performed late, after the code is completed.	Continuous testing the requirements, design and the implementation artifacts. Testing is performed in every iteration
Definition of user requirements	Requirements are defined in detail and specified formally before the design/implementation begins.	Requirements are defined rather informally and they are continuously gathered/evaluated in every iteration.
Cost of restarting the project activities	High.	Low.
Development style	Fixed.	Flexible.
Customer involvement	Low.	High.

Summary of differences (3/3)

	Traditional development	Agile development
Additional skills required from Developers	Technical skills are mainly required.	Technical skills and interpersonal/soft skills (e.g., communication skills) and knowledge of the business objectives.
Scale of projects	Medium scale projects.	Low and medium scale projects.
Requirements nature	Very stable, known in advance.	Unstable, emergent during the process
Architecture	Design represents all the specified requirements.	Design represents the requirements implemented in the current iteration.
Cost of redesign	Expensive.	Not expensive.
Size of the project team	Medium-large project teams.	Small-medium teams.
Primary objective	Adherence to the project plan.	Adherence to customer value.

List of references

- Cockburn, A. (2002). Agile Software Development. Addison-Wesley, 1st edition.
- Martin, R. C. (2002). Agile Software Development, Principles, Patterns and Practice. Prentice Hall, 1st edition.
- Larman, C. (2005). Agile and Iterative Development: A Manager's Guide. Pearson Education, 1st edition.
- Cohn, M. (2005). Agile Estimating and Planning. Prentice Hall. 1st edition.
- Boehm, B. (2002). Get Ready for Agile Methods, with Care, *IEEE Computer*, Vol. 35, Issue 1, pp. 64-69.
- Nerur, S., Mahapatra, R. & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies, *Communications of the ACM*, Vol. 48, Issue 5, 72– 78.
- Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional, *Informatica Economica*, Vol. 17, No. 4, 64-76.
- Leau, Y. B., Loo, W. K., Tham, W. Y. & Tan, S. (2012). Software Development Life Cycle Agile vs Traditional Approaches. In Proceedings of the *2012 International Conference on Information and Network Technology (ICINT 2012)*, IACSIT Press.
- Wikipedia: Agile Manifesto (2001). Retrieved on May 16, 2020 from: http://www.agilemanifesto.org