The Art of Project Management

As defined by Gartner, project management is “the application of knowledge, skills, tools and techniques to project activities to meet the project requirements”.

Being an integral part of software engineering processes along with the business analysis and requirement specification, design, programming and testing, the project management has been a topic of considerable debate for years. **Even today, when company project management practices are becoming more mature, only about half of them (54%), according to survey results by the Project Management Institute (PMI), are fully aware of the importance and value of these practices.**

Regardless of industry, project management has proven to be a crucial element of a company’s efficiency and its eventual success. In fact, the organizations using proven project management practices waste 28 less money and implement projects that are 2.5 times more successful.

Project management professionals conclude that the definition of a successful project is one that is not only completed on time and within budget, but one that also delivers expected benefits.

Project Management Phases

Regardless of the scope, any project should follow a sequence of actions to be controlled and managed. According to the Project Management Institute (PMI), a typical project management process includes the following phases:

1. Initiation
2. Planning
3. Execution
4. Performance/Monitoring
5. Project close

Used as a roadmap to accomplish specific tasks, these phases define the project management lifecycle.

Yet, this structure is too general. A project usually has a number of internal stages within each phase. They can vary greatly depending on the scope of work, the team, the industry and the project itself.

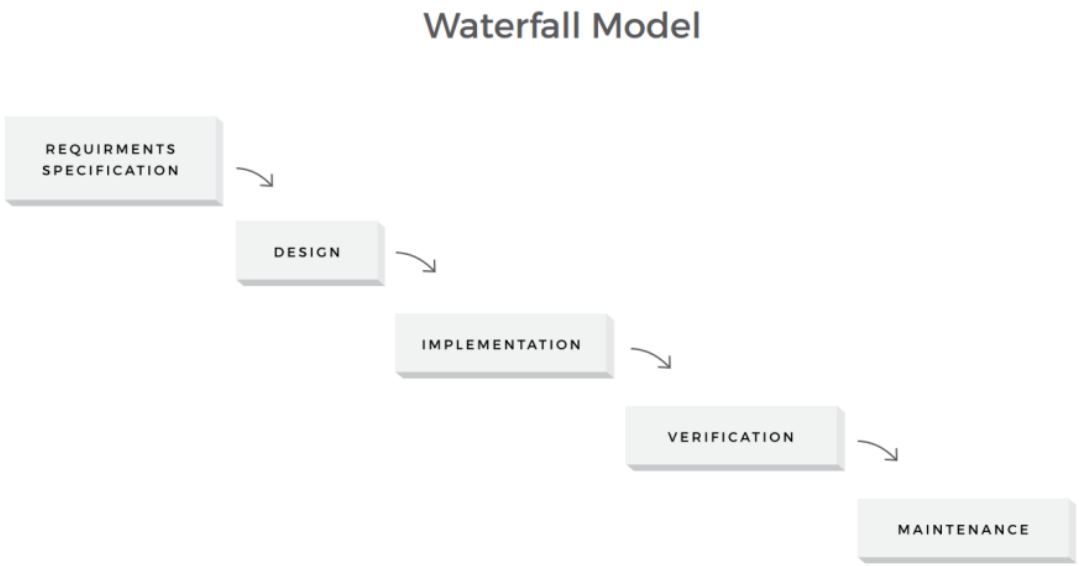
In attempts to find a universal approach to managing any project, humanity has developed a significant number of PM techniques and methodologies.

Traditional Project Management Methodologies

Based on the above-described classic framework, traditional methodologies take a step-by-step approach to the project execution. Thus, the project goes through the initiation, planning, execution, monitoring straight to its closure in consecutive stages.

Often called **linear**, this approach includes a number of internal phases which are sequential and executed in a chronological order. Applied most commonly within the construction or manufacturing industry, where little or no changes are required at every stage, traditional project management has found its application in the software engineering as well.

Known as the **waterfall model**, it has been a dominant software development methodology since the early 1970s, when *formally described by Winston W. Royce: "There are two essential steps common to all computer program developments, regardless of size or complexity. There is first an analysis step, followed second by a coding step … This sort of very simple implementation concept is in fact all that is required if the effort is sufficiently small and if the final product is to be operated by those who built it – as is typically done with computer programs for internal use.*"



Waterfall model has a strong emphasis on planning and specifications development, which takes up to **40 percent of the project time and budget**. Another basic principle

of this approach is the strict order of the project phases. A new project stage does not begin until the previous one is finished.

The method works well for clearly defined projects with a single deliverable and fixed deadline. The Waterfall approach requires thorough planning, extensive project documentation and tight control over the development process. In theory, this should lead to on-time, on-budget delivery, low project risks, and predictable final results.

However, when applied to the actual software engineering process, Waterfall method tends to be slow, costly and inflexible due to numerous restrictions. In many cases, its inability to adjust the product to the evolving market requirements often results in a huge waste of resources and the eventual project failure.

Agile Project Management Philosophy

As opposed to a traditional approach, **Agile project management philosophy** has been introduced as an attempt to make software engineering more flexible and efficient. With 94 percent of the organizations practicing agile in 2016, it has become the industry standard for project management.

The history of agile can be traced back to **1957**: at that time Bernie Dimsdale, John von Neumann, Herb Jacobs, and Gerald Weinberg were using incremental development techniques (which are now known as Agile), building software for IBM and Motorola. Although, not knowing how to classify the approach they were practicing, they realized clearly that it was different from Waterfall in many ways.

However, the modern-day agile approach was officially introduced in **2001**, when a group of 17 software development professionals met to discuss alternative project management methodologies. Having a clear vision of the flexible, lightweight and team-oriented software development approach, they mapped it out in the Manifesto for Agile Software Development.

Aimed at “uncovering better ways of developing software”, the Manifesto clearly specifies the fundamental principles of the new approach:

“*Through this work we have come to value:*

Individuals and interactions *over processes and tools*

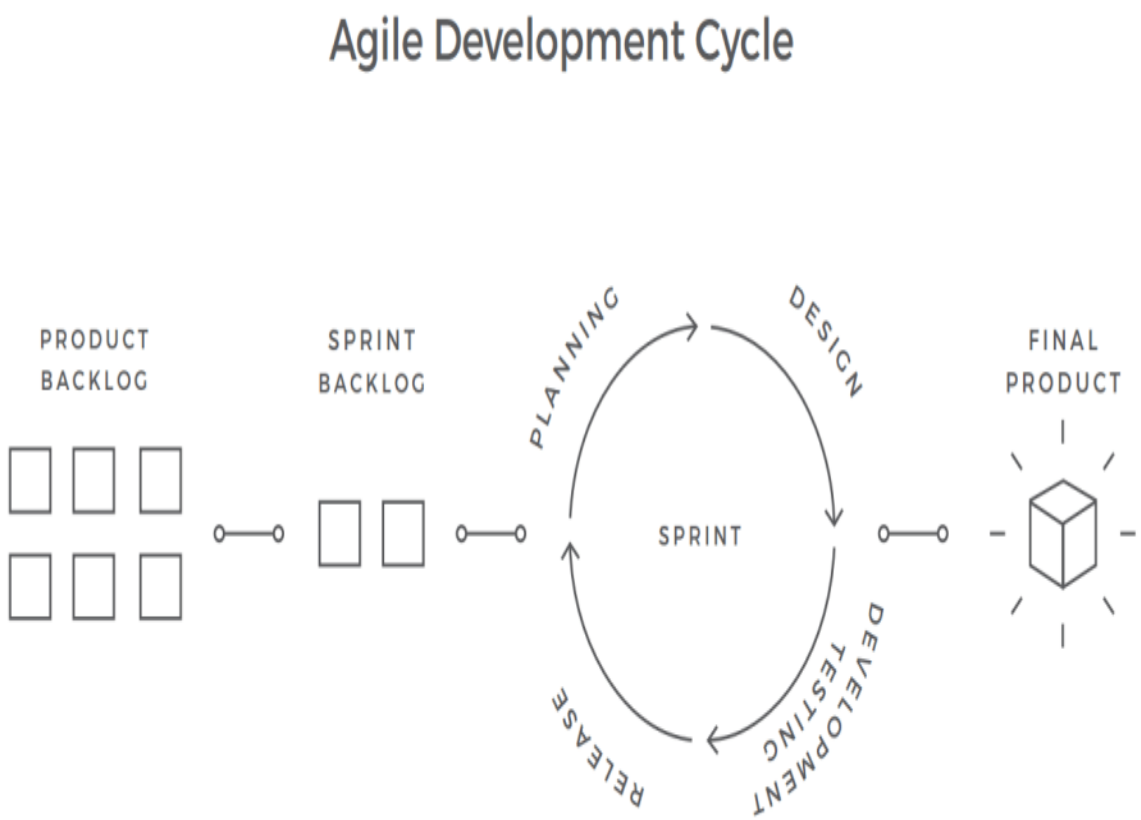
Working software *over comprehensive documentation*

Customer collaboration *over contract negotiation*

Responding to change over following a plan."

Complemented with the Twelve Principles of Agile Software, the philosophy has come to be a universal and efficient new way of managing projects.

Agile methodologies take an iterative approach to software development. Unlike a straightforward linear waterfall model, agile projects consist of a number of smaller cycles – sprints. Each one of them is a project in miniature: it has a backlog and consists of design, implementation, testing and deployment stages within the pre-defined scope of work.



At the end of each sprint, a **potentially shippable product increment is delivered**. Thus, with every iteration new features are added to the product, which results in the gradual project growth. With the features being validated early in the development, the chances of delivering a potentially failed product are significantly lower. Let's summarize the main Agile aspects:

Flexibility: The scope of work may change according to new requirements.

Work breakdown: The project consists of small cycles (known as Sprints in Scrum).

Value of teamwork: The team members work closely together and have a clear vision about their responsibilities.

Iterative improvements: There is frequent reassessment of the work done within a cycle to make the final product better.

Cooperation with a client: A customer is closely engaged in the development and can change the requirements or accept the team's suggestions.

Prioritizing flexibility and rapid turnaround, the Agile approach offers the following benefits, according to the recent research:

- Ability to manage the changing priorities (88%)
- Increased team productivity through daily task allocation (83%)
- Better project visibility due to the simple planning system (83%)

Agile Frameworks

Agile is an umbrella term for a vast variety of methodologies and techniques, sharing the principles and values described above. Each of them has its own areas of use and distinctive features. The most popular frameworks are Scrum, Kanban, Hybrid, Lean, Bimodal, and XP. Before discussing these frameworks in more detail, let's look at their key features.

Framework	Planned Mitigation
Scrum	• The entire scope of work is broken down into short development cycles — Sprints. • The Sprint's duration is from one to four weeks. • The team should strictly follow a work plan for each Sprint. • People involved in a project have predefined roles.
Kanban	• Development is built on workflow visualization. • The current work (work in progress or WIP) is prioritized. • There are no timeboxed development cycles. • The team can change the work plan at any time.
Hybrid	• Agile and Waterfall complement each other. • Agile software development is held under Waterfall conditions (fixed deadline, forecasted budget, and thorough risk assessment).
Bimodal	• There are two separate modes of work — traditional (Mode 1) and Agile (Mode 2). • Two separate teams are working on projects with two different goals. • The Mode 1 team maintains IT system infrastructure. • The Mode 2 team delivers innovative applications. • Cross-team collaboration is important
Lean	• The framework promotes fast software development with less effort, time, and cost. • The development cycle is as short as possible. • The product delivered early is being continuously improved. • The team is independent and has a wider range of responsibilities than those in Scrum, Bimodal, and Hybrid. • Developers can also formulate the product's concept
XP	• The focus is on technical aspects of software development. • XP introduces engineering practices aimed at helping developers write a clear code. • Product development includes consistent stages: core writing, testing, analyzing, designing, and continuous integration of code. • Face-to-face communication within the team and customer involvement in development are crucial

Scrum:Roles, Sprints, and Artifacts

Scrum is a dominant agile methodology. It's used exclusively by 58 percent of organizations while another 18 percent of the companies combine it with other techniques. First described in 1986 by Hirotaka Takeuchi and Ikujiro Nonaka in the New Product Development Game,

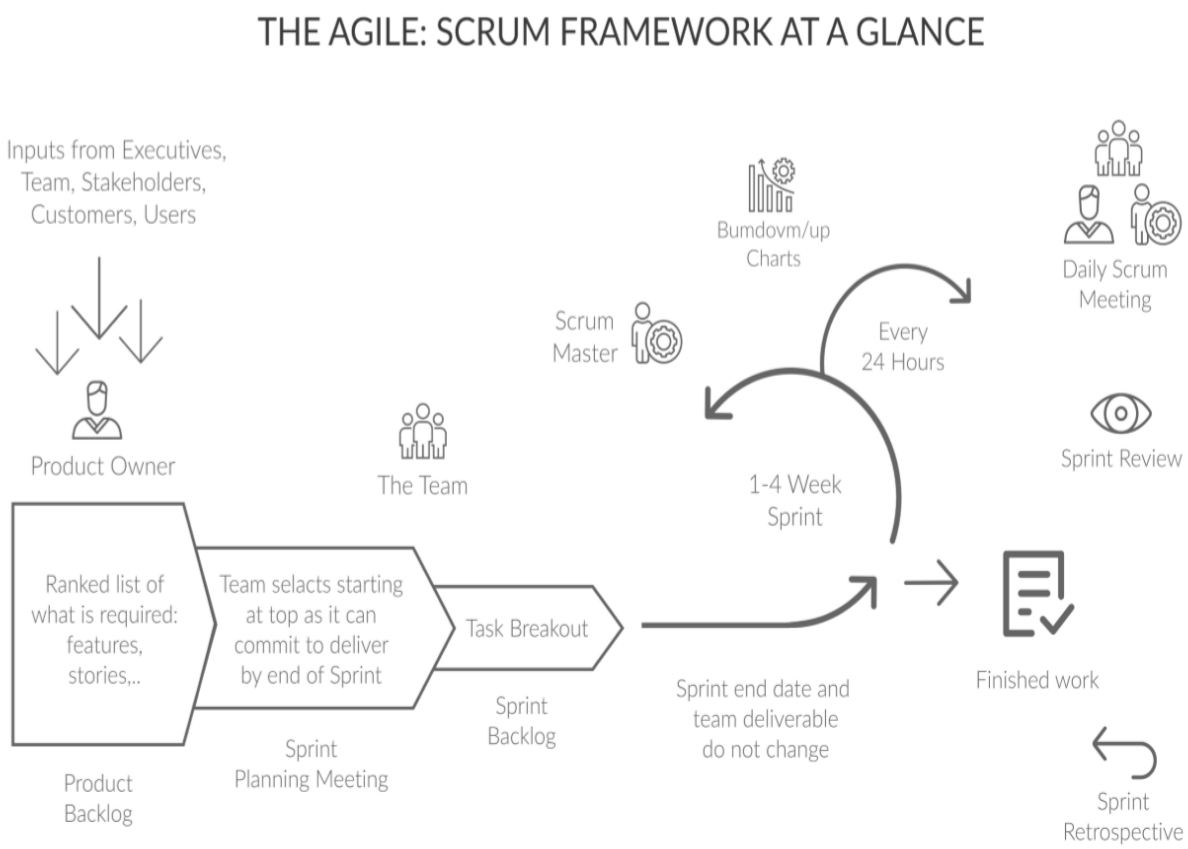
it was formulated almost a decade after. In 1995, Ken Schwaber and Jeff Sutherland, the authors of The Scrum Guide, presented it at the OOPSLA conference. The presentation was based on the knowledge they acquired as they applied the method during the previous few years. While Scrum was introduced far before the Agile Manifesto, it relies on Agile principles and is consistent with the values stated in that document.

Scrum is aimed at sustaining strong collaboration between people working on complex products, and details are being changed or added. It is based upon the systematic interactions between the three major roles: Scrum Master, Product Owner, and the Team.

- **Scrum Master** is a central figure within a project. His principal responsibility is to eliminate all the obstacles that might prevent the team from working efficiently.
- **Product Owner**, usually a customer or other stakeholder, is actively involved throughout the project, conveying the global vision of the product and providing timely feedback on the job done after every sprint.
- **Scrum Team** is a cross-functional and self-organizing group of people that is responsible for the product implementation. It should consist of up to 7 team members, in order to stay flexible and productive.

Sprints and Artifacts

A basic unit of work in scrum – **sprint** – is a short development cycle that is needed to produce a shippable product increment. A sprint usually is between 1 and 4 weeks long: More lengthy iterations lack the predictability and flexibility that are scrum's fundamental benefits. Having no standard duration (as long as it is less than 4 weeks), all the sprints within a project should have a fixed length. This makes it easier to plan and track progress.



Scrum relies on **three main artifacts** which are used to manage the requirements and track progress – Product backlog, Sprint backlog, Sprint burndown chart. The process is formalized through a number of recurring meetings, like Daily Scrum (Standup), Sprint Planning, Review and Retrospective meetings.

The Product Backlog is an ordered list of feature items that might be needed in the project's final product. It is a single source of requirements. The product Backlog updates as new requirements, fixes, features, and details are being changed or added.

The Sprint Backlog is a list of tasks the team must complete to deliver an increment of functional software at the end of each Sprint. In other words, team members agree on which product items to deliver and define a plan on how to do so.

The Sprint Burndown Chart is an illustration of the work remaining in a Sprint. It helps both the team and the Scrum Master as it shows progress on a day-to-day basis and can predict whether the Sprint goal will be achieved on schedule.

Scrum Meetings

The process is formalized through a number of recurring meetings, like the Daily Scrum (Standup), the Sprint Planning, the Review, and Retrospective meetings (the Sprint Retrospective).

The **Daily Scrum** is a timeboxed meeting, during which a Development Team coordinates its work and sets a plan for the next 24 hours. The event lasts 15 minutes and should be held daily at the same place and time.

The work to be completed is planned at the **Sprint Planning**. Everyone involved in the Sprint (a Product Owner, a Scrum Master, and a Development Team) participates in this event. They answer two key questions: which work can be done and how this work will be done. The Sprint Planning lasts no longer than eight hours for a one-month Sprint. For shorter Sprints, the meeting usually takes less time.

At the end of each Sprint, the team and the product owner meet at the **Sprint Review**. During this informal meeting, the team shows the work completed and answers questions about the product increment. All participants collaborate on what to do next to increase the product's value. The Sprint Review is a four-hour timeboxed meeting for one-month Sprints.

The whole team goes to **Retrospective Meetings** to reflect on their work during the Sprint. Participants discuss what went well or wrong, find ways to improve, and plan how to implement these positive changes. The Sprint Retrospective is held after the Review and before the next Sprint Planning. The event's duration is three hours for one-month Sprints.

When to Use Scrum

Scrum works well for long-term, complex projects that require stakeholder feedback, which may greatly affect project requirements. So, when the exact amount of work can't be estimated, and the release date is not fixed, Scrum may be the best choice.

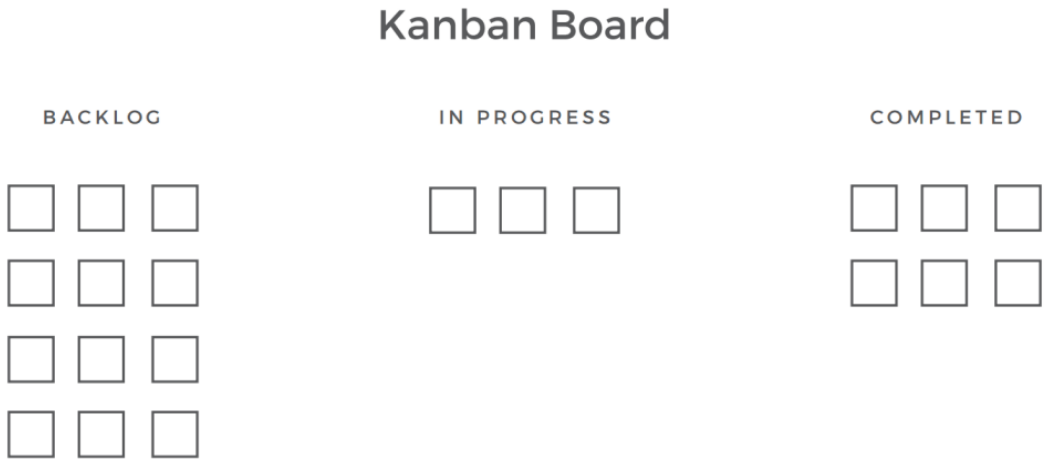
By setting customer needs and on-time/on-budget delivery as the highest priority, Scrum has gained the trust of 89 percent of Agile users. Thus, the list of companies using this approach is impressive. In fact, there is a public spreadsheet with such organizations, including Microsoft, IBM, Yahoo, and Google.

The latest research by the Scrum Alliance suggests that Scrum goes beyond IT. Companies working in the fields of finance, consulting, education, retail, media, and entertainment choose this approach to organize their work processes and enhance cooperation with customers. In 2016, the majority of State of Scrum Report respondents (98 percent) said they are going to use this framework to move forward.

Kanban: Comprehensive Solution to Handling Work in Progress

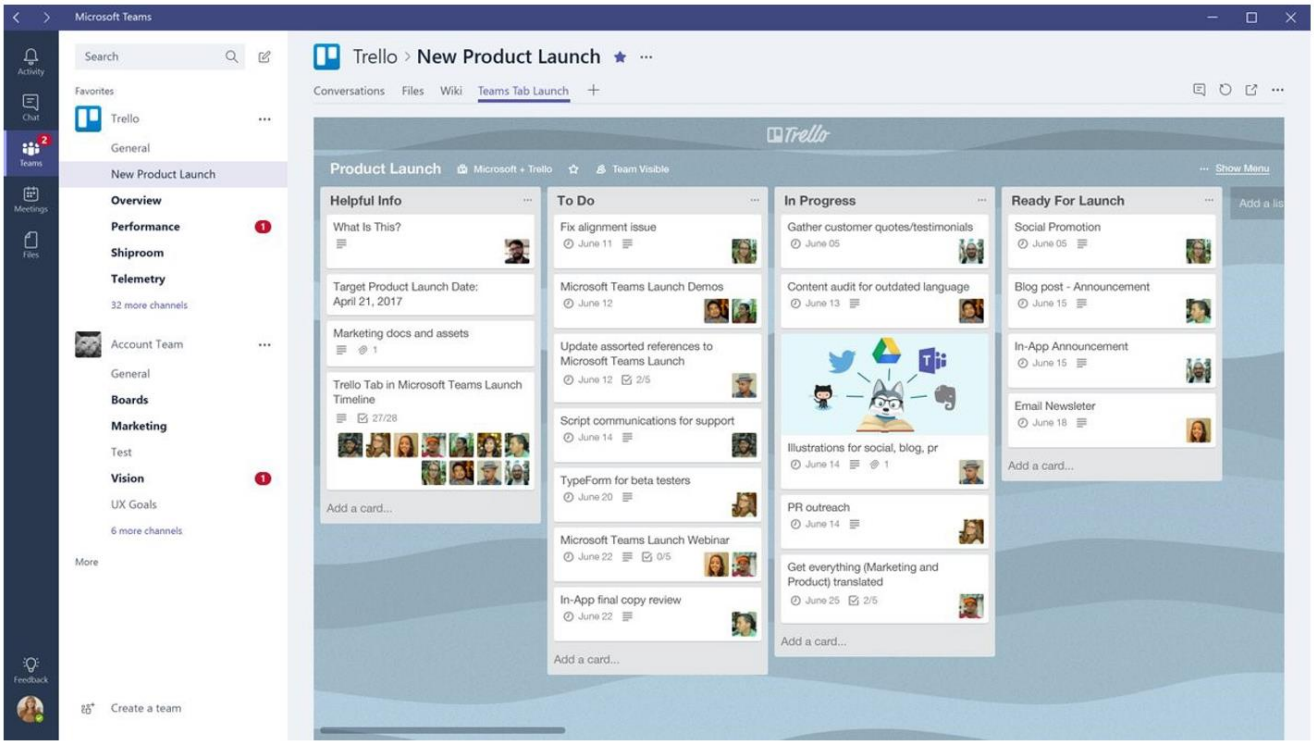
Another common project management framework is Kanban. Forty three percent of companies have stated that they use Kanban as one of the project management

frameworks. Originating from a visual system of cards used in Toyota manufacturing as a production control method, Kanban is simple, yet powerful, approach to developing software products.



Translated as *visual signal* from Japanese, Kanban focuses on the visualization of the workflow and prioritizes the **work in progress (WIP)**, limiting its scope to match it effectively to the team's capacity. As soon as a task is completed, the team can take the next item from the pipeline. Thus, the development process offers more flexibility in planning, faster turnaround, clear objectives, and transparency.

No standard procedures within the process, as well as the fixed iterations, are required in Kanban, as opposed to Scrum. The project development is based on the workflow visualization through a **Kanban board**, usually represented by sticky notes and whiteboards or online tools like Trello.



Trello automates and digitalizes Kanban. Due to the succinct information about a work item each Kanban card contains, everyone in the team knows who is responsible for the item, what each person's task is, when it's supposed to be finished, etc. Team members can also leave comments, attach screenshots, documents, or links to provide more details.

Teams using Kanban tools work in a cooperative manner. The ability to track progress helps coworkers understand everyone's personal input in achieving the common goal, resulting in a focus on completing the task well and on time.

When to Use Kanban

Using Kanban, teams can do small releases and adapt to changing priorities. Unlike Scrum, there are no sprints with their predefined goals. Kanban is focused on doing small pieces of work as they come up. For example, if testers find errors in the product, developers try to fix them right away. Kanban, for instance, works well after the main release of the product.

Companies like Spotify and Wooga (leading mobile games development company) have been using this approach successfully over the years. Yet, 8 percent of organizations combine Scrum with Kanban techniques, using so-called **Scrumban** rather than the original frameworks.

Hybrid: Blend of Waterfall and Agile (Flexible Development and Thorough Project Planning)

Agile and Waterfall are two different visions of software development management. The former is about iterative development and being flexible, while the latter, promoting step-by-step development, requires careful planning, and rejects making changes along the way.

Twenty-three percent of companies realized that using principles of both approaches can be more beneficial than choosing one of the two. The combination of the traditional Waterfall project management approach and Agile is called Hybrid.

Specialists use advantages of the Agile philosophy for software development. When it comes to budgeting, planning, and hardware set up, Waterfall works well. On the other hand, by embedding Agile practices into a traditional Waterfall work process, companies can increase chances of realizing successful projects. For example, project planning can be done in sprints, testing can be incorporated in development, and feedback can be gathered regularly. Other ways of modifying the Waterfall model include using Kanban boards and organizing retrospectives.

It should be noted that the choice of hybrid framework's features may depend on the project. The hybrid frameworks not only imply using both approaches, depending on the project phase, but also include options to inject Agile practices into a Waterfall process.

When to use Hybrid

Hybrid is an effective solution when product delivery relies on both hardware and software operations. But, there is another reason to choose Hybrid. The situation in which a customer is not satisfied with an unspecified timeframe and budget, as well as the lack of planning, is not rare. Such uncertainty is typical for Agile. In this case, planning, requirements specification, and an application design can be accomplished in Waterfall. Agile is in place for software development and testing.

Bimodal: traditional Waterfall combined with Agile

The Bimodal approach is quite popular: It is estimated that 16 percent of companies choose it. The term "Bimodal IT" was introduced by Gartner in 2014. Bimodal is the

practice of managing two separate but consistent styles of work: one focused on predictability and the other on agility.

Mode 1 is traditional; thus, it works perfectly in well-understood and predictable areas. According to Gartner, it focuses on exploiting what is known while transforming the legacy environment into a state fit for a digital world.

Mode 2 involves rapid application development. It is exploratory, nonlinear, and optimized for solving new problems. Mode 2 is especially useful for working on projects that need to be finished as quickly as possible.

Both modes require different skills, techniques, and tools. Therefore, two separate work groups are needed. These teams have two distinct goals — ensuring stability while adopting innovations. Team members focus on projects that suit their mode best.

The Mode 1 team develops and maintains applications and core systems to support long-term business needs. A company's technological capabilities depend directly on the work that's done by this team.

The Mode 2 team frequently delivers innovative applications to engage new customers and meet short-term business needs. This team may change the product's functionality after having received feedback and analyzed the market.

The teams use different delivery mechanisms and report through different organizational structures. Nevertheless, they need to communicate with each other to exchange ideas and share results.

As Sandy Kemsley specifies, Mode 2 relies on the information and services infrastructure provided by Mode 1, while Mode 1 relies on Mode 2 for testing both new product ideas and new development methods that may eventually be rolled back into Mode 1.

When to use Bimodal

If the company specializes in both long- and short-term projects that require different development and management approaches, Bimodal might be the right choice. This framework is about keeping the balance between maintaining IT system infrastructure and driving innovations. When successfully implemented, Bimodal helps organizations quickly deliver solutions that users need to stay competitive.

Lean: Eliminating Waste in Software Engineering

According to the latest estimates, 17 percent of organizations adopt **Lean**. Its popularity decreased from 2015 to 2016. Nevertheless, this framework remains one of the 5 most widely used Agile frameworks. Having the same origins as Kanban, the approach started as a technique applied to physical manufacturing. It stemmed from Toyota Production System as a management approach aimed at “making the vehicles ordered by customers in the quickest and most efficient way, in order to deliver the vehicles as quickly as possible.”

The application of Lean principles to software development was initially introduced by Mary and Tom Poppendieck in their book Lean Software Development: An Agile Toolkit. It includes the 7 basic principles:

- Eliminate waste
- Amplify learning and create knowledge
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity/quality in
- See the whole

Now let’s have a closer look at these principles.

Eliminating waste. In terms of a project, a term “waste” refers to anything that is not adding the value to the project and thus should be eliminated. In software engineering, this can be idle time, unnecessary features, or defects.

Amplify learning and create knowledge. In Lean, software development is perceived as an ongoing learning process. Developers don’t usually write clear code on the first try. After having detected and fixed errors, they write an improved variation of the previous code. Engineers gain knowledge during development by solving problems and producing code variations. So, the best way to improve the software development environment is to amplify learning.

Decide as late as possible. Late decisions are more informed ones because they are based on facts. Keeping in mind that technologies become obsolete increasingly faster, delaying an irreversible design decision is a wise move. A major strategy for making commitments late is to reserve the capacity for the change in the system.

Deliver as fast as possible. The fourth principle is about the pros of fast software development. Short development cycles allow developers to learn more by getting feedback. They also allow a customer to delay making a final decision about design until they know more. So, fast delivery helps eliminate waste.

Empower the team. Developers should have the right to make technical decisions as they understand the details of their work like no one else. They can create a roadmap and follow it.

Build in integrity/quality. The user's perception of the software and its characteristics must coincide. If a customer thinks that software has all the needed features and is easy to use, that system has a perceived integrity. Conceptual integrity means that the software has a coherent architecture, and scores high on usability and fitness of purpose. It can be maintained, adapted, and extended.

See the whole. Engineers should take charge of the overall efficiency of the system, instead of focusing on their small portion. If experts adhere to this principle, they can create a system with integrity.

These fundamentals perfectly describe Lean philosophy: its aim is to deliver more value through less effort, investment and time.

Lean software development is an **iterative and incremental framework**. Therefore, as in any other Agile approach, the working product increment is delivered at the early stages of development. The further progress depends largely on the product owner's feedback.

What differentiates Lean approach is that the team is not restricted to use any formal processes, such as recurring meetings or thorough task prioritization.

When to Use Lean

Lean allows companies to follow a minimum viable product (MVP) development technique. It includes a deployment of a product with a minimum, sufficient set of features to satisfy early users. The idea of the MVP strategy is to gather and analyze customer feedback to know if they like this product and want to buy it. Knowledge of a customers' habits, tastes, and needs is the key to producing commercially successful products. Developers use feedback to create a roadmap for future development.

Lean works well for small, short-term projects due to their short life cycles. This approach is also appropriate if the customer can participate in a project realization as Lean requires ongoing feedback. Another important condition to the adoption of Lean is the whole team should work in one office to enable communication.

Being effectively adopted by a vast number of manufacturing companies, like Nike, Ford and Intel, Lean principles are widely used in other industries. Startups and successful companies, e.g. Corbis, PatientKeeper, and Xerox, apply Lean software engineering practices to their processes.

Extreme Programming: Engineering Practices For Writing A Good Code

Extreme Programming (XP) differs from the above-mentioned frameworks by its focus on technical aspects of software development. XP is used at 9 percent of companies.

It combines the most essential, providing agile teams with a number of tools to optimize the engineering process. Extreme Programming is a set of certain practices, applied to software engineering in order to improve its quality and ability to adapt to the changing requirements.

XP requires developers to perform a little number of engineering practices on the highest, almost extreme level possible, hence the name.

XP was introduced in the 1990s. Kent Beck, one of the initial signatories of the Agile Manifesto, invented it while working on a Chrysler Comprehensive Compensation System project. He aimed at finding ways of doing sophisticated tasks as expeditiously as possible. In 1999, he documented XP practices in the book Extreme Programming Explained: Embrace Change. The most commonly used **XP practices** are:

- Test-Driven Development (TDD)
- Refactoring
- Continuous Integration
- Pair Programming

Test-Driven Development is an advanced engineering technique that uses automated unit tests to propel software design process. As opposed to the regular development cycle, where the tests are written after the code (or not written at all), TDD has a test-first approach. This means that the unit tests are written prior to the code itself.

According to this approach, the test should fail first when there is no code to accomplish the function. After that the engineers write the code, focusing on the functionality to make the test pass. As soon as it's done, the source code should be improved to pass all the tests. These three steps are often referred to as the RedGreen-Refactor cycle.

TDD has proven to provide the following **benefits**:

1. The tests are used to capture any defects or mistakes in the code, providing constant feedback on the state of every software component. Thus, the quality of the final product is increasingly high.
2. The unit tests can be used as an always up-to-date project documentation, changing as the project evolves.

3. Being deeply involved in the product development, the team needs to be able to critically analyze it and foresee the planned outcome in order to test it properly. This keeps the team motivated and engaged, contributing to the product quality.
4. With a thorough initial testing, the debugging time is minimized.

Apart from being used within the TDD cycle, **code refactoring** is a common practice in agile software development. Basically, it's a process of a constant code improvement through simplification and clarification. The process is solely technical and does not call for any changes in software behavior.

Extending the source code with each iteration, agile teams use refactoring as a way to weed out code clutter and duplications. This helps prevent software rot, keeping the code easy to maintain and extend.

Continuous Integration (CI) is another practice agile teams rely on for managing shared code and software testing. We believe CI is an evolutionary development of the Agile principles. Instead of doing short iterations, developers can commit newly written parts of a code several times a day. This way, they constantly deliver value to users.

To verify the quality of the software — through testing — and automate its deployment, teams usually use Tools like CruiseControl, Atlassian Bamboo, TeamCity or Jenkins.

In addition, CI helps maintain the shared code, eliminating the integration issues. Thus, the product's mainline is robust and clean and can be rapidly deployed.

Pair Programming, or "pairing", is considered to be a very controversial agile practice. This technique requires two engineers working together. While one of them is actually writing the code, the other one is actively involved as a watcher, making suggestions and navigating the through process.

Being focused on both code and more abstract technical tasks, this team of two is expected to be more efficient, creating better software design and making fewer mistakes. Another benefit of this approach lies in spreading the project knowledge across team members.

However, this practice has often been accused of having a negative impact on the team's short-term productivity. The research shows that each task usually requires 15-60 percent more time, which is a major drawback of the approach. Yet, there are some opinions that the extra time is easily compensated in the long term through the overall higher quality of the software.