SOFTWARE ENG

01 March 2024 02:19 AM

Creator of internet =
Vinton Cerf and Bob Kahn

Tim Berners-Lee invented the World Wide Web while working at CERN in 1989

Linus Benedict Torvalds (/ˈliːnəs ˈtɔːrvɔːldz/ *LEE-nəs TOR-vawldz*,[2] Finland Swedish: [ˈliːnʉs ˈtuːrvalds] ⓘ; born 28 December 1969) is a Finnish-American software engineer who is the creator and lead developer of the Linux kernel. He also created the distributed version control system Git.

The term Waterfall was first coined by Winston W. Royce in 1970

The first handheld calculator was a 1967 prototype called Cal Tech, whose development was led by Jack Kilby at Texas Instruments

Electronic Numerical Integrator And Computer-eniac-first computer

Cookies were created in 1994 by Lou Montulli, a web browser programmer at Netscape Communications.

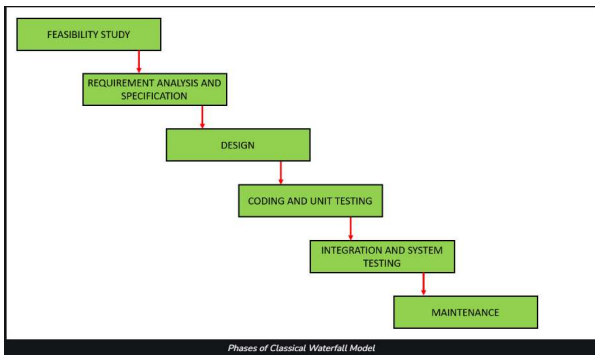
The term 'Internet of Things' was coined in 1999 by the computer scientist Kevin Ashton. While working at Procter & Gamble

Definition. The term big data has been in use since the 1990s, with some giving credit to John Mashey for popularizing the term.

The waterfall model is useful in situations where the project requirements are well-defined and the project goals are clear. It is often used for large-scale projects with long timelines, where there is little room for error and the project stakeholders need to have a high level of confidence in the outcome.

Features of the Waterfall Model

1. **Sequential Approach**: The waterfall model involves a sequential approach to software development, where each phase of the project is completed before moving on to the next one.
2. **Document-Driven:** The waterfall model relies heavily on documentation to ensure that the project is well-defined and the project team is working towards a clear set of goals.
3. **Quality Control:** The waterfall model places a high emphasis on quality control and testing at each phase of the project, to ensure that the final product meets the requirements and expectations of the stakeholders.
4. **Rigorous Planning**: The waterfall model involves a rigorous planning process, where the project scope, timelines, and deliverables are carefully defined and monitored throughout the project lifecycle.



Phases of Classical Waterfall Model

Importance of Waterfall Model

1. **Clarity and Simplicity:** The linear form of the Waterfall Model offers a simple and unambiguous foundation for project development.
2. **Clearly Defined Phases:** The Waterfall Model's phases each have unique inputs and outputs, guaranteeing a planned development with obvious checkpoints.
3. **Documentation:** A focus on thorough documentation helps with software comprehension, upkeep, and future growth.
4. **Stability in Requirements:** Suitable for projects when the requirements are clear and steady, reducing modifications as the project progresses.
5. **Resource Optimization:** It encourages effective task-focused work without continuously changing contexts by allocating resources according to project phases.
6. **Relevance for Small Projects:** Economical for modest projects with simple specifications and minimal complexity.

Advantages of the Classical Waterfall Model

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model.

- **Easy to Understand:** The Classical Waterfall Model is very simple and easy to understand.
- **Individual Processing:** Phases in the Classical Waterfall model are processed one at a time.
- **Properly Defined:** In the classical waterfall model, each stage in the model is clearly defined.
- **Clear Milestones:** The classical Waterfall model has very clear and well-understood milestones.
- **Properly Documented:** Processes, actions, and results are very well documented.
- **Reinforces Good Habits:** The Classical Waterfall Model reinforces good habits like define-before-design and design-before-code.
- **Working:** Classical Waterfall Model works well for smaller projects and projects where requirements are well understood.

Disadvantages of the Classical Waterfall Model

The Classical Waterfall Model suffers from various shortcomings we can't use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model.

- **No Feedback Path:** In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.
- **Difficult to accommodate Change Requests:** This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but the customer's requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.
- **No Overlapping of Phases:** This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can't be maintained. To increase efficiency and reduce cost, phases may overlap.
- **Limited Flexibility:** The Waterfall Model is a rigid and linear approach to software development, which means that it is not well-suited for projects with changing or uncertain requirements. Once a phase has been completed, it is difficult to make changes or go back to a previous phase.
- **Limited Stakeholder Involvement:** The Waterfall Model is a structured and sequential approach, which means that stakeholders are typically involved in the early phases of the project (requirements gathering and analysis) but may not be involved in the later phases (implementation, testing, and deployment).
- **Late Defect Detection:** In the Waterfall Model, testing is typically done toward the end of the development process. This means that defects may not be discovered until late in the development process, which can be expensive and time-consuming to fix.
- **Lengthy Development Cycle:** The Waterfall Model can result in a lengthy development cycle, as each phase must be completed before moving on to the next. This can result in delays and increased costs if requirements change or new issues arise.
- **Not Suitable for Complex Projects:** The Waterfall Model is not well-suited for complex projects, as the linear and sequential nature of the model can make it difficult to manage multiple dependencies and interrelated components.

Rapid prototyping is an iterative approach to the design stage of an app or website. The objective is to quickly improve the design and its functionality using regularly updated prototypes and multiple short cycles. This saves time and money by solving common design issues before product development begins, helps businesses to reach market quicker, and puts the focus of development on the needs of the end-user.

In the context of software development, rapid prototyping is normally associated with the Rapid Application Development (RAD) methodology, although you can also use it alongside an agile methodology. In some cases, the end product is actually a new product. However, in many situations, success is defined by improved user experience, a more seamless user interface, and other clear improvements to existing software.

The rapid prototyping process involves three simple steps:

Prototyping – The team creates one or more initial rapid prototypes. This is a visual representation of the design specifications as set out in the requirements document. The prototype may be either low-fidelity or high-fidelity (more to come on high- and low-fidelity prototypes) and may be interactive or non-interactive.

Feedback – The creators share the prototype with other team members, stakeholders, and focus groups made up of the intended end-users. Everyone evaluates both the design and usability before submitting feedback.

Improvement – The feedback is used to create a new iteration of the prototype. The design process then cycles round to Step 2 for further feedback. This continues until there are no more changes or a specified cut-off is reached (either a date, a number of iterations, or a clearly finished product).

What are cookies?

Cookies are small files which are stored on a user's computer. They are used to hold a modest amount of data specific to a particular client and website and can be accessed either by the web server or by the client computer When cookies were invented, they were basically little documents containing information about you and your preferences. For instance, when you select your language in which you want to view your website, the website would save the information in a document called a cookie on your computer, and the next time when you visit the website, it would be able to read a cookie saved earlier. That way the website could remember your language and let you view the website in your preferred language without having to select the language again. A cookie can contain any type of information such as the time when you visited the website, the items that you added into your shopping basket, all the links you clicked in website, etc.,

Who can access?

If a cookie is created in a particular website, and you visit another website later, the latter would not be able to read the contents from the first website, in other words only the same website that saves information to a cookie can access it.

Types of internet Cookies

Session Cookies: A session cookie only lasts for the duration of users using the website. A web browser normally deletes session cookies when it quits. A session cookie expires if the user does not access the website for a period of time chosen by the server(idle timeout). They only last for the duration of time we are on the site. If someone comes and uses our computer, they would not be able to see anything on the sites that use session cookies, because they need to enter the username and password again. **Persistent Cookies:** A persistent cookie outlast user sessions. If a persistent cookie has its maximum age 1 year, then within a year, the initial value set in the cookie would be sent back to the server every time the user visits the server. This could be used to record a vital piece of information such as how the user initially came to the website. For this reason, persistent cookies are also called tracking cookies. These are kind of cookies are used on websites that need to know who we are but offer us the ability to "remember me" when you enter username and password. **Example:** Gmail. **Third-party cookies:** Third party cookies are the cookies being set with the different domain than the one shown in the address bar. For example, if you were to visit Sendflowers.com, it may set the cookie for the address of SomeAdvertiser.com. Later, when you visit RebuidEngines.com it may set the cookie for SomeAdvertiser.com. Both of these cookies will be used by SomeAdvertiser.com to ascertain that you are a person who sends flowers and works on cars. These are used by advertising agencies who have clients that pay for displaying ads for products and services but don't want to waste money displaying them to people who aren't going to be a customer.

HTTP Cookies: When you send a request to the server, the server sends a reply in which it embeds the cookie which serves as an identifier to identify the user. So, next time when you visit the same website, the cookie lets the server know that you are visiting the website again. **Related Article:** Javax.servlet.http.Cookie class in Java

- **JavaScript:** Brendan Eich (invented in 1995)
- **C++:** Bjarne Stroustrup (developed in the early 1980s)
- **CSS:** Håkon Wium Lie (invented in 1994)
- **HTML:** Tim Berners-Lee (invented in 1989)
- **Python:** Guido van Rossum (developed in the late 1980s)

The **Internet of things** (**IoT**) describes devices with sensors, processing ability, software and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks.[1][2][3][4][5] The Internet of things encompasses electronics, communication, and computer science engineering. "Internet of things" has been considered a misnomer because devices do not need to be connected to the public internet; they only need to be connected to a network[6] and be individually addressable.[7][8]

Big data refers to extremely large and diverse collections of structured, unstructured, and semi-structured data that continues to grow exponentially over time. These datasets are so huge and complex in volume, velocity, and variety, that traditional data management systems cannot store, process, and analyze them.

The Agile Manifesto was written in 2001 by seventeen independent-minded software practitioners. While the participants didn't often agree, they did find consensus around four core values.

The Authors

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
Robert C. Martin
Steve Mellor
Dave Thomas
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Ken Schwaber
Jeff Sutherland

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

Pair programming is a development technique in which two programmers work together at single workstation. Person who writes code is called a **driver** and a person who observes and navigates each line of the code is called **navigator.** They may switch their role frequently. Sometimes pair programming is also known as pairing. **Pairing Variations :** There are three pairing variations –

- **Newbie-newbie** pairing can sometimes give a great result. Because it is better than one solo newbie. But generally, this pair is rarely practiced.
- **Expert–newbie** pairing gives significant results. In this pairing, a newbie can learn many things from expert, and expert gets a chance to share his knowledge with newbie.
- **Expert–expert** pairing is a good choice for higher productivity as both of them would be expert, so they can work very efficiently.

Advantages of Pair Programming :

- **Two brains are always better than one** – If driver encounters a problem with code, there will be two of them who'll solve problem. When driver is writing code, navigator can think about a solution to problem.
- **Detection of coding mistakes becomes easier** – Navigator is observing each and every line of code written by driver, so mistakes or error can be detected easily.
- **Mutual learning** – Both of them can share their knowledge with each other and can learn many new things together.
- **Team develops better communication skills** – Both of them share knowledge and work together for many hours a day and constantly share information with each other so this can help in developing better communication skills, especially when one of members is a newbie and other is an expert.

Disadvantages of Pair Programming :

- **Team Fit** – High-intensity communication of pair programming is not a good fit for every developer. Sometimes, drivers are supposed to speak loud as they write code. Some people may not agree on idea of sitting, literally shoulder-to-shoulder, with a colleague for eight hours a day. Some experienced developers are more productive in solo rather than in pair programming.
- **Newbie-newbie pairing problem** – Newbie–newbie pairing can produce results better than two newbie working independently, although this practice is generally avoided because it is harder for newbie to develop good habits without a proper role model.

Compiled languages:

- **Translation process:** Before execution, the entire program is translated into a low-level language, typically machine code, that can be directly understood by the computer's processor. This translation happens using a program called a compiler.
- **Execution:** Once compiled, the machine code can be run directly by the processor without further translation, making it generally **faster and more efficient** at runtime.
- **Examples:** C, C++, Java, Rust

Interpreted languages:

- **Translation process:** The code is translated line by line at runtime by a program called an interpreter. The interpreter reads each line, understands its meaning, and then executes the corresponding instructions.
- **Execution:** While interpreted languages can be slower than compiled languages due to the ongoing translation, they offer several advantages:
  - **Portability:** Interpreted code can run on any machine with the appropriate interpreter installed, regardless of the underlying hardware architecture.
  - **Flexibility and ease of development:** Changes to the code can be made and tested quickly without needing to recompile the entire program.
- **Examples:** Python, JavaScript, Ruby, PHP

In software engineering, **velocity** refers to a metric used primarily in **agile methodologies** to estimate the **team's capacity to deliver work** over a specific timeframe. It essentially reflects the **average amount of work a team can complete within a sprint**.

Here's a breakdown of the key points about velocity:

Measuring Velocity:

- Velocity is typically measured in **story points**, which are **relative units** assigned to user stories based on their complexity and effort required to complete them.
- It is calculated by **averaging the number of story points completed in previous sprints**.

Purpose of Velocity:

- **Planning and forecasting:** Velocity helps teams **estimate the amount of work they can commit to in future sprints**. This allows for **realistic planning** and **predictable release cycles**.
- **Monitoring progress:** Velocity can be tracked over time to **identify trends** and **improve team efficiency**. If velocity consistently drops, it might indicate roadblocks or changing project complexities that need to be addressed.

Important Considerations:

- **Velocity is relative, not absolute:** It is specific to each team and project and should not be compared across different teams or projects.
- **Focus on trends, not individual data points:** Short-term fluctuations in velocity are normal and may not be significant. It's crucial to analyze trends over multiple sprints to understand the team's overall performance.
- **Velocity is not a measure of team performance:** While it can be used to monitor progress, it should not be solely used to evaluate individual or team performance. Other factors like code quality, team morale, and unexpected challenges also contribute to project success.

Embracing Change in Software Engineering

- **Traditional Approach Vs. Agile Approach:** Traditionally, software projects followed a rigid waterfall model, where detailed plans were laid out upfront, leaving little room for adjustments as the project unfolded. However, the software industry has widely adopted agile methodologies that emphasize adaptability and responsiveness to change.

- **Reasons for embracing change:**
  - **Market Evolution:** In the rapidly changing technology landscape, requirements and market needs can shift quickly. Adapting to these changes is crucial for the success of the product.
  - **User Feedback:** As users interact with early versions of the software, they provide valuable feedback that helps shape the product. Agile methodologies welcome this feedback and incorporate it iteratively.
  - **Technological Advancements:** New tools and technologies emerge frequently, offering better ways to solve problems. Teams that embrace change can leverage these advantages.

- **How is change accommodated?**
  - **Iterative development:** Agile methods favor short development cycles (sprints) with frequent releases. This allows teams to get feedback early, incorporate it, and pivot the product direction if necessary.
  - **Prioritization:** The product owner continuously re-prioritizes the product backlog based on changing needs, insights, and feedback.
  - **Retrospectives:** Teams regularly conduct retrospectives to analyze their processes and identify areas for improvement. This helps them react faster and better to future changes.

Sustainability in Software Engineering

In software engineering, sustainability primarily focuses on creating and maintaining software systems that prioritize long-term viability in the following areas:

- **Resource Efficiency:** Sustainable software is designed to be mindful of its hardware and energy consumption. This includes:
  - **Lightweight design:** Focusing on efficient algorithms and data structures to minimize processing power demands.
  - **Scalability:** Ability to handle increasing workloads without excessive resource scaling.
  - **Minimizing e-waste:** Designing systems that are maintainable and upgradeable to prolong their useful lifespan and reduce electronic waste.