

Software Development Lifecycle Models

Software Development Life Cycle, is a step-by-step process that software teams follow to create and release new software. It starts with planning, where they decide what the software needs to do, then moves through designing, building, testing, and finally releasing the software to users. After release, they continue to maintain and improve it as needed.

1) Waterfall Model

Implantation Year: Late 1950s to early 1970s (popularized in the 1970s).

Phases:

1. Requirements Gathering and Analysis:

- Collect and document all requirements from stakeholders.
- Analyze requirements to ensure clarity and completeness.

2. System Design:

- Create system architecture and design based on gathered requirements.
- Specify hardware and system requirements.

3. Implementation:

- Develop software according to the design specifications.
- Divide the project into manageable units or modules.

4. Testing:

- Perform unit testing for each module to ensure individual functionality.
- Conduct system testing to verify the software as a whole.

5. Deployment:

- Install and release the software to users or deploy it in the customer environment.

6. Maintenance:

- Provide ongoing support, bug fixes, and updates after deployment.



Pros:

- This model is simple and easy to understand.
- This is very useful for small projects.
- This model is easy to manage.
- The end goal is determined early.
- Each phase of this model is well explained.
- It provides a structured way to do things.
- This is a base model, all the SDLC models that came after this were created keeping this in mind, although they worked to remove its shortcomings.
- In this model, we can move to the next phase only after the first phase is successfully completed so that there is no overlapping between the phases.

Cons:

- In this model, complete and accurate requirements are expected at the beginning of the development process.
- Working software is not available for very long during the development life cycle.
- We cannot go back to the previous phase due to which it is very difficult to change the requirements.
- Risk is not assessed in this, hence there is high risk and uncertainty in this model.
- In this the testing period comes very late.
- Due to its sequential nature this model is not realistic in today's world.
- This is not a good model for large and complex projects.

2) V-Model:

Implantation Year: Late 1980s to early 1990s.

Phases

1. Requirements Analysis:

- Define and document detailed requirements, ensuring they are testable.
- Establish traceability between requirements and corresponding test cases.

2. System Design:

- Design system architecture and specify interfaces between modules.
- Create detailed design documents for each module.

3. Module Design:

- Design individual modules or components based on functional and non-functional requirements.
- Document module specifications and interfaces.

4. Implementation:

- Code modules according to design specifications.
- Perform unit testing for each module.

5. Integration and Testing:

- Integrate modules and perform integration testing.
- Conduct system testing to verify end-to-end functionality.

6. Acceptance Testing:

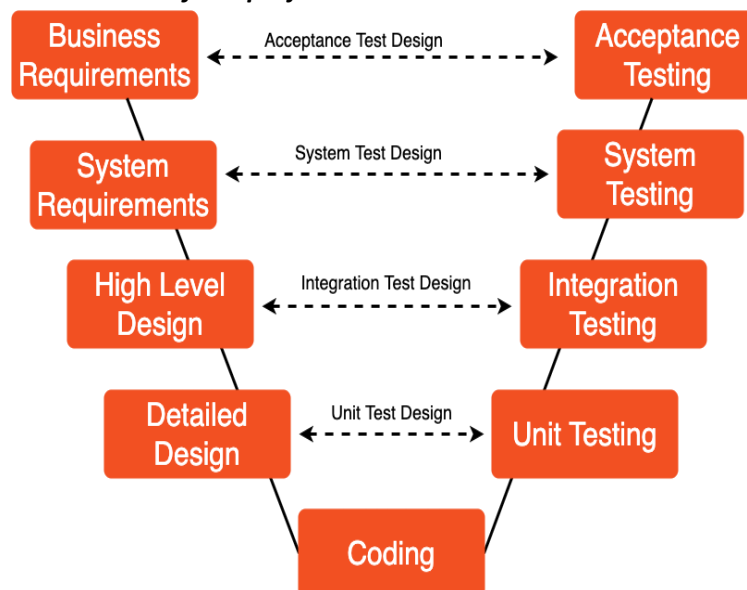
- Verify that the system meets user requirements through user acceptance testing.

• Pros:

- This is a simple and easy to use model.
- Planning, testing and designing tests can be done even before coding.
- This is a very disciplined model, in which phase by phase development and testing goes on.
- Defects are detected in the initial stage itself.
- Small and medium scale developments can be easily completed using it.

• Cons:

- This model is not suitable for any complex projects.
- There remains both high risk and uncertainty.
- This is not a suitable model for an ongoing project.
- This model is not at all suitable for a project which is unclear and in which there are changes in the requirement.



3) Spiral Model:

Implantation Year: Late 1980s (proposed by Barry Boehm in 1986).

Phases

1. **Determine Objectives:**

- Set project goals and identify alternative solutions.
- Define constraints and evaluate risks associated with each alternative.

2. **Risk Analysis:**

- Identify and analyze potential risks.
- Develop risk mitigation strategies and contingency plans.

3. **Prototype Development:**

- Build and test prototypes to validate concepts and mitigate key risks.
- Gather feedback from stakeholders and refine prototypes iteratively.

4. **Customer Evaluation:**

- Review prototypes with stakeholders and gather feedback.
- Evaluate feasibility and address concerns raised during evaluation.

5. **Plan Next Iteration:**

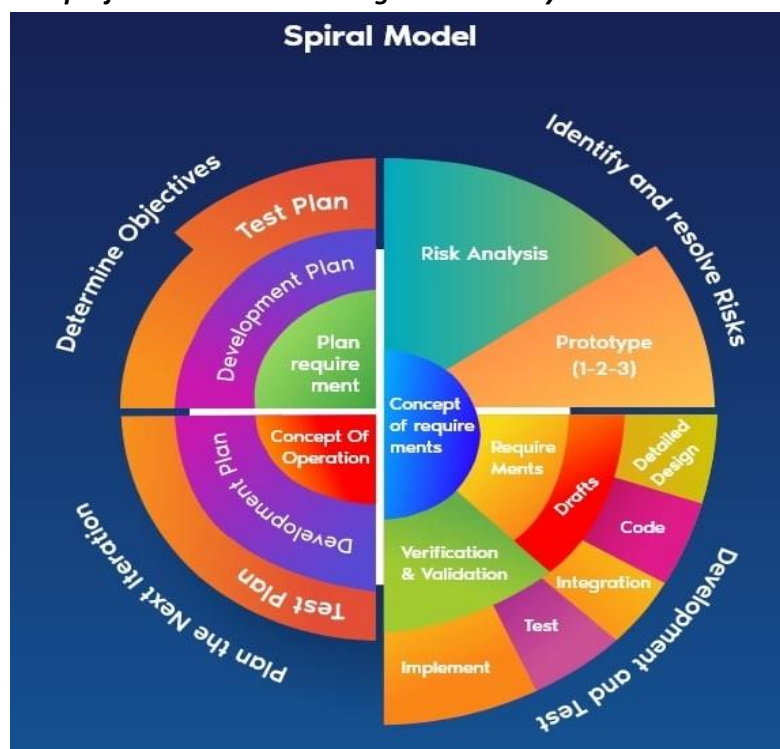
- Plan the next iteration based on feedback and risk analysis.
- Adjust project objectives and development approach as necessary.

• Pros:

- Emphasizes risk management by addressing risks early in the project.
- Accommodates changes throughout the development process.
- Suitable for large and complex projects with evolving requirements.
- Allows for iterative development and continuous refinement.

• Cons:

- Requires experienced personnel for effective risk assessment.
- May lead to increased costs and time if risks are not managed effectively.
- Complex and time-consuming due to the iterative nature of the model.
- Difficult to estimate project timelines and budgets accurately.



4) Iterative Model:

Implantation Year: The concept dates back to the 1950s, but it gained popularity in the 1990s and early 2000s.

Phases

1. **Plan Iteration:**

- Define goals and requirements for the iteration.
- Prioritize features and establish iteration timeline.

2. **Develop Features:**

- Implement new features or enhancements according to iteration goals.
- Use incremental development to deliver functionality in iterations.

3. **Test Features:**

- Test implemented features for functionality and compatibility.
- Conduct regression testing to ensure existing features remain functional.

4. **Review and Refine:**

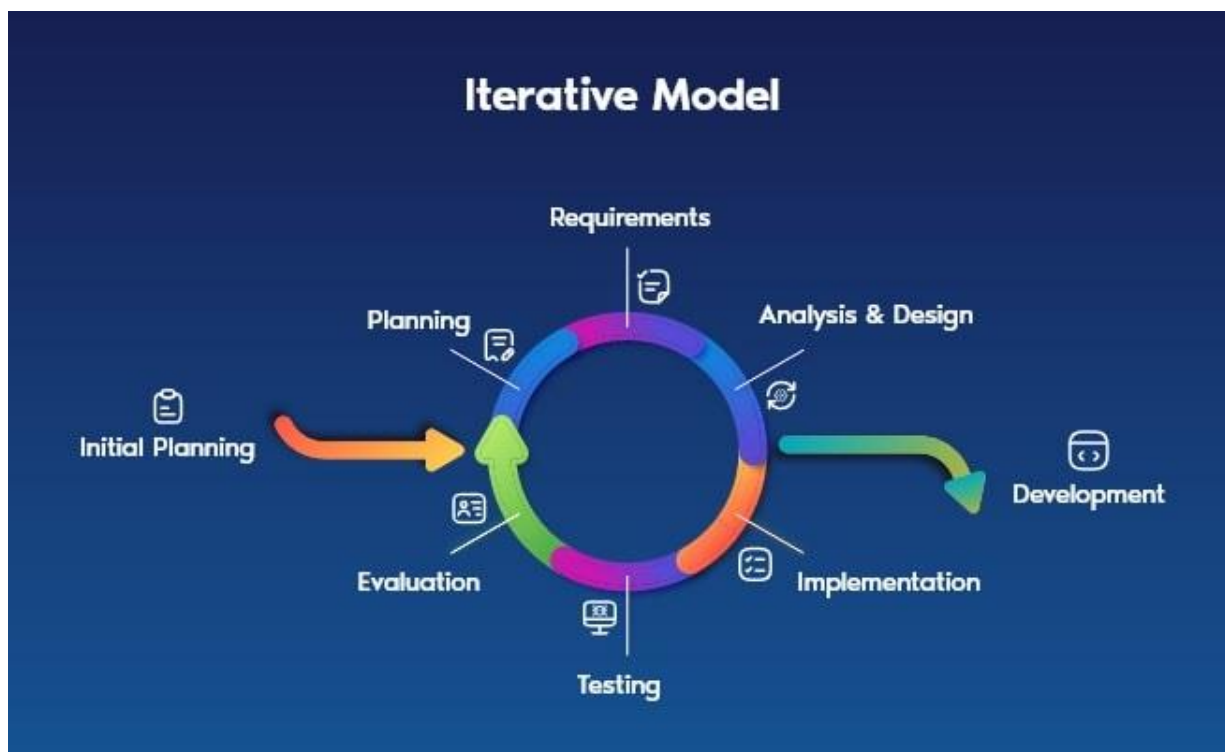
- Gather feedback from stakeholders on implemented features.
- Incorporate feedback to refine and improve features as necessary.

• Pros:

- Allows for flexibility and adaptation to changing requirements.
- Enables early delivery of partial solutions, promoting stakeholder engagement.
- Enhances product quality through iterative refinement and feedback.
- Reduces the risk of project failure by addressing issues early.

• Cons:

- Requires a clear vision of the end product to be effective.
- May lead to scope creep if not managed properly.
- Can be challenging to coordinate and manage multiple iterations simultaneously.
- Relies heavily on active stakeholder involvement and communication.



5) Incremental Model:

Implantation Year: The concept has roots in the 1960s, but it gained prominence in the 1970s and 1980s.

Phases

1. **Identify Modules:**

- Decompose the project into modules or components based on functionality.
- Prioritize modules for incremental development based on business value.

2. **Develop Increments:**

- Develop and test each increment independently.
- Use iterative development to deliver increments in stages.

3. **Integrate Increments:**

- Integrate increments into the overall system.
- Perform integration testing to ensure modules work together seamlessly.

4. **Test Integrated System:**

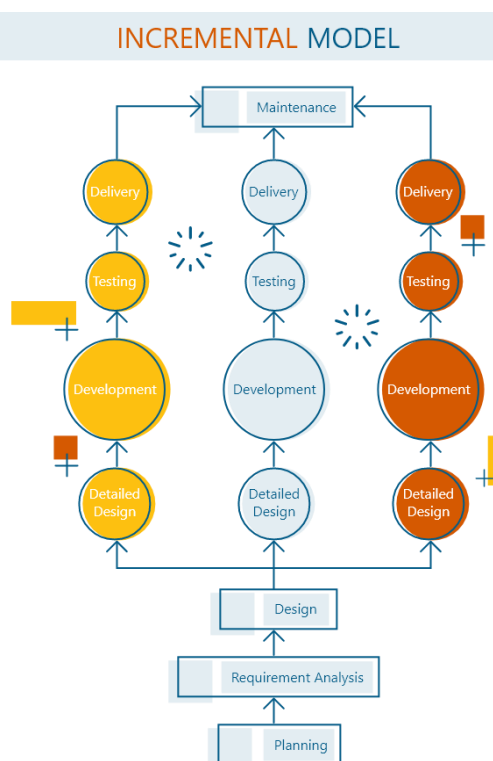
- Test the entire system for functionality, performance, and reliability.
- Conduct system-level testing to validate end-to-end functionality.

• Pros:

- Prepares the software fast.
- Clients have a clear idea of the project.
- Changes are easy to implement.
- Provides risk handling support, because of its iterations.
- Adjusting the criteria and scope is flexible and less costly.
- Comparing this model to others, it is less expensive.
- The identification of errors is simple.

• Cons:

- Requires a clear vision of the end product to be effective.
- May lead to scope creep if not managed properly.
- Can be challenging to coordinate and manage multiple iterations simultaneously.
- Relies heavily on active stakeholder involvement and communication.



6) Rapid Application Development (RAD):

Implantation Year: Late 1980s to early 1990s.

Phases

1. Define Requirements:

- Gather high-level requirements from stakeholders.
- Prioritize requirements based on business value and feasibility.

2. Develop Prototypes:

- Build quick prototypes to visualize and validate requirements.
- Use iterative prototyping to refine prototypes based on feedback.

3. Gather Feedback:

- Solicit feedback from stakeholders on prototype functionality and usability.
- Incorporate feedback to refine prototypes iteratively.

4. Refine Prototypes:

- Improve prototypes based on feedback and evolving requirements.
- Iterate on prototype design and functionality to converge on a final solution.

5. Implement and Integrate:

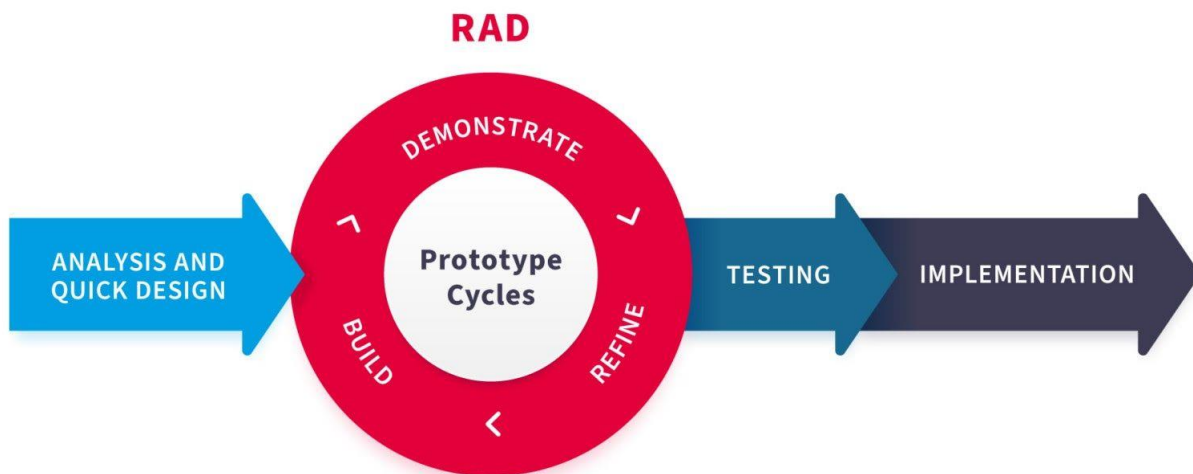
- Develop the final solution based on refined prototypes.
- Integrate components and perform system-level testing.

• Pros:

- Requires a clear vision of the end product to be effective.
- May lead to scope creep if not managed properly.
- Can be challenging to coordinate and manage multiple iterations simultaneously.
- Relies heavily on active stakeholder involvement and communication.

• Cons:

- Requires a clear vision of the end product to be effective.
- May lead to scope creep if not managed properly.
- Can be challenging to coordinate and manage multiple iterations simultaneously.
- Relies heavily on active stakeholder involvement and communication.



7) Agile Model:

Implantation Year: Early 2000s (Agile Manifesto published in 2001).

Phases

1. **Plan Iteration:**

- Define iteration goals and prioritize requirements.
- Establish iteration timeline and team capacity.

2. **Develop and Test Features:**

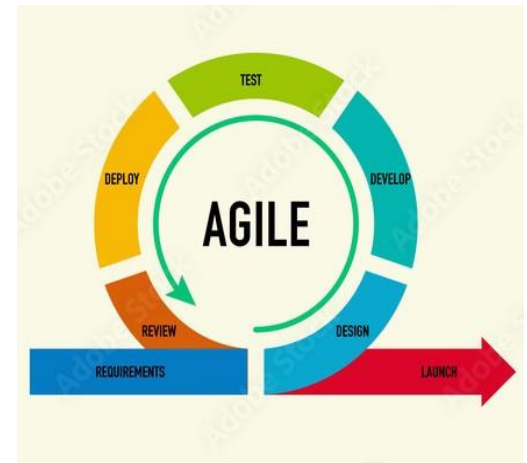
- Implement and test features incrementally throughout the iteration.
- Use test-driven development to ensure quality and maintainability.

3. **Review and Demo Features:**

- Review completed features with stakeholders for feedback.
- Conduct sprint review meetings to demo functionality and gather input.

4. **Retrospective and Adaptation:**

- Reflect on iteration performance and identify areas for improvement.
- Adapt development processes and practices based on retrospective findings.



• Pros:

- **Flexibility:** Agile allows for flexibility and adaptability to changing requirements throughout the development process. Teams can respond quickly to customer feedback and market changes, ensuring that the final product meets user needs effectively.
- **Continuous Feedback:** Agile emphasizes continuous feedback loops, promoting collaboration between development teams and stakeholders. Regular feedback sessions ensure that the product remains aligned with customer expectations, leading to higher satisfaction levels.
- **Early Delivery:** Agile facilitates the early delivery of working software increments, enabling stakeholders to see tangible progress and provide input at each stage. This iterative approach fosters a sense of ownership and engagement among stakeholders.
- **Transparency and Collaboration:** Agile promotes transparency and collaboration within development teams. Through practices like daily stand-up meetings, sprint planning, and retrospectives, team members share progress, challenges, and insights, fostering a culture of trust and accountability.

• Cons:

- **Customer Involvement:** Agile requires active customer involvement throughout the development process. Maintaining continuous engagement with stakeholders can be challenging, particularly in large or distributed teams where communication may be more complex.
- **Scalability:** While Agile works well for small to medium-sized teams, scaling Agile practices to large or distributed teams can be challenging. Coordination and alignment across multiple teams may become more difficult, potentially leading to inconsistencies and inefficiencies.
- **Dependency on Team Skills:** Agile relies heavily on the skills and expertise of team members to self-organize, collaborate effectively, and deliver high-quality software increments. Teams with limited experience or expertise may struggle to implement Agile practices successfully.
- **Documentation and Technical Debt:** Agile prioritizes working software over comprehensive documentation, which can lead to technical debt if not managed properly. Balancing the need for agility with the necessity of maintaining adequate documentation and code quality can be a delicate task.

8) DevOps:

Implantation Year: Mid to late 2000s (term coined around 2009).

Phases

1. Continuous Development:

- Developers write and commit code frequently to version control systems.
- Use agile practices to break down work into small, manageable tasks.

2. Continuous Testing:

- Automate testing processes to validate changes in code.
- Use test automation frameworks to ensure code quality and reliability.

3. Continuous Integration:

- Integrate code changes into a shared repository frequently.
- Use continuous integration servers to build and test applications automatically.

4. Continuous Deployment:

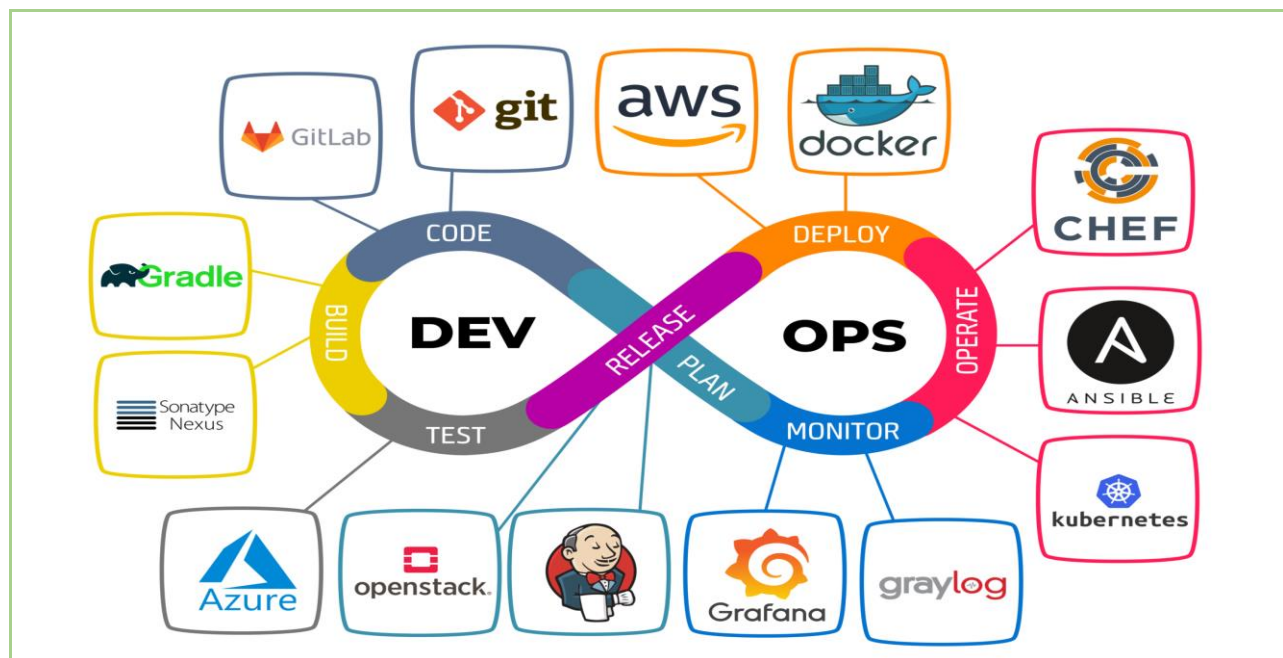
- Automate deployment processes to release code changes rapidly.
- Use deployment pipelines to automate testing and deployment tasks.

5. Continuous Monitoring:

- Monitor application and infrastructure performance in real-time.
- Use monitoring tools to detect and respond to issues proactively.

6. Continuous Feedback:

- Collect feedback from users and stakeholders to drive continuous improvement.
- Use feedback loops to inform development and operations practices.



• Pros:

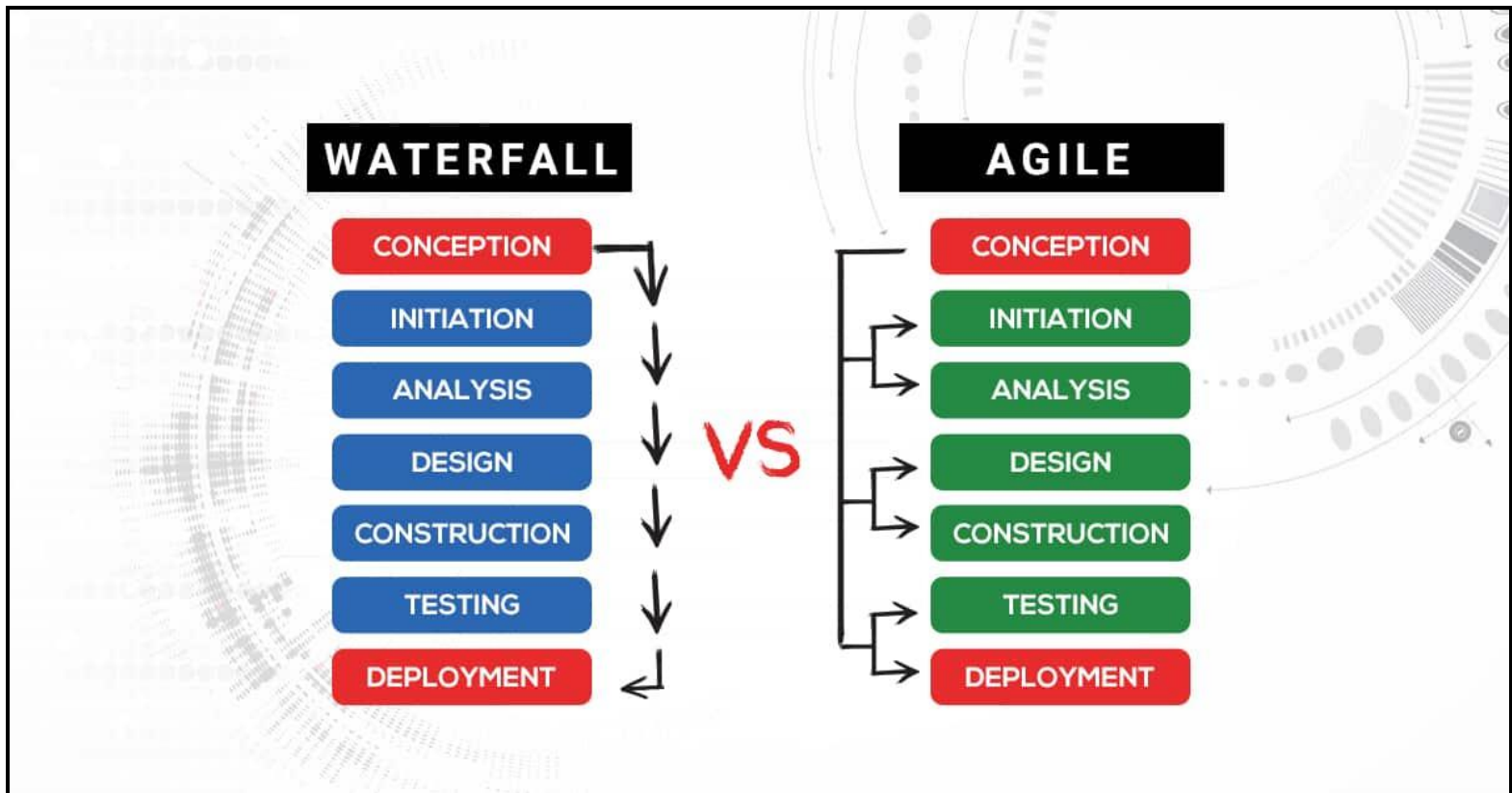
- **Faster Delivery:** DevOps enables faster delivery of features and updates by streamlining the development, testing, and deployment processes. Automation tools and practices reduce manual errors and accelerate deployment cycles, allowing teams to release software more frequently and reliably.
- **Improved Collaboration:** DevOps fosters collaboration and communication between development and operations teams. By breaking down silos and promoting shared responsibilities, DevOps encourages a culture of collaboration and collective ownership, leading to smoother workflows and faster problem resolution.
- **Continuous Monitoring and Feedback:** DevOps emphasizes continuous monitoring of systems and applications, enabling teams to detect and address issues proactively. Real-time feedback loops provide valuable insights

into system performance, user behavior, and potential issues, facilitating continuous improvement and optimization.

- **Infrastructure as Code (IaC):** DevOps promotes the use of infrastructure as code (IaC) to automate the provisioning, configuration, and management of infrastructure resources. IaC reduces manual overhead, improves consistency, and enhances scalability, making it easier to manage complex and dynamic environments..





- **Cons:**

- **Complexity and Learning Curve:** Implementing DevOps practices and tools can be complex and challenging, especially for organizations with traditional, siloed structures. Teams may require time and resources to acquire the necessary skills, adapt their processes, and overcome cultural resistance to change.
- **Security and Compliance Concerns:** DevOps introduces security and compliance challenges related to automation, rapid deployments, and shared responsibilities. Ensuring the security and compliance of applications and infrastructure requires careful planning, implementation of security best practices, and integration of security controls into the DevOps pipeline.
- **Toolchain Integration:** DevOps involves the use of various tools and technologies for automation, monitoring, and collaboration. Integrating and managing these tools effectively can be daunting, particularly for teams with diverse toolchains and heterogeneous environments. Compatibility issues, version conflicts, and tool sprawl may complicate the DevOps implementation process.
- **Cultural and Organizational Change:** DevOps requires a cultural shift and organizational change to break down silos, promote collaboration, and foster a culture of continuous improvement. Resistance to change, lack of buy-in from stakeholders, and entrenched organizational norms may impede the adoption and success of DevOps initiatives.



Waterfall vs Agile vs DevOps

demandsimplified.marketing

|  |  |  |  |
|---|---|---|--|
| <p>Approach</p> | <p>Waterfall model provides a linear sequential approach to managing software projects. Each phase depends on deliverables from the previous one.</p> | <p>Agile is a highly dynamic and iterative approach where you do not need the complete set of requirements to start with. You can develop some features and check customer response before next steps</p> | <p>DevOps is an Agile methodology encompassing Development (Dev) and Operations (Ops). It enables end-to-end lifecycle delivery of features, fixes, and updates at frequent intervals.</p> |
| <p>Year</p> | <p>1970</p> | <p>2001</p> | <p>2009</p> |
| <p>Scope and Schedule</p> | <p>Adjust schedule to preserve scope. Fixed requirements. Limited flexibility.</p> | <p>Adjust scope to preserve schedule. Iterative approach allows for prioritization</p> | <p>Adjust scope to preserve schedule. Highly responsive to business needs</p> |
| <p>Time to Market</p> | <p>Slow</p> | <p>Rapid</p> | <p>Continuous</p> |
| <p>Automation</p> | <p>Low</p> | <p>Varied</p> | <p>High</p> |
| <p>Customer Feedback</p> | <p>End of Project</p> | <p>After every sprint</p> | <p>Continuous</p> |
| <p>Quality</p> | <p>Low. Issues are identified only at testing stage</p> | <p>Better. Issues are identified after every sprint</p> | <p>High. Automated unit testing during development</p> |
| <p>Collaboration</p> | <p>Teams operate in silos</p> | <p>Multiple teams are involved, but not all</p> | <p>All stakeholders are involved from start to finish</p> |
| <p>Variations</p> | <p>V-model</p> | <p>Scrum, XP, LeSS, SAFe</p> | <p>CI/CD, DevSecOps</p> |