Process 1: Waterfall Model

The Waterfall model is one of the earliest methodologies used in software development and is a sequential (non-iterative) process model. It is called "Waterfall" because its diagrammatic representation resembles a cascade of waterfalls, where the flow of phases moves steadily downwards through the phases of conception, initiation, analysis, design, construction, testing, deployment, and maintenance. Here's a breakdown of the key characteristics and phases of the Waterfall model:

**Characteristics**

* **Linear and Sequential**: Each phase must be completed fully before the next phase can begin. There is no overlapping in the phases.
* **Well-defined Stages**: The requirements are very well understood and do not change over time. Each phase has specific deliverables and a review process.
* **Easy to Understand and Manage**: Due to its linear nature, it is straightforward to understand and manage. Each phase has specific deliverables and a review process.

**Phases**

1. **Requirements Analysis**: This phase involves gathering all the detailed information required to design and implement the project.
2. **System Design**: The system's architecture is designed, specifying hardware and system requirements and defining the overall system architecture.
3. **Implementation**: The actual coding of the software takes place in this phase.
4. **Integration and Testing**: Once the software is built, it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.
5. **Deployment**: After successful testing, the system is deployed to the customer for their use.
6. **Maintenance**: This phase involves making updates to the software, fixing issues that weren't discovered in earlier stages, and improving performance.

**Advantages**

* Simplicity and ease of use.
* Clear documentation and milestones.
* Well understood roles and processes.

**Disadvantages**

* Inflexibility to change. Once a phase is completed, it is difficult to go back and make changes.
* Assumes that requirements can be completely defined upfront, which is often not the case in complex projects.
* Risk and uncertainty are high, as testing is done after completion, which can lead to challenges in addressing fundamental issues discovered late in the process.

The Waterfall model is best suited for projects with very well-defined requirements and where changes are not expected. It has largely been succeeded by more flexible Agile methodologies in many types of software projects, especially those requiring frequent iteration and adaptation

**The Prototype Model** is a software development methodology that emphasizes the creation of a working prototype early in the development process to better understand customer requirements and expectations. Here's a summary based on the information from the slide titled "Process 1.5: Prototype model":

1. **Requirements Gathering**: The initial stage involves collecting the requirements and expectations of the end-users or clients to understand what needs to be developed.
2. **Quick Design**: Based on the gathered requirements, a quick design phase is undertaken to outline the basic structure and functionality of the software.
3. **Refine Requirements**: With a preliminary design in place, the requirements are refined to ensure they align with the potential capabilities of the final product.
4. **Build Prototype**: A prototype, which is a basic version of the software, is then built. This prototype is designed to include the key functionalities and features that the final product will have, but it is not the finished product.
5. **Customer Evaluation of Prototype**: The prototype is presented to the customer or end-users for evaluation. This feedback phase is crucial as it allows the development team to understand the user's perspective, including what aspects meet their needs and what needs improvement.
6. **Design, Implement, Test, Maintenance**: Following the evaluation and acceptance of the prototype by the customer, the development process moves into the more detailed phases of design, implementation, testing, and maintenance, culminating in the final product.
7. **Customer Acceptance**: The final stage is customer acceptance, where the finished product is delivered and assessed by the customer to ensure it meets the agreed-upon requirements and standards.

The Prototype Model is particularly useful in projects where the requirements are not fully understood at the outset or are expected to change. It allows for iterative feedback and adjustments early in the development cycle, helping to reduce misunderstandings and mismatches between the developed software and the customer's needs.

# Incremental Development (growing software)

Incremental Development is a software development approach that focuses on delivering software in small chunks, allowing for frequent feedback and testing. In this method, each iteration adds a layer of functionality, until the software is complete. This approach enables developers to quickly identify and fix issues, improve the user experience, and make changes as needed.

Pros:

There is a working system at all time

Clients can see the system and provide feedback

Progress is visible, rather than being buried in documents

PLAN DRIVEN -🡪 MON STAGE

**The Agile Manifesto** is a foundational document for agile software development, created by 17 software developers in 2001. It values:

1. **Individuals and interactions** over processes and tools.
2. **Working software** over comprehensive documentation.
3. **Customer collaboration** over contract negotiation.
4. **Responding to change** over following a plan.

These values emphasize flexibility, collaboration, and customer satisfaction. The manifesto also includes 12 principles to guide agile practices, focusing on adaptability, continuous improvement, and delivering value to customers early and often.

A white and black text on a white background

Description automatically generatedTop of Form

A qr code on a blue background

Description automatically generated



**Extreme Programming Practice**

* **People，not process**, are supported through pair programming, collective ownership and a process that avoids long working hours.
* Maintaining **simplicity** through constant refactoring of **code**.
* **Customer involvement** means full-time customer engagement with the team.
* **Change supported** through regular system releases.
* **Incremental development** is supported through small, frequent system releases.

It appears that the document does not contain specific text about Extreme Programming (XP) practices, or it may not be able to be displayed. Based on my training data, I can provide you with an overview of the core practices in Extreme Programming:

1. **User Stories**: Writing requirements from the perspective of end users in a non-technical language.
2. **Test-Driven Development (TDD)**: Writing tests before the actual code to ensure the code meets the required specifications.
3. **Pair Programming**: Two developers work together at one workstation. One writes code while the other reviews each line of code as it is typed in.
4. **Continuous Integration (CI)**: Code changes are integrated into the main branch frequently and tested to detect integration errors as quickly as possible.
5. **Refactoring**: Regular modification of the code to improve its structure and readability without changing its external behavior.
6. **Simple Design**: The system design should be as simple as possible at any given moment, with extra complexity avoided.
7. **Collective Ownership**: Every team member can contribute to any part of the codebase at any time.
8. **Sustainable Pace**: Teams should work at a pace that can be sustained indefinitely, avoiding burnout.
9. **Coding Standards**: The team agrees to a common coding standard to promote consistency and quality.
10. **Metaphor**: Using simple shared stories to describe how the software works.
11. **On-site Customer**: Having an actual or potential user on the team to provide rapid feedback and decisions.
12. **Small Releases**: The software is developed in small increments with new releases being delivered to customers frequently.

These practices are designed to improve software quality and responsiveness to changing customer requirements.

Top of Form

**Scrum**

The document doesn't seem to provide specific searchable text on Scrum. However, Scrum is a widely recognized Agile framework used primarily for managing software development projects. Here's a summary of its key elements based on my previous knowledge:

* **Roles**: Scrum has specific roles including the Scrum Master, who facilitates the process; the Product Owner, who represents the stakeholders; and the Development Team, which includes professionals who do the work of delivering a potentially releasable increment of "Done" product at the end of each Sprint.
* **Ceremonies**: Scrum includes several ceremonies or meetings that are held at regular intervals, such as Sprint Planning, Daily Scrum (or stand-up), Sprint Review, and Sprint Retrospective.
* **Artifacts**: Scrum uses several artifacts to help manage work, such as the Product Backlog (a list of everything that might be needed in the product), the Sprint Backlog (a list of tasks to be completed during the sprint), and the Increment (the sum of all the Product Backlog items completed during a Sprint and all previous Sprints).
* **Sprints**: Work is divided into Sprints, which are time-boxed iterations, typically lasting two to four weeks, at the end of which a usable product increment should be delivered.
* **Adaptation and Transparency**: Throughout the Scrum process, transparency, inspection, and adaptation are key pillars. Team members continuously reflect on what's working and what's not to improve the product and the process.

The goal of Scrum is to create a culture of collaboration, accountability, and continuous improvement to deliver the highest value product increments efficiently and effectively.

A diagram of a scrum

Description automatically generated

A white background with black text

Description automatically generated

A close-up of a scrum

Description automatically generated

* The ‘**Scrum master**’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog, and communicates with customers and management outside of the team.
* Short daily meeting (aka Scrums)
  + All team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
  + Agile principle: help each other
* Name the important agile techniques/practices that were introduced in XP 🡪 User stories for specification, Refactoring, Test-first development, Pair programming

SOFTWARE EFFORT AND COST

1. **Experience-based Techniques:** These methods rely on the experience of project managers and the development team. They make use of historical data from past projects that are similar in nature to the current project. The assumption is that past experience can provide a valuable basis for estimating the effort and cost of a new project. This method benefits from direct knowledge of how similar challenges were addressed in the past, allowing for a more intuitive and potentially faster estimation process.
2. **Algorithmic Cost Modeling:** This technique employs mathematical models to estimate the software effort and costs. The most famous example is the Constructive Cost Model (COCOMO), which uses a formula based on the size of the software (usually in lines of code) and various cost drivers that affect productivity. These models are more quantitative and aim to provide a more objective basis for estimates. They can be particularly useful when detailed information about the project is available to feed into the model.

Each of these techniques has its advantages and can be chosen based on the project's context, the available data, and the team's experience. Experience-based techniques are often favored in agile environments where flexibility and rapid adjustments are valued. In contrast, algorithmic cost modeling might be preferred in more traditional, plan-driven projects where detailed upfront planning is more common.

**AGILE PLANNING**

Agile planning within the context of the presentation focuses on several key points essential for effective software project management:

1. **Estimation as a Collective Activity:** Estimates are made collaboratively, not by a single individual, leveraging the expertise and perspectives of the entire team.
2. **Relative Size Estimation:** The presentation highlights the use of story points and epics to estimate the relative size and complexity of tasks rather than exact durations. This approach acknowledges the inherent uncertainty in software development tasks, especially for larger work units.
3. **Planning Poker:** A collaborative estimation technique where team members privately select their estimate for a task, then discuss the high and low estimates to reach a consensus. This method promotes understanding and accommodates diverse viewpoints within the team.
4. **Prioritization and Risk Management:** Tasks are prioritized based on discussions with stakeholders (such as the customer or teaching assistant in an academic setting), with an emphasis on including a risk or safety factor in planning.
5. **Planning Horizons:** The presentation outlines the importance of using different planning horizons (Day, Iteration, Release, Product) to manage tasks and expectations effectively. Each horizon offers a different focus, from daily tasks to the overall project timeline, helping teams navigate through the development process efficiently.
6. **Velocity:** A key concept in Agile planning is the measurement of a team's velocity, or the rate at which it completes work. Velocity is used to forecast project timelines based on the team's past performance, making it a critical factor in planning and adjusting project schedules.
7. **Agile Estimation Key Points:** The presentation emphasizes the separation of effort estimation from duration estimation, suggesting that project duration should be derived from the team's velocity. It advocates for spending only a small amount of time on estimation, focusing instead on the relative effort of tasks to compute project duration.

These principles underscore Agile's flexible, iterative approach to planning, where the focus is on adapting to change, collaborative decision-making, and delivering value in small increments.

DEV-OPS

DevOps is defined as a software development approach that integrates **development (Dev) and operations (Ops) teams to streamline and automate the entire software delivery process**. It focuses on collaboration, continuous integration, continuous delivery, and automation to accelerate software development, improve quality, and enhance the agility of organizations in delivering software products and services. This approach is pivotal in fostering a culture of communication and collaboration between developers, IT professionals, and other stakeholders involved in the software development lifecycle, thus enabling more efficient and reliable delivery of software products.

The presentation on DevOps outlines both the advantages and challenges associated with implementing DevOps practices. Here is a summary:

### Advantages:

* **Speed and Efficiency:** DevOps practices, such as continuous integration and continuous delivery, allow for more rapid deployment cycles, enabling organizations to bring products to market faster.
* **Improved Collaboration:** By fostering a culture of collaboration between development and operations teams, DevOps helps break down silos and improve communication, leading to more efficient problem-solving and innovation.
* **Enhanced Quality:** Continuous testing and monitoring allow for immediate feedback and quicker resolution of issues, which improves the overall quality of the software.
* **Increased Automation:** Automation of repetitive tasks reduces errors, frees up team members to focus on more value-added work, and speeds up the software delivery process.
* **Better Resource Management:** DevOps practices can lead to more efficient use of resources, with automation and streamlined processes reducing the need for manual intervention and overhead.

### Challenges:

* **Cultural Shift Required:** Implementing DevOps requires a significant cultural shift within the organization, with teams needing to adopt a collaborative mindset that may be different from traditional ways of working.
* **Skill Gaps:** The integration of development and operations requires team members to possess a broad set of skills, leading to potential skill gaps that organizations need to address through training and hiring.
* **Initial Implementation Effort:** Transitioning to DevOps practices can require significant upfront effort in terms of setting up the necessary infrastructure, tools, and processes.
* **Toolchain Complexity:** DevOps involves the use of various tools for automation, testing, and monitoring, which can become complex to manage and integrate effectively.
* **Security and Compliance:** Integrating security into the DevOps process (a practice often referred to as DevSecOps) can be challenging, requiring careful attention to ensure that rapid development cycles do not compromise security or compliance standards.

These pros and cons highlight that while DevOps offers significant benefits in terms of speed, efficiency, and quality, it also presents challenges that organizations need to carefully manage to ensure successful implementation.

Top of Form

The presentation on DevOps outlines both the advantages and challenges associated with implementing DevOps practices. Here is a summary:

### Advantages:

* **Speed and Efficiency:** DevOps practices, such as continuous integration and continuous delivery, allow for more rapid deployment cycles, enabling organizations to bring products to market faster.
* **Improved Collaboration:** By fostering a culture of collaboration between development and operations teams, DevOps helps break down silos and improve communication, leading to more efficient problem-solving and innovation.
* **Enhanced Quality:** Continuous testing and monitoring allow for immediate feedback and quicker resolution of issues, which improves the overall quality of the software.
* **Increased Automation:** Automation of repetitive tasks reduces errors, frees up team members to focus on more value-added work, and speeds up the software delivery process.
* **Better Resource Management:** DevOps practices can lead to more efficient use of resources, with automation and streamlined processes reducing the need for manual intervention and overhead.

### Challenges:

* **Cultural Shift Required:** Implementing DevOps requires a significant cultural shift within the organization, with teams needing to adopt a collaborative mindset that may be different from traditional ways of working.
* **Skill Gaps:** The integration of development and operations requires team members to possess a broad set of skills, leading to potential skill gaps that organizations need to address through training and hiring.
* **Initial Implementation Effort:** Transitioning to DevOps practices can require significant upfront effort in terms of setting up the necessary infrastructure, tools, and processes.
* **Toolchain Complexity:** DevOps involves the use of various tools for automation, testing, and monitoring, which can become complex to manage and integrate effectively.
* **Security and Compliance:** Integrating security into the DevOps process (a practice often referred to as DevSecOps) can be challenging, requiring careful attention to ensure that rapid development cycles do not compromise security or compliance standards.

These pros and cons highlight that while DevOps offers significant benefits in terms of speed, efficiency, and quality, it also presents challenges that organizations need to carefully manage to ensure successful implementation.

Top of Form

**Requirement**

A qr code on a blue square

Description automatically generated



**Requirements** in the context of software engineering are specifications of what should be implemented in the software system. They are statements that identify attributes, capabilities, characteristics, or qualities of a system for it to have value and utility to a user or stakeholder. Requirements are generally divided into two main categories:

1. **Functional Requirements**: These specify what the system should do, the tasks it should perform, and the functions it must support. They describe the interactions between the system and its environment independent of the implementation. For example, "The system shall allow users to enter their credentials to access their account information."
2. **Non-functional Requirements**: These define the system’s properties and constraints, such as performance metrics, security standards, and quality attributes. They describe how the system performs a particular function, rather than the function itself. For example, "The system shall provide response times of less than 2 seconds for all transactions in the customer interface during peak usage."

Requirements should be clear, unambiguous, and testable. They are fundamental for the system design, verification, and validation processes. Good requirements should contribute to the overall success of the software development process by providing a clear understanding of what is to be built, helping to avoid scope creep, and providing a basis for estimating costs and time to deliver the final product.

A qr code on a blue background

Description automatically generated



The INVEST checklist for user stories in Agile software development includes the following criteria:

1. **Independent**: The user stories should be independent, meaning they can be developed in any sequence and changes to one story do not affect the others.
2. **Negotiable**: User stories are not explicit contracts and should leave space for discussion.
3. **Valuable**: Each user story must deliver value to the end users.
4. **Estimable**: You must be able to estimate the size or difficulty of the user story.
5. **Small**: User stories should be small enough that they can be completed within one sprint.
6. **Testable**: A user story must provide enough information to make test development possible. It should be clear how to test the functionality, and the tests should be defined.

These criteria help teams to create user stories that are useful and actionable, ensuring that they can be effectively prioritized, implemented, and tested.

PROJECT PLANNING

A qr code on a blue background

Description automatically generated

B ANSWER

A white paper with black text

Description automatically generated

A diagram of activity

Description automatically generated

A screenshot of a computer program

Description automatically generated

A white background with black text and black text

Description automatically generated