1. Agile methodologies

Agile methodologies are approaches to product development that are aligned with the values ​​and principles described in the [Agile Manifesto](https://agilemanifesto.org/) for software development. Agile methodologies aim to deliver the right product, with incremental and frequent delivery of small chunks of functionality, through small cross-functional self-organizing teams, enabling frequent customer feedback and course correction as needed.

Application of Agile Methodology

Through most of its brief [history](https://en.wikipedia.org/wiki/Agile_software_development#History) (since 1999-2000), “Agile” has been predominantly an approach to software development and IT application development projects.  Since then, however, it now extends to other fields, too, especially in the knowledge and services industries.

Agile is about being responsive to the market and to the customer by responding quickly to their needs and demands and being able to change direction as the situation demands.  Be it IT or software development or any other field where there is a flow of work and delivery of work products, Agile methods are applicable.  Agile methods attempt to maximize the delivery of value to the customer and minimize the risk of building products that do not – or no longer – meet market or customer needs.

They do this by breaking up the traditionally long delivery cycle (typical of the legacy “waterfall methods”) into shorter periods, called sprints or iterations. The iteration provides the cadence for delivering a working product to the customer, getting feedback and making changes based on the feedback.

Thus, Agile methods have sought to reduce delivery times (delivering early, delivering often) to ensure that smaller vertical chunks of the product get to the market, enabling customers to provide feedback early and ensure that the product they finally get meets their needs.

Agile has become an umbrella term for a variety of planning, management and technical methods and processes for managing projects, developing software and other products and services in an iterative manner. These methods include Scrum, by far the most prevalent and popular method for software, XP (eXtreme Programming or Paired Programming), and more lately Kanban.

Agile methods also include technical practices – most of which fall under the umbrella term DevOps – that enable Test Automation, Continuous Integration/ Continuous Delivery/ Deployment (CI/ CD) and overall, an ever-shrinking delivery cycle for software and other products and services.

The use of Agile as an approach to project management has increased dramatically in recent years. [Gartner predicts](https://www.gartner.com/webinar/3169117) that agile development methods will soon be used in 80% of all software development projects.

The [Agile Manifesto](http://agilemanifesto.org/principles.html) is a statement of core values ​​and principles for software development. The Agile Manifesto for software development was set up in 2001 and it is a declaration of 4 vital rules and 12principles that serve as a guide for people in [agile software development](https://www.digite.com/swiftenterprise/). It was created by 17 professionals who already practiced agile methods such as XP, DSDM, SCRUM, FDD, etc, gathered in the snowy mountains of the US state of Utah, convened by [Kent Beck](https://www.kentbeck.com/)[.](https://www.digite.com/swiftenterprise/)

#### Core values of Agile Manifesto

Individuals and interactions over processes and tools – The first value emphasizes teamwork and communication. We must understand that software development is a human activity and that the quality of interaction between people is vital. Tools are an important part of software development, but making great software depends much more on teamwork, regardless of the tools team may use.

Working software over comprehensive documentation – Documentation has its place and can be a great resource or reference for users and coworkers alike. The main goal of software development, however, is to develop software that offers business benefits rather than extensive documentation.

Customer collaboration over contract negotiation – Development teams must work closely and communicate with their customers frequently. By listening to and getting feedback, teams will understand what all stakeholders really want.

Responding to change over following a plan – Changes are a reality in Software development, a reality that your Software process should reflect. A project plan must be flexible enough to change, as the situation demands.

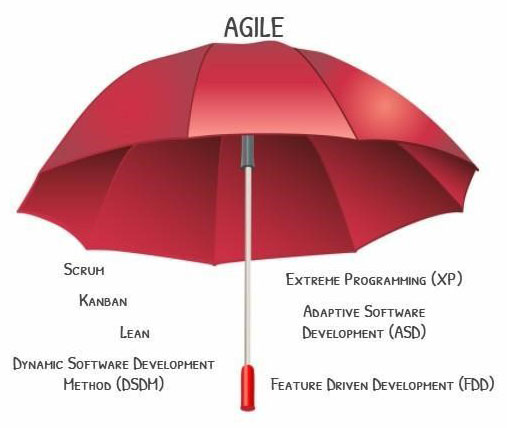
DevOps is the direct descendant of [agile software development](https://www.synopsys.com/glossary/what-is-agile-sdlc.html), born from the need to keep up with increased software development velocity and throughput agile methods. Advancements in agile development highlighted the need for a more holistic approach to the software delivery life cycle, resulting in DevOps.

“Agile development” is an umbrella term for several iterative software development methodologies, many of which have carried over to DevOps:

#### Key Agile Methodologies

Agile is an umbrella term for several methods and practices. Let’s look at some of the popular methodologies:

* Scrum
* Extreme Programming (XP)
* Adaptive Software Development (ASD)
* Dynamic Software Development Method (DSDM)
* Feature Driven Development (FDD)
* Kanban
* Behavior Driven Development (BDD)

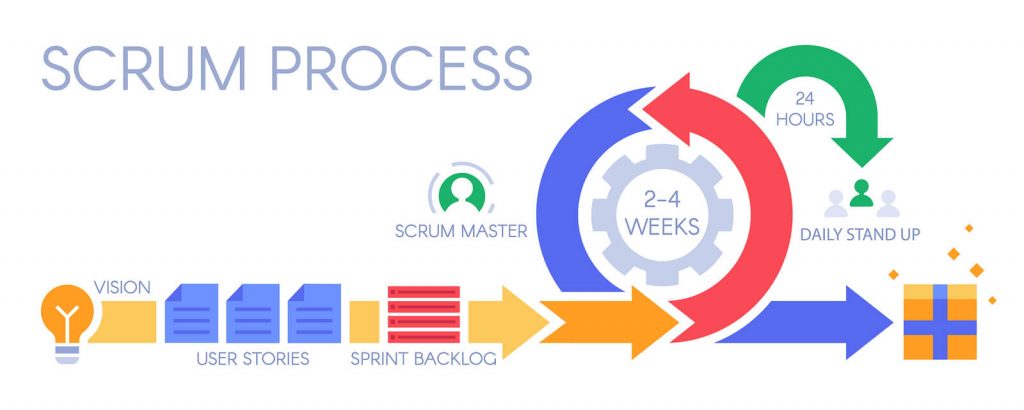


Scrum Methodology

Scrum methodology is a simple [framework](https://www.scrum.org/resources/what-is-scrum) for working with complex projects, and it was created by [Ken Schwaber](https://www.scrum.org/team/ken-schwaber) and [Jeff Sutherland](https://www.scrumalliance.org/community/profile/jsutherland).

Agile software development methodologies are iterative, meaning the work is divided into iterations, which are called Sprints in the case of Scrum. Scrum is executed by small teams of between 7-9 people, including a Scrum Master and a Product Owner.

In Scrum, projects are divided into cycles (typically 2 or 3 week cycles) called Sprints. The Sprint represents a timebox within which a set of features must be developed.  Multiple sprints might be combined to form a Release – where formal software/ product delivery is made to the customer/ market.



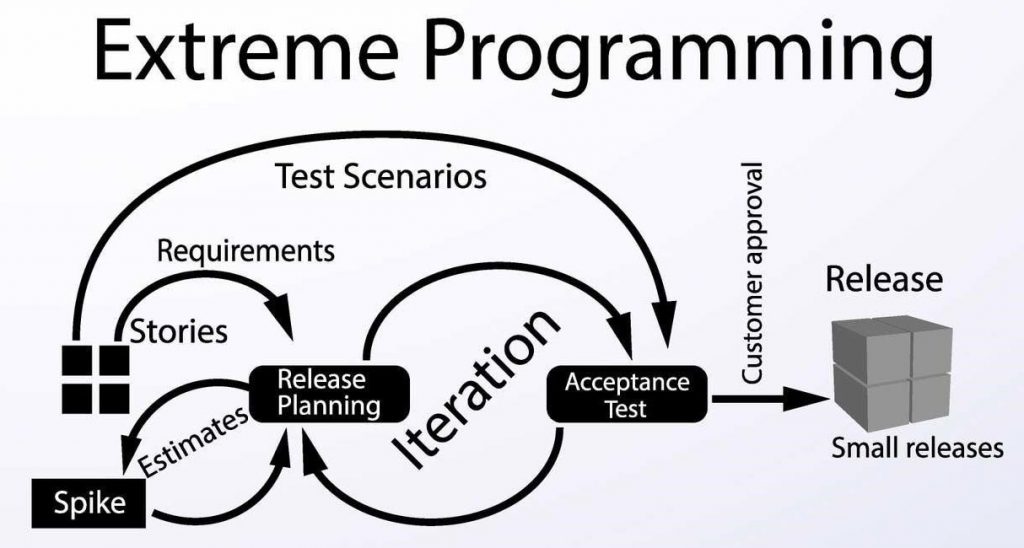
The overall product functionality is broken down by the Product Owner into smaller features (typically described as Epics and User Stories – or just Stories).   These Stories are prioritized and taken up in each Sprint or Iteration.  The intent of the method is for the team to be able to demo at the end of each Sprint working pieces of the product to the Product Owner, to make sure that the product is working as intended.

Overall, the Scrum method breaks the long waterfall process delivery into smaller cycles, which enables product teams and the end-customer to frequently review working software and ensure that it meets their business requirements. This ensures that the end product also meets the final requirements of the customer.

The Scrum method is characterized by specific ceremonies such as the Daily Standup meeting, the Sprint Review Meeting, the Demo to the Product Owner and the Sprint Retrospective meeting.  All of these meetings provide collaboration and review opportunities to the team to ensure that development is progressing as intended, and any issues are resolved quickly.

Extreme Programming (XP)

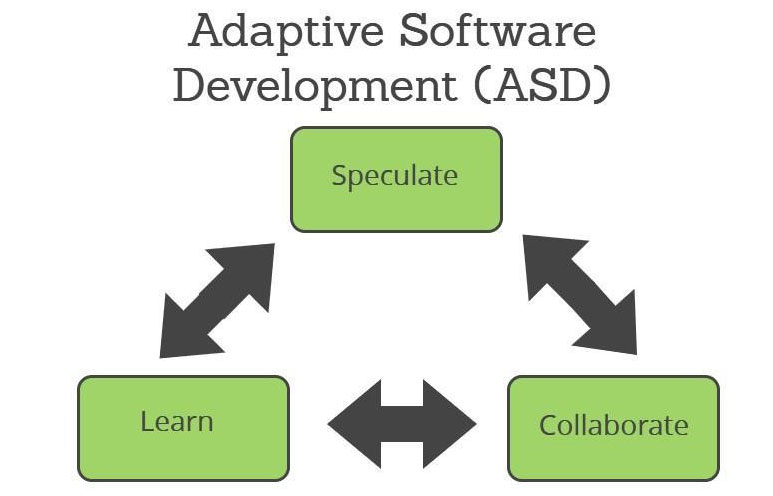
[Extreme Programming](https://en.wikipedia.org/wiki/Extreme_programming) (XP) – or Paired Programming is a methodology developed by [Kent Beck](https://www.kentbeck.com/) in the early 90s. This agile methodology focuses on enhancing interpersonal relationships as a key to success in [software development](https://www.fatbit.com/fab/top-10-custom-software-development-companies/). XP also focuses on promoting teamwork, caring for the learning of developers, and fostering a good working environment. It is characterized by developers working in pairs where one developer programs while the other developer observes; and they switch these roles on a regular basis throughout the Sprint. This way, they enable continuous code review and feedback that enhances code quality and developer capability.



Extreme Programming (XP) promotes continuous feedback between the client and the development teams, fluid communication between all participants, simplicity in the implemented solutions and the readiness to face changes. XP is especially suitable for projects with indistinct and highly changing requirements, and where there is high technical risk.

Adaptive Software Development (ASD)

Adaptive Software Development ([ASD](https://en.wikipedia.org/wiki/Adaptive_software_development)) was developed by [Jim Highsmith](https://www.goodreads.com/author/show/309733.Jim_Highsmith) and Sam Bayer in the early 1990s. It incorporates the principles of continuous adaptation, i.e., *adapt to change and not fight against it*. Adaptive Software Development uses a dynamic development cycle known as Speculate, Collaborate, and Learn. This cycle is dedicated to constant learning and intense collaboration between developers and customers due to the constant change in the business environment.

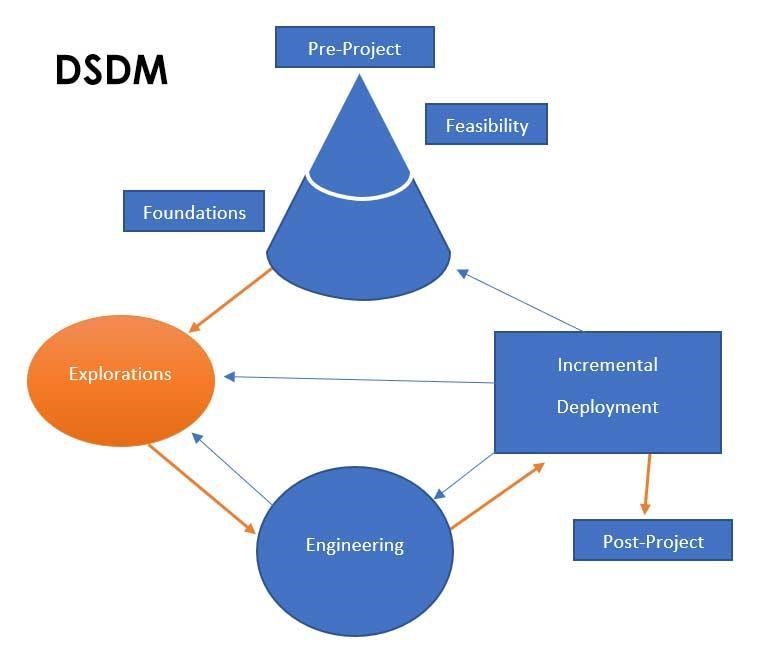


Unlike most Software development methodologies which use a static life cycle i.e., Plan-Design-Build, ASD offers a non-linear iterative life cycle, where each cycle can iterate and be modified while another cycle is being executed.  It points towards Rapid Application Development ([RAD](https://blog.capterra.com/what-is-rapid-application-development/)), which emphasizes development speed to create a high quality, low maintenance product involving the user as much as possible. The main characteristics of ASD are:

1. Speculate: This is the initiation phase of the project where it is necessary to establish the main objectives and goals of the project by understanding the limitations (risk areas) with which the project operates.
2. Collaborate: This is the phase where most of the development is centered, maintaining co-ordination between teams that ensures what is learned by one team is communicated to the rest and does not have to be learned again by other teams from scratch.
3. Learn: The last stage ends with a series of collaboration cycles – the job is to capture what has been learned, both positive and negative. This stage is critical for the effectiveness of the project.

Dynamic Software Development Method (DSDM)

Dynamic Software Development Method ([DSDM](https://www.agilebusiness.org/page/whatisdsdm)) was developed in the year 1994 by a group of vendors and experts in the field of Software development. DSDM focuses on Software projects that are characterized by tight budgets and schedules. It focuses on frequent delivery of product cycles, and development is iterative and incremental.



With Dynamic Software Development Method (DSDM), one can design a roadmap of early and continuous deliveries for the project, implementing an incremental solution, adapting from the feedback obtained throughout the process, and checking that the expected benefits are being met.

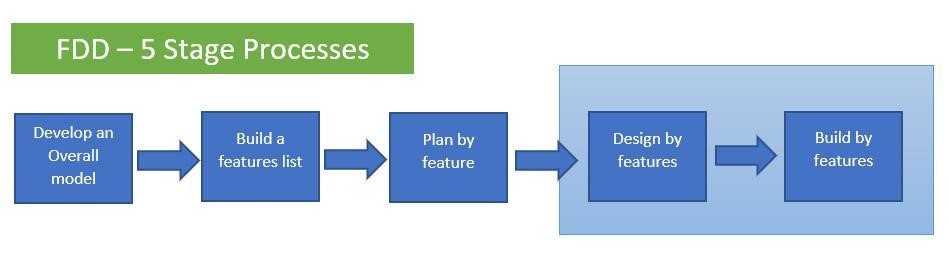
DSDM is an agile model that can undoubtedly help organizations that are used to working on projects to change their mentality and way of working to improve their capacity to deliver value and reduce time to market.

* **Share**
* **Tweet**
* **Share**
* **Share**
* **Pin**
* **Share**

Feature Driven Development (FDD)

Feature Driven Development ([FDD](http://agilemodeling.com/essays/fdd.htm)) methodology is mainly oriented for larger teams with more people than those to whom other agile methodologies such as Scrum are normally applied. FDD was developed by [Jeff De Luca](http://www.jeffdeluca.com/) and Peter Coad in the year 1997. This methodology focuses on short iterations, which allow tangible deliveries of the product in a short period of time (2 weeks).

Projects with multiple teams and a large number of people represent the challenge that not all will be equally talented and disciplined. FDD includes specific activities that help address communication challenges and coordination of such projects.



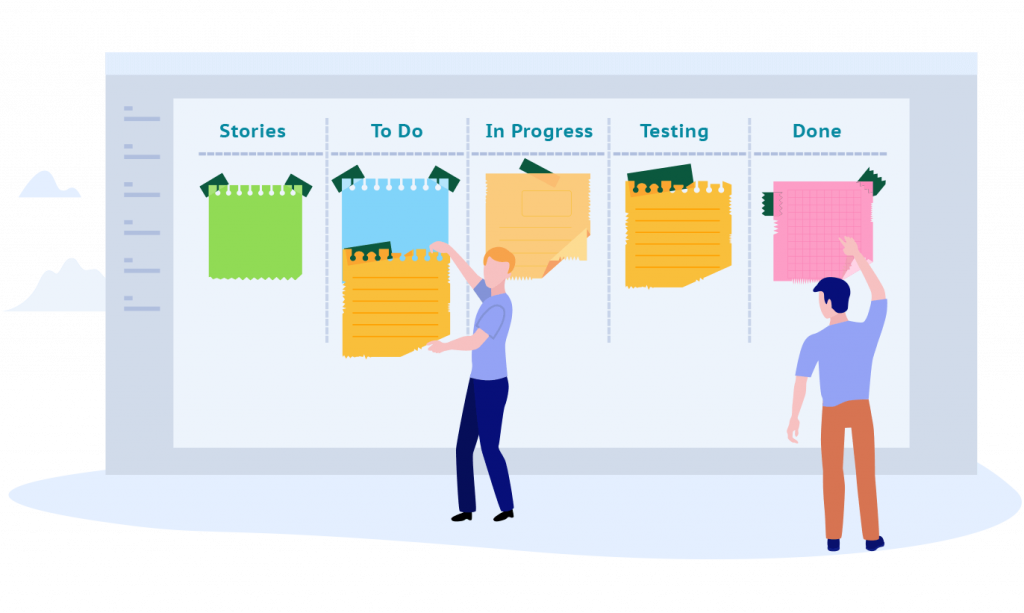
FDD is a 5-stage process, the first 3 of which are sequential and the final two stages are iterative (as shown in the diagram above). All agile methodologies follow a series of principles that make them resemble each other. FDD, however, offers solutions on how to organize the team and how to program the code, which makes it especially viable for large development teams building complex software.

One of the most popular books on the FDD method was published by Stephen Palmer in 2002, titled “[A Practical Guide to Feature-Driven Development](https://books.google.co.in/books/about/A_Practical_Guide_to_Feature_driven_Deve.html?id=NhlFAAAAYAAJ&redir_esc=y)“.

Kanban Method

The Kanban Method was defined by David Anderson in the early -to-mid 2000s, in response to some of the challenges of the various Agile methods, especially Scrum.  These methods, while trying to solve the challenges of traditional/ waterfall methods, became victim to some of the same challenges themselves.

The 2-3 week sprint cycle became too long to wait for many business contexts, the changes required in organizational structure (new roles and responsibilities) and a project management/ planning processes put too much strain on organizations, and many teams found themselves not meeting even sprint-level commitments of scope and quality.  For most organizations, implementing these methods became very disruptive.



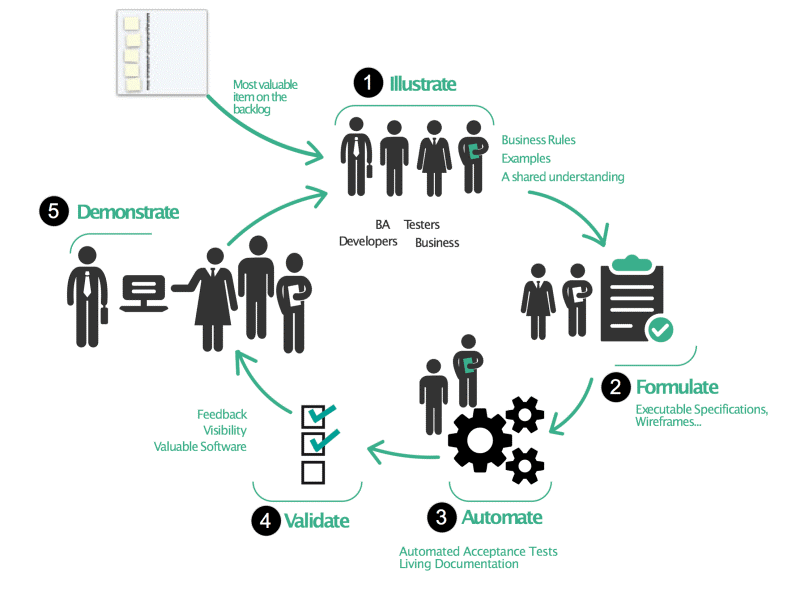
The Kanban Method was defined as the opposite of that – a non-disruptive evolutionary method for improvement, that ultimately enables teams to deliver continuously instead of in time-buckets of 2-3 weeks, get feedback faster and reduce the lead time to deliver value to the customer.

Kanban is a visual system for managing work as it moves through a process. [Kanban](https://www.digite.com/kanban/what-is-kanban/) visualizes both the process (the workflow) and the actual work passing through that process. The goal of Kanban is to identify potential bottlenecks in your process and fix them, so work can flow through it cost-effectively at an optimal speed or throughput.

Kanban is defined as a highly effective and efficient production system. The origin of the Kanban methodology lies in the “just-in-time” (JIT) production processes devised by Toyota, in which cards were used to identify material needs in the production chain.

Behavior Driven Development (BDD)

Behavior Driven Development ([BDD](https://www.agilealliance.org/glossary/bdd/)) is a behavior-oriented agile development methodology. It was created by [Dan North](https://dannorth.net/introducing-bdd/) in 2003 as an evolution of the TDD methodology. Dan North aimed to bring non-technical people together in the process of creating the system’s technical functionality. It happens that when we develop software, we involuntarily fail to include business concepts present in the functionality, resulting in a possible flow for recurring and even serious bugs.



*Source:*[*Johnfergusonsmart.com*](https://johnfergusonsmart.com/behaviour-driven-development-3-minute-rundown/)

BDD uses universal language concepts that encourage collaboration between people with or without technical knowledge in a software project. The BDD development process is based on writing test scenarios and features. These contain the requirements and acceptance criteria for the system behavior. It tells you what the functionality needs to get started, what it will do next, and what the results will be after it is executed.

BDD helps teams more accurately communicate requirements, discover defects early, and build software that remains sustainable over time.

**DevOps**

DevOps is about removing the barriers between traditionally siloed teams, development and operations. Under a DevOps model, development and operations teams work together across the entire software application life cycle, from development and test through deployment to operations.

## **Benefits of DevOps**

* **Speed.** DevOps practices let you move at the velocity you need to innovate faster, adapt to changing markets better, and become more efficient at driving business results.
* **Rapid delivery.**When you increase the pace of releases, you can improve your product faster and build competitive advantage.
* **Reliability.**DevOps practices like [continuous integration and continuous delivery](https://www.synopsys.com/glossary/what-is-cicd.html) can ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining an optimum experience for end users.
* **Improved collaboration.**Under a DevOps model, developers and operations teams collaborate closely, share responsibilities, and combine their workflows. This reduces inefficiencies and saves time.
* **Security.**You can adopt a DevOps model without sacrificing security by using automated, integrated [security testing tools](https://www.synopsys.com/software-integrity/security-testing.html).

**B. Coding best practices – Industry standards followed to maintain the code quality:**

Importance of code quality and coding standards-

Code quality is crucial for software development. It has a significant impact on the overall quality of software. You can either define a code as good/high-quality code or bad/low-quality code.

Of course, quality, either good or bad, is a subjective matter. Different software development teams may have different definitions depending on the context of coding.

The reason coding standard is so important is highlighted in the real-world stats:

* Five seconds of loading time engages the audience 70% longer than nineteen seconds of loading time.
* 100-millisecond drop in website speed also drops the rate of conversion by almost 7%
* 79% of online shoppers won't return to a website with poor site performance.

## **Why Invest in Good Quality Code?**

Software developers globally adhere to certain coding standards to maintain a quality development environment. Given below are some key benefits when these standards are followed:

* Easier to read
* Easier to maintain
* Easier to understand
* Reusable codes
* Effective performance
* Consistent throughout the solution/software
* Easy knowledge transfer and demonstration

## **Why Code Quality Matters?**

Code quality is the usefulness and maintainability of code throughout the use of the application. It covers good and bad quality codes. It also provides other useful insights about coding standards in software engineering.

Code quality also measures how a code communicates between developers. Even similar software might show a huge difference in coding standards between two developers. However, what matters is consistent code quality throughout the software. A good code is clear and straightforward, bug-free, well tested, documented, refactored, and performant.

Developers spend a sizeable amount of their time addressing technical glitches and fixing bugs. Hence, it is imperative to focus on programming standard, which lies in the code. Code quality isn't a new parameter. It has existed since the 1970s, and it is now a necessity. 90% of companies are using code review tools to improve their code quality.

**Coding Standards Best Practices Help:**

Coding standards best practices are best defined as an assortment of essential rules, best practices, and guidelines to help programmers write good and cleaner code.

Programming standard also promotes sound practices of programming and improves efficiency, while checking that the software is:

* Safe, secure, and hack-proof that can be used without hurting or harming
* Reliable and easy to maintain, which works as per norms, and the codebase also grows
* Testable at the code level
* Compatible or portable with different implementation environments that lead to consistent results.

Both beginner and experienced software developers should follow the coding standards best practices to write good, clean, secure, and reliable code. This reduces future rework as the codebase grows and accelerates marketing time and software performance.

When you follow programming standards, you ensure compliance with the current industry standards, such as IEC and ISO, besides enhanced consistency, security, and code reliability.

## **Benefits and Importance of Code Quality**

The following qualities define coding standards in software engineering:

### 1. Increased Efficiency

Coders spend approximately 75% of their time on debugging. Rather than increasing development costs, executing coding guidelines to identify bugs earlier, and fixing them in due time, is essential. This measure, however, is an excellent long-term strategy for increased efficiency.

Efficiency directly correlates to the speed and performance of software, whereby quality is evaluated. Write reusable codes to reduce resource consumption and use appropriate data types, looping, and function at proper places.

### 2. Reduces Risk of Project Failure

According to this report, 14% of IT projects have been deemed a failure. Implement good quality code to avoid failures and reduce future risks.

### 3. Easy Code Maintenance

Writing a good uniform code makes maintenance easier by decreasing the time of bug detection. If you want to switch IT firms, a new coder could easily navigate the code to identify and fix bugs.

### 4. Creates Clean, Cost-Efficient Code

When you push for coding standards, you get excellent code that can drastically reduce [**software development costs**](https://radixweb.com/blog/cost-to-hire-software-developer) and efforts. Reusing the code saves time, ensuring you can deliver before your deadline.