University of Jordan

King Abdullah II School of Information Technology

Information Technology Project Management

1904472

WATERFALL MODEL

Instructor: Dr. Ahmad Khreissat

Ahmad Obeidat 2213379

Bana Waleed AlShinnawi 0229975

Dareen Majdi Qutishat 0221733

Lara Refat AlZaghloul 2220241

Wajd Abdullah AlFaouri 0222278

Zain Omar Bakri 022361

Introduction

Waterfall Model is a predictive model to software engineering in the field of Information Technology. It consists of 6 stages: Requirements, Design, Implementation, Verification, Deployment, and Maintenance. It is mainly used for projects that are not prone to change, with clear requirements up front. It requires a software team to meet with the client up front to agree on all the requirements. The team then completes the project fully, going through the stages, completing every stage before moving on to the next one. Then, the completed project is delivered to the client. It has several advantages and disadvantages in comparison to other software project management models, which will be discussed further along with an in depth look at each of the stages.

Waterfall Model in IT Project Management (ITPM)

In IT project management, the Waterfall model is valuable because it supports strong planning and control. Since requirements are agreed early (as mentioned in the introduction), the project manager can build a clear scope baseline, develop a WBS, estimate time and cost, and assign resources with higher confidence. This makes Waterfall especially helpful when the organization needs predictable delivery and clear accountability.

Another key benefit is governance through phase-based deliverables. Each stage produces documented outputs (e.g., requirements documentation, design specifications, test plans, and deployment plans), which makes progress easier to track and report. In practice, this structure works like stage-gates: stakeholders review and approve deliverables before the project moves forward. For the project manager, this reduces ambiguity, improves communication, and creates an audit trail for decisions—useful in environments that require formal approvals or strict documentation.

Compared to iterative approaches (like Agile), Waterfall is often chosen when predictability and documentation are more important than flexibility. Agile performs better when requirements change frequently, but Waterfall can be more effective when change is limited, and the priority is delivering against a fixed plan. The main trade-off is that Waterfall is less flexible if requirements evolve after execution begins, which can lead to rework and schedule/cost impacts.

Stage 1: Requirements – The Discovery Phase

During this stage, project managers take a long time to gather requirements from the stakeholders (client, investors, team, management, etc..). They start by getting a clear idea or concept that the customer wants to do. They discuss the software problems that the system will resolve. Through this, they can deduce the functionality that the customer requires.

This stage needs to be done carefully, and all points should be thoroughly discussed, since the client will not be involved in the following stages. This makes it a crucial stage, as the software will only be shown to the client again once it is completed. If there is any confusion about the requirements or the functionality, and the customer is unsatisfied, the work will need to be redone starting from the beginning.

Once the requirements are finalized and approved, the Waterfall model proceeds in a fixed, sequential order. **Figure 1** provides an overview of these phases from requirements through maintenance.

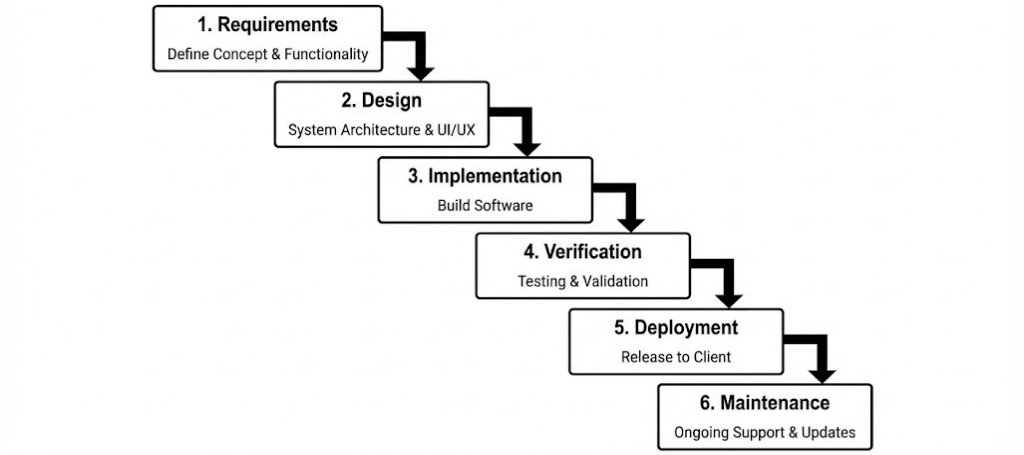

Figure 1:Waterfall model overview: sequential phases from requirements to maintenance.

Stage 2: Design

This stage includes the logical and physical design of the project. The team looks at the requirements, then make sure to design the software in such a way that it suits the customer's specifications and needs. Everything gathered in the requirements phase gets translated into how the system will function. The team thinks about what components they need and how they will interact. This creates the system architecture, which is the backbone of the project.

The design may be high level design or low-level design. The high-level design is like a blueprint of a house before picking the details, it shows the system design. Meanwhile, the low-level design looks at the details of system design, such as detailed logic and database tables. Next, they plan the UI/UX structure. Not as a full design yet, but it shows user flow diagrams, wireframes, and navigation mapping.

The last part of the design phase is the document creation, where everything is written in design documents, which are: the high-level design document, the low-level design document, the data dictionary, and the interface definitions. The team verifies the requirements before moving on to the programming phase.

Stage 3: Implementation

This phase cannot begin until the design and requirements phases are complete. This is where the real work gets done, as the team builds the design into actual software. Sometimes, the software is built in increments and later integrated as a whole. When one software unit is complete, work on the next unit begins.

In this stage, developers translate the design documents into working code by following the chosen architecture, coding standards, and technology stack. The work is usually divided into modules (for example: user interface, business logic, and database layer), then integrated once each unit is implemented. Unit testing and code reviews are commonly performed alongside coding to catch defects early and to ensure the implementation matches the design.

Typical deliverables of the implementation stage include:

- Source code and build artifacts
- Unit tests and unit test results

• Technical documentation (setup instructions, API/interface details)

Stage 4: Verification (Testing)

Verification ensures that the implemented system satisfies the requirements gathered in Stage 1. Testing activities are planned and executed to validate functionality, performance, usability, security, and compatibility. In the waterfall model, most testing happens after implementation is completed, which makes a clear and thorough test plan critical.

Common testing activities in this stage include:

• Integration testing to verify that modules work together correctly
• System testing to validate the complete system against the specification
• User Acceptance Testing (UAT) with stakeholders to confirm the system meets business needs
• Bug fixing and regression testing to confirm that changes do not break existing features

Stage 5: Deployment

Deployment is the process of delivering the verified system to the target environment (such as the client's servers or the cloud) and making it available to end users. This stage may include preparing production infrastructure, installing the software, migrating data, configuring security and access controls, and training users or support staff. A deployment plan is often prepared to reduce risks during go-live.

Typical deployment deliverables include:

• Release package and installation/configuration guide
• Deployment checklist and rollback plan
• User documentation and training material

Stage 6: Maintenance

After the system is deployed, maintenance begins. Maintenance covers correcting defects discovered in production, adapting the system to changes in the operating environment (for example, OS updates or new regulations), improving performance, and adding minor enhancements. In practice, maintenance can last longer than the initial development and requires proper documentation and change management.

Maintenance is commonly categorized into:

• Corrective maintenance: fixing faults and bugs
• Adaptive maintenance: adjusting to external changes (platforms, regulations, hardware)
• Perfective maintenance: improving features, usability, and performance
• Preventive maintenance: refactoring and updates to reduce future risks

Advantages of the Waterfall Model

The waterfall model can be effective when project requirements are stable and well understood. Key advantages include:

• Clear structure and well-defined milestones, making progress easy to track
• Strong documentation produced at each stage, which supports onboarding and maintenance
• Simple planning and management since each phase has specific deliverables
• Works well for smaller projects or regulated environments where formal approval is required

Disadvantages of the Waterfall Model

Despite its simplicity, waterfall has limitations, especially in projects with uncertainty or changing requirements. Key disadvantages include:

• Limited flexibility: changes after the requirements phase can be expensive and time-consuming
• Late feedback: users typically see the system near the end, which increases the risk of dissatisfaction
• Testing occurs late in the process, so critical issues may be discovered only after implementation
• Not ideal for complex or innovative projects where requirements evolve during development

Managing Waterfall Risks in IT Project Management

Even though the Waterfall model is strong in planning and control, its main risk is that problems can stay hidden until later phases. This happens because Waterfall progresses sequentially, so mistakes made early (especially in requirements) can affect everything that comes after. To manage this, a project manager should identify where risks typically concentrate and apply controls (reviews, approvals, and change management) at the right phases.

Figure 2 below highlights the phases where Waterfall projects commonly face the highest risk. The "hotspots" are usually **Requirements** (because unclear or incomplete requirements create misunderstandings) and **Testing** (because defect discovery may occur late if validation is delayed).

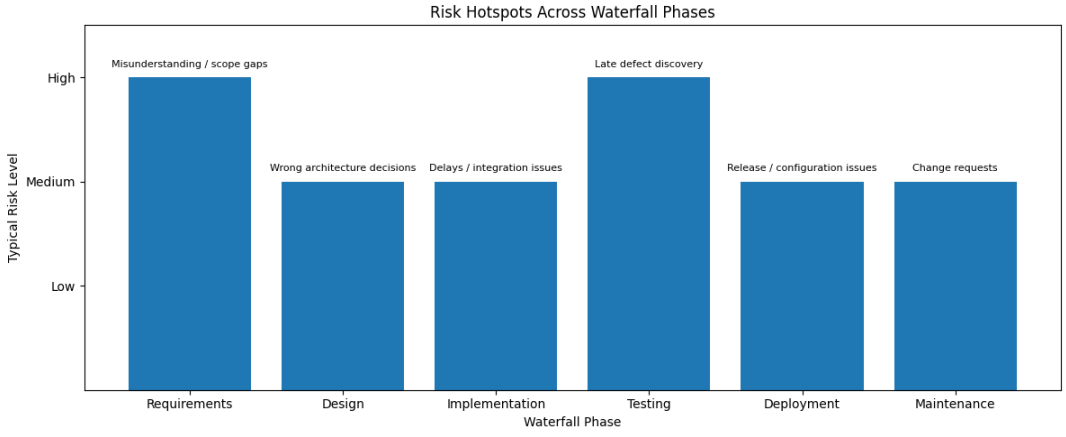

Figure 2 :Risk hotspots across the Waterfall phases (typical areas of risk concentration).

How to reduce the risks (practical controls)

Based on the hotspots shown in Figure 2, these actions help reduce Waterfall risk:

- **Requirements (High risk): prevent misunderstanding**
  - Run structured requirement workshops with stakeholders.
  - Write clear acceptance criteria for each requirement.
  - Get formal sign-off before moving to design.
  - Use simple prototypes/wireframes if needed to confirm understanding early.
- **Design (Medium risk): avoid wrong technical direction**
  - Conduct design reviews with senior/technical reviewers.
  - Validate the design against requirements (traceability).
- **Implementation (Medium risk): control delays and integration issues**
  - Break implementation into manageable modules with internal checkpoints.
  - Track progress using milestones and deliverables (not only time spent).
- **Testing (High risk): avoid late defect "surprises"**
  - Prepare a test plan early (during design), not at the end.
  - Add early verification steps (reviews + component testing) before full system testing.
- **Deployment (Medium risk): reduce release problems**
  - Use a deployment checklist and rollback plan.
  - Validate configuration and environment readiness before go-live.
- **Maintenance (Medium risk): manage change requests**
  - Apply a formal change control process (impact on scope/time/cost).
  - Prioritize fixes/enhancements and document updates.

Conclusion

The waterfall model provides a structured and document-driven approach to software development through its sequential stages: requirements, design, implementation, verification, deployment, and maintenance. It is a strong choice for projects with stable requirements and strict governance, but it can be risky for projects that need frequent iteration or rapid user feedback. Understanding the strengths and limitations of waterfall helps teams choose the most appropriate development methodology for each project.

References

1. Royce, W. W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON.
2. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.
3. Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill Education.
4. IEEE. (2014). ISO/IEC/IEEE 29148:2011 - Systems and software engineering - Life cycle processes - Requirements engineering.
5. Project Management Institute (PMI). (2021). *The Standard for Project Management and A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition*. Project Management Institute
6. ISO/IEC/IEEE. (2017). *ISO/IEC/IEEE 12207:2017 — Systems and software engineering — Software life cycle processes*. International Organization for Standardization.
7. IEEE. (2016). *IEEE Std 1012-2016 — IEEE Standard for System, Software, and Hardware Verification and Validation*. IEEE Standards Association.
8. Khoury, J. (2018, March 14). *Waterfall model definition and example* [Video]. YouTube.
9. Khreissat, A. *Waterfall Model* (IT Project Management lecture notes).