Higher Nationals – Summative Assignment Feedback Form

Student Name/ID	MOHAMMED MAHROOF MOHAMMED AASHIK/E230667		
Unit Title	Unit 7 : Software Development Life Cycle		
Assignment Number	1	**Assessor**	
Submission Date	24.12.2024	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	

Assessor Feedback:

LO1 Describe different software development lifecycles

Pass, Merit & Distinction Descripts	**P1** ☐	**P2** ☐	**M1** ☐	**D1** ☐				

LO2 Explain the importance of a feasibility study

Pass, Merit & Distinction Descripts	**P3** ☐	**P4** ☐	**M2** ☐	**D2** ☐				

LO3 Undertake a software development lifecycle.

Pass, Merit & Distinction Descripts	**P5** ☐	**P6** ☐	**M3** ☐	M4 ☐	D3 ☐			

LO4 Discuss the suitability of software behavioural design techniques.

Pass, Merit & Distinction Descripts	**P7** ☐	**M5** ☐	**M6** ☐	**D4** ☐				

Assessor Feedback

Grade:	**Assessor Signature:**	**Date:**
Resubmission Feedback:		
Grade:	**Assessor Signature:**	**Date:**
Internal Verifier's Comments:		
Signature & Date:		

* Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Important Points:

1. It is strictly prohibited to use textboxes to add texts in the assignments, except for the compulsory information. eg: Figures, tables of comparison etc. Adding text boxes in the body except for the before mentioned compulsory information will result in rejection of your work.
2. Avoid using page borders in your assignment body.
3. Carefully check the hand in date and the instructions given in the assignment. Late submissions will not be accepted.
4. Ensure that you give yourself enough time to complete the assignment by the due date.
5. Excuses of any nature will not be accepted for failure to hand in the work on time.
6. You must take responsibility for managing your own time effectively.
7. If you are unable to hand in your assignment on time and have valid reasons such as illness, you may apply (in writing) for an extension.
8. Failure to achieve at least PASS criteria will result in a REFERRAL grade.
9. Non-submission of work without valid reasons will lead to an automatic RE FERRAL. You will then be asked to complete an alternative assignment.
10. If you use other people's work or ideas in your assignment, reference them properly using HARVARD referencing system to avoid plagiarism. You have to provide both in-text citation and a reference list.
11. If you are proven to be guilty of plagiarism or any academic misconduct, your grade could be reduced to A REFERRAL or at worst you could be expelled from the course
12. Use word processing application spell check and grammar check function to help editing your assignment.
13. Use **footer function in the word processor to insert Your Name, Subject, Assignment No, and Page Number on each pag**e. This is useful if individual sheets become detached for any reason.

STUDENT ASSESSMENT SUBMISSION AND DECLARATION

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name:M.M.M.AASHIK	Assessor name:	
Issue date:	Submission date:24.12.2024	Submitted on: **24.12.2024**
Programme: HND in Computing		
Unit: 7 – Software Development Life Cycle		
Assignment number and title: 7 | Software Development Life Cycle		

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student signature: E230667 Date:21.12.2024

Assignment Brief

Student Name/ID Number	M.M.M AASHIK/E230667
Unit Number and Title	Unit7: Software Development Life Cycle
Academic Year	2024/2025
Unit Tutor	
Assignment Title	**Software Development Lifecycles on Large-scale Projects**
Issue Date	
Submission Date	24.12.2024
Submission Format	

The submission is in the form of the following.

A report exploring and highlighting key information on the use of software development lifecycles (SDLCs) in large-scale computer projects, and how the needs of an identified organisation can be met through the development of a business-related Application.
The recommended word limit is 1,500–2,000 words, although you will not be penalised for exceeding the total word limit.

A formal 10-minute presentation (10–20 slides as a guide, with speaker notes) to Communicate an evaluation of your investigation to a technical audience, highlighting key information regarding the planning methodology, tools and techniques used to design a solution to the given problem.

A design portfolio undertaking a software development lifecycle investigation for a Specified problem. The portfolio will include the following.

- Planning documentation produced to define and scope the problem and intended solution, including a requirements analysis and functional and non-functional

requirements.

- The recommended word limit is 2,000–2,500 words, although you will not be penalised for exceeding the total word limit.
- Solution design documents produced to meet the requirements of a given problem, Including logical, algorithmic and data designs. produced in a suitable format (e.g. Flowchart, pseudo code, DFDs).
  - The recommended word limit is 1,500–2,000 words, although you will not be penalised for exceeding the total word limit.
- Development documentation, including records of testing and review processes and Refinements made to the application throughout development.
  - The recommended word limit is 1,500–2,000 words, although you will not be penalised for exceeding the total word limit.
- Support documentation, including user and technical manuals.
  - The recommended word limit is 1,500–2,000 words, although you will not be penalised for exceeding the total word limit.
- Implemented application (final working version) in a format suitable to be run and Assessed for functionality.

Evaluative Report Produce an evaluative report that communicates an evaluation of your solution to your manager. The report should include analysis, discussion and the evaluation of the SDLC model chosen and followed.

You are required to make use of headings, paragraphs and subsections as appropriate, and all work must be supported with research and referenced using the Harvard referencing system.

Note: The report, Speaker notes, design portfolio, and evaluative report should be attached together and submitted as one copy.

Unit Learning Outcomes

LO1 Describe different software development lifecycles

LO2 Explain the importance of a feasibility study

LO3 Undertake a software development lifecycle.

LO4 Discuss the suitability of software behavioural design techniques.

Transferable skills and competencies developed

Computing-related cognitive skills

- Demonstrate knowledge and understanding of essential facts, concepts, principles and theories relating to computing and computer applications.
- Use such knowledge and understanding in the modelling and design of computer-based systems for the purposes of comprehension, communication, prediction and the understanding of trade-offs.
- Recognise and analyse criteria and specifications appropriate to specific problems, and plan strategies for their solutions.
- Analyse the extent to which a computer-based system meets the criteria defined for its current use and future development.
- Deploy appropriate theory, practices and tools for the design, implementation and evaluation of computer-based systems.

Computing-related practical skills

- The ability to evaluate systems in terms of quality attributes and possible trade-offs presented within the given problem.
- The ability to plan and manage projects to deliver computing systems within constraints of requirements, timescale and budget.
- The ability to recognise any risks and safety aspects that may be involved in the deployment of computing systems within a given context. The ability to deploy effectively the tools used for the construction and documentation of computer applications, with particular emphasis on understanding the whole process involved in the effective deployment of computers to solve practical problems.
- The ability to critically evaluate and analyse complex problems, including those with incomplete information, and devise appropriate solutions, within the constraints of a budget.

Generic skills for employability

- Intellectual skills: critical thinking; making a case; numeracy and literacy
- Self-management: self-awareness and reflection; goal setting and action planning, independence and adaptability; acting on initiative; innovation and creativity

- Contextual awareness, e.g. the ability to understand and meet the needs of individuals, business and the community, and to understand how workplaces and organisations are governed.

Learning Outcomes and Assessment Criteria

Pass	Merit	Distinction
LO1 : Describe different software development lifecycles		
P1: Describe two iterative and two sequential software lifecycle models. **P2**: Explain how risk is managed in software lifecycle models.	**M1**: Discuss using an example, why a particular lifecycle model is selected for a development environment.	**D1**: Assess the merits of applying the Waterfall lifecycle model to a large software development project.
LO2 : Explain the importance of a feasibility study.		
P3: Explain the purpose of a feasibility report. **P4**: Describe how technical solutions can be compared.	**M2:** Discuss the components of a feasibility report.	**D2**: Assess the impact of different feasibility criteria on a software investigation.
LO3 : 3 Undertake a software development lifecycle		
P5: Undertake a software investigation to meet a business need. **P6**: Use appropriate software analysis tools/techniques to carry out a software investigation and create supporting documentation.	**M3**: Analyse how software requirements can be traced throughout the software lifecycle. **M4**: Discuss two approaches to improving software quality.	**D3**: Evaluate the process of undertaking a systems investigation with regard to its effectiveness in improving a software quality.
LO4 : Discuss the suitability of software behavioural design techniques.		

Vocational scenario.

You work as a junior software developer at Phonyt Digital Solutions (PDS), an independent software development company that specializes in designing bespoke computer systems to meet client needs. Enomy-Finances is an organisation in the financial sector that provides advice and services related to mortgages, savings and investments. Enomy-Finances has recently seen an increase in demand for its services and is expanding to meet this demand. They would like a new computer system to support both staff and clients.

The project client – the Enomy-Finances Chief Technical Officer – has asked PDS if they would be interested in designing and implementing the new computer system. The client has specified that the new system must be deployed on an upgraded infrastructure. The CEO has asked you to carry out a preliminary software investigation on the new system to be developed prior to taking on the project. You are to identify the appropriate software methodology to be used should PDS agree to take on the project. You have also been asked to carry out a basic investigation into the project to determine scope, requirements and constraints, and to identify core system processes.

You are also required to generate some simple software designs to present to the project client to make sure that you have understood their requirements correctly. PDS has a range of in-house expertise using a range of software development paradigms – Event, Object Oriented, Procedural and Functional Programming. You have been given the freedom to select the most appropriate development approach and the appropriate project lifecycle methodology.

The requirements of the Enomy-Finances project are listed in Appendix 1. It contains further information about the application you are required to investigate. Ensure that you read the information carefully before you attempt this assignment. The current system is a networked application, but the project client has expressed an interest in possibly moving to a web-based system. The project client has agreed that, if required, they can be available throughout the project to answer any questions.

Assignment activity and guidance:

Activity 1 and 2– Report

Activity 1

Write a report that explores how the needs of Economy-Finance can be met, through the development of a computer application. Your report should include the following.

- A definition of the problem that needs to be solved, including user and system requirements that need to be met,
- An overview of different lifecycle models that could be applied to solving the problem. Your overview should contain, as a minimum, coverage of two iterative and two sequential software lifecycle models
- Consideration of different risks to this software project and to software lifecycle models in general, and how these risks can be managed.
- Recommendation of an appropriate SDLC model that could be implemented, supported by:
    - a discussion of the benefits and drawbacks of your chosen SDLC mode
    - Supported judgments as to the appropriateness of your chosen SDLC model for the Enomy-Finances project.
    - A discussion of which design and development tools would be suitable to implement the solution.
    - Supported judgments as to which design and development tools will be used and the preferred methodology undertaken.

Activity 2

Conduct research that explores the importance of feasibility studies in large-scale computer projects. Your research should include:

- consideration of the contents and purpose of a feasibility report, which includes:
    - an explanation of the purpose of a feasibility study
    - An overview of how different technical solutions, for an identified set of problems, can be compared.
    - A discussion of the different components of a feasibility report.
- An assessment of the impact, a feasibility study would have on the Enomy-Finances project, with reference to specific identified feasibility criteria.

You should support the points you make in the report with well-chosen examples from the information booklet and other relevant examples from any research you have carried out On related sectors or projects.

Activity 3 – presentation with speaker notes

Formal presentation covering the application of an appropriate software development approach to plan the implementation of the application for the Enomy-Finances project.

- SDLC model for the Enomy-Finances project.

As part of your planning, you should:

- identify stakeholder requirements,
- plan the scope of the module, including inputs, outputs, processes and process descriptors, consideration of alternate solutions and security considerations,
- explore relevant constraints,
- produce a set of functional and non-functional requirements,
-

Activity 4 – Design, Implementation and Maintenance portfolio

Design a solution that could be implemented, and which communicates effectively understanding of the program, including:

- o algorithmic software designs, for example, flowcharts, pseudocode
- o logical software design, including a finite state machine (FSM) and an extended FSM
- o data designs, e.g. DFDs, ERDs data dictionaries.

- You should ensure that you review your designs with others, recording the feedback received and actioning it to refine your proposed solution.

Use your improved designs to implement a functional business application to meet the requirements of the Enomy-Finances project. You should:

- interpret and implement a given design while remaining compliant with security and maintainability requirements
- apply an appropriate software development methodology/SDLC to manage the development process
- apply appropriate algorithms, logic and data structures
- build, manage and deploy code for the business application into a relevant environment and link code to data sets
- document the testing and refinement process, including creation of test plans and other quality assurance documents
- produce user and technical manuals to support the use and implementation of the application.

Activity 5 -Evaluative report

Produce an evaluative report that communicates an evaluation of your solution to your manager. The report should include the following.

- An analysis of how software requirements for the management were traced throughout the entire software lifecycle
- A discussion of at least two approaches to improving software quality within a software lifecycle
- Evaluation of the suitability of your solution in relation to the needs of Enomy-Finances, to include:
    - a discussion, using examples, of how the design and implementation of your solution meets the needs of the Enomy-Finances project
    - an analysis of the suitability of your chosen design and development tools and techniques for solving the given problem in comparison to other tools and techniques that could be used, ensuring coverage of algorithmic, logical and data designs
    - a critical review of undertaking a systems investigation which led to the design, development and testing of the solution and how it ensured improved reliability and effectiveness of the solution, including consideration of how risks were addressed
    - a critical review of how data driven software would improve the reliability and effectiveness of the solution
    - a review of the performance of the application against identified requirements.

- An assessment of how feedback was acted on to improve the solution, including justification of actions taken and any feedback that was not acted on.
- Recommendations for future improvements to the solution, including a supported rationale for these suggested improvements.

  You should support the points you make in the report with well-chosen examples from the information booklet, and other relevant examples from any research you have carried out on related sectors or projects.

Note: Reports, Speaker notes, design portfolio, and presentation slides should be attached together and submitted as one copy.

Recommended resources.

Textbooks

Dennis, A. and Haley, W. (2009) Systems Analysis and Design. John Wiley & Sons Ltd.

Lejk, M. and Deeks, D. (2002) An Introduction to System Analysis Techniques. 2nd Ed. Addison-Wesley.

Murch, R. (2012) The Software Development Lifecycle: A Complete Guide. Kindle.
Smart, J. F. (2014) BDD in Action: Behavior-driven development for the whole software lifecycle. Manning.

Web

www.freetutes.com-

FreeTutes
Systems Analysis and Design – Complete Introductory Tutorial for Software Engineering (Tutorial)

www.ijcsi.org-
IJCSI International Journal of Computer Science Vol. 7, Issue 5, September 2010 A Comparison Between Five Models Of Software Engineering (Research)

IJCSI International Journal of Computer Science Vol. 6, Issue 1, 2015 Software Development Life Cycle Models – Comparison, Consequences (Research)

P7: Discuss, using examples, the suitability of software behavioural design techniques.	**M5**: Analyse a range of software behavioural tools and techniques. **M6**: Differentiate between a finite state machine (FSM) and an extended FSM, providing an application of use for both.	**D4**: Present justifications of how data-driven software can improve the reliability and effectiveness of software.

Acknowledgment

I would like to express my heartfelt gratitude to everyone who supported and guided me throughout this project. Firstly, I would like to Thank My Lecture Mrs. Dimalsha for their valuable insights, constant encouragement, and unwavering support, which were instrumental in the successful completion of this project.

Special thanks to my peers and classmates for their collaborative spirit and for always being willing to share knowledge and feedback, which enriched the project significantly.

Lastly, I would like to thank my family and friends for their continuous encouragement and for always being a source of strength and motivation.

Thank you all for your invaluable contributions and support.

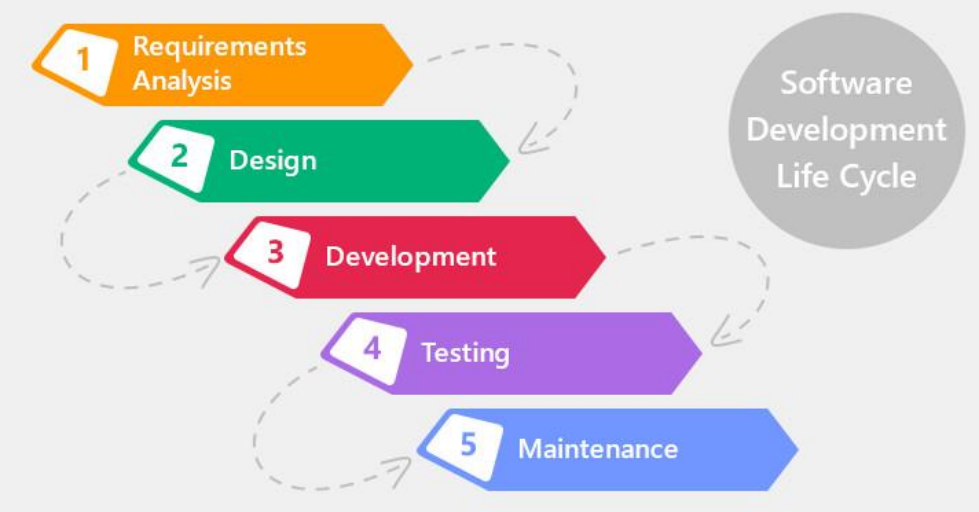
Activity 1

Software Development Life Cycle (SDLC) Models

A software life cycle model (also known as a process model) is a diagrammatic and graphical depiction of the software life cycle. A life cycle model represents all of the procedures necessary to move a software product through the stages of its life cycle. It also captures the structure in which these procedures will be carried out.

A life cycle model is a diagram that depicts the numerous actions that occur on a software product from its creation through its retirement. Different life cycle models may assign different development tasks to different periods. Thus, regardless of whatever life cycle model is used, the key activities are included in all life cycle models, even though the actions are carried out in various orders in different life cycle models. Multiple activities may be carried out at any point of the life cycle. (Anon., n.d.)

Stages of SDLC



The Systems Development Life Cycle (SDLC) is a framework that defines the phases involved in developing and delivering an information system. It provides a structured approach to planning, designing, developing, testing, and deploying information systems. The SDLC can help organizations to:

- Develop high-quality, cost-effective information systems
- Reduce the risk of project failure
- Improve communication and collaboration among project stakeholders
- Manage project scope and requirements
- Deliver information systems on time and within budget

- **Requirement Analysis**

  At this stage, software needs are identified through discussions among stakeholders to clarify usage, users, input requirements, and desired outcomes. This information is then analyzed to determine if the requirements can be integrated into the program being developed.

- **Design**

  The software and system design are created here in accordance with the instructions in the 'Requirement Specification' paper. The design step determines which hardware and system requirements are required, as well as the overall system architecture.

- **Implementation & Coding**

  The actual coding is done at this step, and the code is created based on the design specifications.

- **Testing**

  Following the construction of the code, it is tested to ensure that it fits all of the requirements established in the first step. Various types of testing are performed, including system testing, unit testing, acceptability testing, and integration testing.

- **Maintenance**

  This is the stage at which the completed software is provided to the customer. Once the user starts using the product, the genuine issues become apparent. These issues are handled on a regular basis as they arise. (Anon., n.d.)

Why Do We Use SDLC?

The Software Development Life Cycle (SDLC) is a structured approach used to create high-quality software in a well-organized and manageable way. There are several key reasons why SDLC is important and widely used in software projects:

1. High Level of Control

SDLC allows developers and project managers to have better control over the development process. Every stage of the project from planning to testing is clearly defined. This means teams can track progress and make sure the software meets the client's original needs. The design and testing stages are also done in an organized way, reducing the chances of unexpected problems during release.

2. Repeatable Process

One of the best things about using an SDLC model is that it can be repeated. If a project is completed successfully using one method (like Agile or Waterfall), the same steps can be followed for a similar future project. This repeatable structure makes it easier for teams to start new projects with more confidence, knowing what works. It also helps teams refine and improve the process every time.

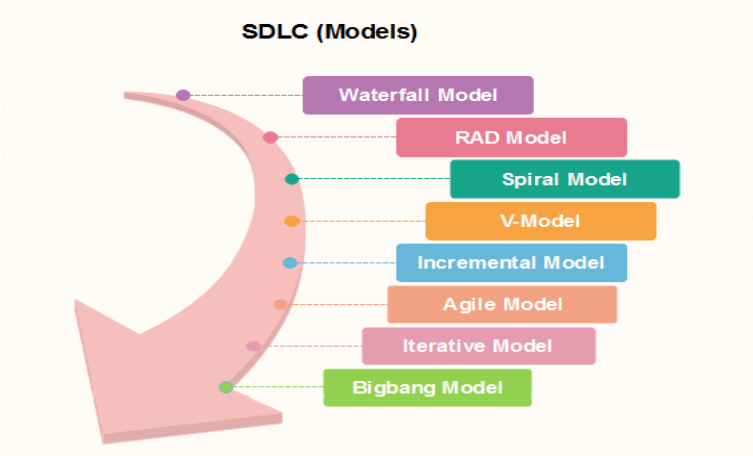
3. Increases Efficiency

SDLC breaks down the software development work into smaller, more manageable tasks. Each phase (such as planning, design, development, and testing) can be handled separately. This makes it easier to focus on specific goals and track progress. Developers can even work on different stages at the same time, which helps save time and increase productivity. Overall, the process becomes faster, more organized, and less stressful.

SDLC Model

Types of SDLC Models

The Software Development Life Cycle, or SDLC, is a method used to guide the process of creating software in an organized way. It helps teams plan, build, test, and maintain software systems step by step. Each SDLC model breaks the work into different stages like planning, analysis, design, development, testing, deployment, and maintenance. Depending on the project's size, goals, and deadlines, different models (like Waterfall, Agile, or Spiral) can be used. The main idea is to reduce risks and improve the quality of the final product by following a clear and structured path from start to finish.



(www.javatpoint.com, 2024)

1. Iterative Model

The iterative model is a flexible approach to software development where the project is built in small parts or cycles. Each cycle includes planning, design, coding, and testing. After each cycle, the team gets feedback,

usually from users or stakeholders, which helps guide the next version. This model is useful when the project's requirements are not fully clear at the beginning or are expected to change over time.

Key Points

- **Step-by-step Development:** The software is created and improved in several smaller versions rather than all at once.
- **Ongoing Feedback:** Users give feedback regularly, helping the team make changes and improvements early.
- **User Involvement:** Since users are part of the process from the start, the final product often meets their expectations better.
- **Handles Changes Well:** This model allows changes to be made easily during the development cycle.
- **Less Risk:** Any problems are found and solved early, which reduces the chance of major issues later.

2. Sequential Model

The sequential or waterfall model is a more traditional way of building software. It follows a straight-line process, where one step must be finished before moving on to the next. This model works well when the project's requirements are clear and unlikely to change.

Key Points

- **One Step at a Time:** Each stage like planning, designing, building, testing happens one after the other.
- **Needs Strong Planning:** All the details and requirements must be clearly understood before development starts.
- **Testing Comes Later:** Testing is done after the system is fully built, which helps find issues before launch.
- **Detailed Records:** This model involves creating a lot of documents that explain each part of the system.

- **Good for Simple Projects:** It's a reliable approach for projects where things are unlikely to change once they're planned.

Iterative Model

The iterative model is somewhat like the waterfall model, but there is one major difference. In the waterfall model, everything is planned in full detail at the very beginning this includes all the requirements, designs, and features. Only after everything is clearly planned out does the actual development start.

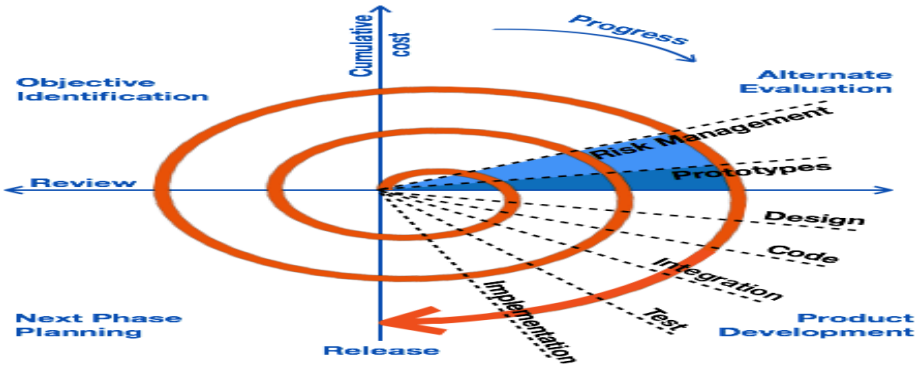
In contrast, the iterative model breaks the development process into smaller cycles or "iterations." Each cycle focuses on a few features at a time rather than the whole system. This makes it more flexible because the team doesn't need to know every single requirement before they start building the application. Instead, they can begin with the most important or basic features, and as the project moves forward, more features or improvements can be added based on feedback and new needs.

Each iteration usually lasts between two to six weeks. At the end of every cycle, the new features are added to the parts already developed. Over time, these pieces come together to form the complete system.

For example, imagine an application has ten main features. In the first iteration, the team might work only on three of them. They go through all the usual software development steps for those three like gathering requirements, designing, coding, testing, and deploying. Once that's done, they move on to the next set of features, continuing this cycle until everything is built.

This model allows teams to deliver working parts of the application early and update or improve the system based on feedback during each step.

- **Spiral model**



The spiral model blends the notion of repeated development with the waterfall model's methodical, regulated characteristics. This spiral model combines the iterative development process model and the sequential linear development model.

Spiral Model Design

The Spiral Model is a software development approach that consists of four stages, iterated through multiple spirals

- **Identification**

  This phase focuses on gathering business requirements and understanding system needs. It involves continuous interaction between the client and system analysts. At the end of each spiral, the product is released to the market.

- **Design**

  The design process starts with conceptual design and evolves through architectural, logical, and physical design stages across successive spirals, gradually refining the product.

- **Construct or Build**

  Each spiral involves building a version of the product. In the initial spiral, a Proof of Concept (POC) is created for user feedback. In later spirals, a fully functional version of the software is built and reviewed by the client.

- **Evaluation and Risk Analysis**

This phase focuses on identifying and assessing risks, including technical feasibility and project management risks like schedule delays and cost overruns. After each iteration, the software is tested, reviewed by the client, and adjusted based on feedback.

Spiral Model Suitability

- When there is a financial restriction and risk assessment is critical.
- For projects with a medium to high level of risk.
- Long-term project commitment due to the possibility of changes in economic priorities as requirements change over time.

Advantages and Disadvantages of Spiral Model

Advantage	Disadvantage
▪ Changing needs can be addressed. ▪ Prototypes can be used extensively. ▪ More precise requirements can be captured. ▪ Users first notice the system. ▪ Development may be broken into smaller portions, and dangerous aspects can be created early, allowing for better risk management.	▪ Management is becoming increasingly difficult. ▪ The project's conclusion may not be known for some time. ▪ tiny or low-risk initiatives are not suited, and tiny projects may be costly. ▪ The procedure is complicated. ▪ The spiral might carry on indefinitely. ▪ A large number of intermediate steps need an abundance of documentation.

Risk Management

The waterfall and iterative approaches are combined in this model. Each stage of the process incorporates risk management, and the process cycles back on itself to handle risks and uncertainties. Identifying possible risks, analyzing their likelihood and effect, devising and testing mitigation methods, and monitoring those tactics as the project proceeds are all part of the risk management strategy.

Agile Model.

Agile is a modern software development method that focuses on building a project step by step through small, manageable parts called iterations or sprints. These sprints usually last between 1 to 3 weeks. Agile encourages constant teamwork between developers, testers, and stakeholders, helping the project stay flexible and aligned with changing user needs. One of the main ideas behind Agile is that requirements can change, and the development process should be able to adapt quickly without causing delays.

Key Features

- **Sprints:** The project is broken into short cycles, and after each sprint, a working version of the software is delivered.
- **Flexibility:** Changes in user needs or project direction can be easily managed and added to future sprints.
- **Collaboration:** Regular communication with clients and team members ensures that everyone is on the same page.
- **Iterative Feedback:** After each sprint, feedback is gathered and used to improve the next stage of development.

Advantages

- **Fast Results:** Clients can see progress quickly and start using parts of the system even before the whole project is done.
- **Client Involvement:** Regular client interaction helps ensure the software meets their real needs and expectations.
- **Flexible to Changes:** Agile can handle changes to requirements or priorities without disrupting the entire project.

Disadvantages

- **High Client Involvement Needed:** Clients must be available throughout the project for reviews and feedback, which might not always be possible.

- **Hard to Predict Time and Cost:** Since things can change often, it's difficult to estimate the final timeline and budget accurately.

Risk Management

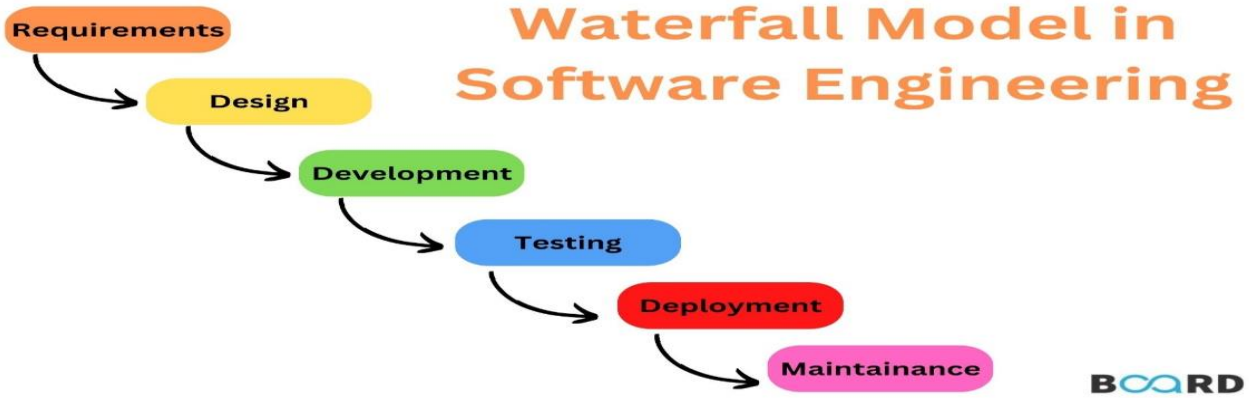
Agile helps reduce risks by delivering a working version of the software at the end of each sprint. This makes it easier to catch and fix issues early. If a feature doesn't work properly, it can be improved or removed in the next sprint. Regular involvement from users and stakeholders also helps spot problems before they become major risks.

Sequential SDLC Model

The spiral model is a software development approach that combines ideas from both the waterfall and iterative models, but with a strong focus on identifying and managing risks. One of the main challenges in this model is deciding the right time to move from one phase to the next. To handle this, teams usually set fixed timeframes for each phase. These timeframes are often based on experience from previous projects or expert judgment. Even if a phase isn't fully finished, the project can move forward according to the set schedule.

In this model, the overall development is divided into smaller parts or cycles. If a major risk appears during any cycle, that phase can be shortened or ended early to concentrate on solving the risk. This helps avoid delays and ensures that potential issues are tackled before they grow. By repeating this cycle and continuously checking for problems, the spiral model makes it easier to deliver a safer and more reliable software product.

Waterfall model



Waterfall Model is a linear-sequential life cycle model. It is incredibly simple to grasp and apply. In a waterfall model, each step must be finished before the next phase can begin, and the stages must not overlap. Typically, the conclusion of one step serves as the input for the following phase in this Waterfall approach.

- **Requirement Gathering and analysis**

  During this phase, all potential system needs are identified and recorded in a requirement specification document.

- **System Design**

  This phase studies the need specifications from the previous phase and prepares the system design. This system design aids in the specification of hardware and system requirements, as well as the definition of the overall system architecture.

- **Implementation**

  The system is first built in discrete programs called units, with input from the system design, and then combined in the following step. Unit testing is the process of developing and testing each unit for functioning.

- **Integration and Testing**

  Following unit testing, all units generated during the implementation phase are integrated into a system. Following integration, the complete system is tested for flaws and failures.

- **Deployment of system**

Following the completion of functional and non-functional testing, the product is deployed in the client environment or launched to the market.

- **Maintenance**

  There are a few difficulties that arise in the client environment. Patches are published to address these vulnerabilities. In order to improve the product, newer versions are published. Maintenance is performed in order to implement these modifications in the client environment.

Waterfall Model Suitability

When the requirements are very well documented, clear, and fixed, the product definition is stable, the technology is understood and is not dynamic, there are no ambiguous requirements, ample resources with required expertise are available to support the product, and the project is short, the Waterfall model is most appropriate

Advantages and Disadvantages of the Waterfall Model

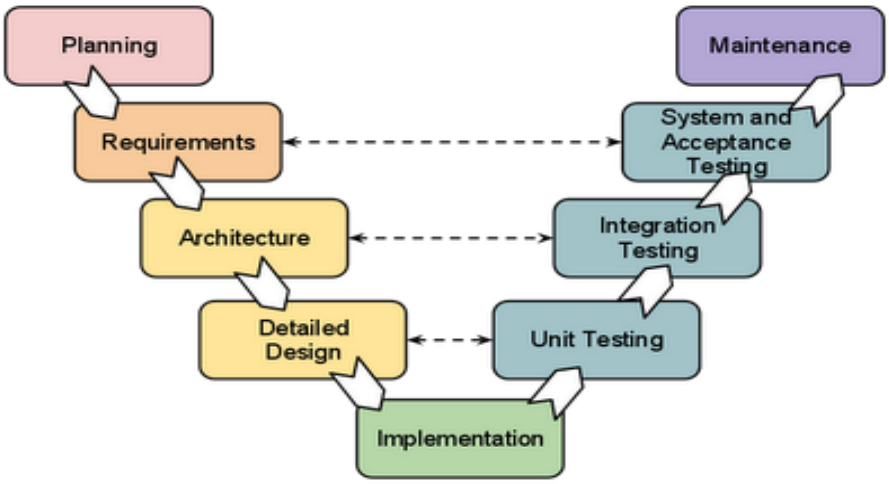
Advantage	Disadvantage
▪ simple to comprehend and apply ▪ Because of the model's rigidity, it is simple to manage. Each phase includes its own set of deliverables and a review procedure. ▪ One phase at a time is processed and completed. ▪ Works effectively for smaller projects with well-defined needs. ▪ The process and outcomes are thoroughly documented.	▪ Until late in the life cycle, no functioning software is developed. ▪ There is a lot of danger and uncertainty. ▪ Not suitable for complex, object-oriented programs. ▪ Poor model for long-term projects. ▪ Not appropriate for projects with changing needs. As a result, the risk and uncertainty associated with this process model are significant. ▪ Cannot meet changing needs.

Risk Management

In the Waterfall model, risk management focuses on identifying potential risks early in the project, assessing their impact and likelihood, and creating mitigation strategies before moving to subsequent phases. Since Waterfall follows a sequential approach, changes are difficult to implement once a phase is completed, making thorough planning and risk analysis crucial. Contingency plans are developed for unforeseen risks, and risks are continuously monitored, documented, and communicated to ensure the project stays on track and issues are addressed in a timely manner.

V-Shaped Model

It is an extension of the waterfall model, Instead of moving down in a linear way, the process steps are bent upwards after the implementation and coding phase, to form the typical V shape. The major difference between the V-shaped model and the waterfall model is the early test planning in the V-shaped model.



The usage

- Software requirements clearly defined and known
- Software development technologies and tools are well-known

Advantages and Disadvantages

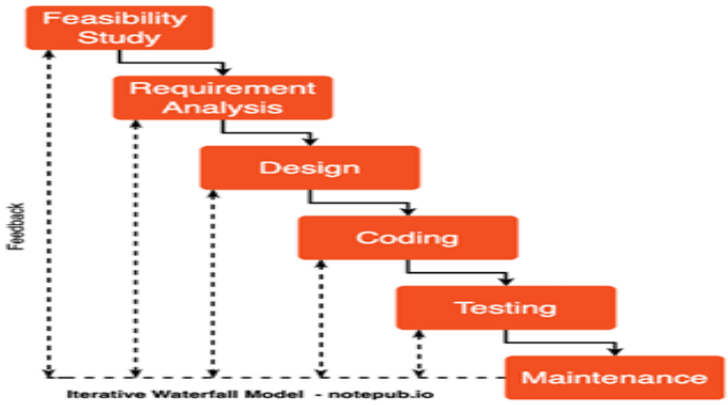
Advantages	Disadvantages

□ It is simple and easy to understand. □ Every phase produces clear results or deliverables. □ Has better chances of success compared to the waterfall model because test plans are made early. □ Works good when the requirements are clear and easy to know. □ The product is checked and tested in the early stages of development.	□ This model is quite rigid, similar to the waterfall approach. □ Changing the project scope is hard and can be costly. □ Since the software is only built during the implementation stage, there are no early versions or prototypes available. □ It does not offer a clear way to handle issues discovered during testing. □ Overall, it can take more time and be expensive because it requires very detailed planning from the start.

Risk Management

Emphasizes verification and validation at each stage to catch risks early. Testing occurs simultaneously with development to detect issues early. Structured approach limits flexibility in addressing unforeseen risks once a phase is complete. During the development of the savings module, testing uncovers a critical error in calculations, allowing early correction before implementation. (Sami, 2012)

- **Iterative Waterfall Model**



The Iterative Waterfall model closely resembles the traditional Waterfall model but incorporates feedback loops, making it a more realistic approach. It anticipates the likelihood of discovering issues during the testing phase, necessitating a return to either the design or requirement analysis phases for rectification.

The key aim is to identify and address errors within the same phase to minimize the cost of correction. Delaying issue identification leads to increased expenses, which is the fundamental concept behind Phase error containment.

Iterative Waterfall model suitability.

- The iterative waterfall model can be appealing for organizations that value a structured and systematic approach to development, testing, and documentation.
- If the project's requirements are well-defined and stable.
- For smaller to moderately sized projects with limited complexity.

Advantages and Disadvantages of the Iterative Model

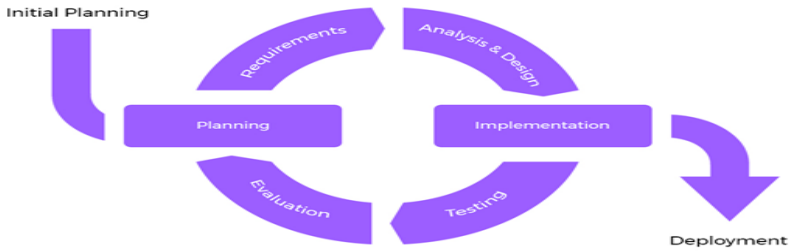
Advantage	Disadvantage
Easy to understand, easy to use. Provides a reference for inexperienced staff. Milestones are well understood by the team. It provides requirements stability. Facilitates strong management controls.	▪ All requirements must be known upfront. ▪ Deliverables created for each phase are considered frozen. ▪ It can give a false impression of progress as there is no output until the project's completion. ▪ Integration is one big bang at the end.

	▪ Little opportunity for the customer to preview the system.

Risk Management in Iterative Waterfall model

Risk management in the iterative waterfall model involves identifying risks at the project's outset and continuously assessing and addressing them throughout development cycles. This approach allows for adjustments in mitigation strategies based on feedback from each iteration, thereby reducing the likelihood of significant issues arising later in the project.

- **Iterative model**



An iterative life cycle model does not seek to begin with a complete statement of needs. Instead, development begins by describing and developing only a portion of the program, which is then examined to discover additional needs. This procedure is then repeated, resulting in a new version of the program at the conclusion of each iteration of the model.

Iterative model design

Iterative Model Suitability

The iterative software development module focuses on requirements, design, implementation, and testing in cycles. Each release builds on the previous one until it meets all requirements. Success hinges on thorough requirement validation and testing for each version, with tests evolving as the software progresses.

In the software industry, iterative and incremental development have specialized uses. This model is most commonly used in the cases listed below.

- The system's requirements are well stated and understood.
- Although major needs must be established, some functionality or suggested additions may develop over time.
- There is a time limitation in the market.
- While working on the project, the development team is using and learning a new technology.
- Resources with the required skill sets are unavailable and will be employed on a contract basis for certain iterations.
- Some features and aims are high-risk and may change in the future

Advantages and Disadvantages of the Iterative Model

Advantage	Disadvantage
▪ Some functionalities may be created fast and early in the life cycle. ▪ Early and regular results are obtained. ▪ Parallel development is possible. ▪ Progress may be quantified. ▪ It is less expensive to adjust the scope/requirements. ▪ It is simple to test and troubleshoot during lesser iterations. ▪ During iteration, risks are recognized and handled, and each iteration is a manageable milestone. ▪ Risk management is simplified since the high-risk portion is completed first. ▪ Every increment delivers operational product. ▪ It accommodates shifting requirements. ▪ Suitable for Larger and more mission-critical projects	▪ Additional resources may be necessary. ▪ Although the cost of change is lower, it is not well suited to changing requirements. ▪ More managerial involvement is necessary. ▪ Defining increments may need the specification of the entire system. ▪ Not appropriate for tiny tasks. ▪ Management complexity is increasing. ▪ The end of the project may not be known, which poses a danger. ▪ Risk analysis necessitates the use of highly qualified resources. ▪ The risk analysis step is critical to project development.

Risk Management

This methodology promotes iterative development and ongoing feedback. Each iteration incorporates risk management, and the process is adaptable to changing objectives and priorities. Identifying and analyzing risks, taking action to minimize or resolve them, and monitoring and changing the risk management plan as the project advances are all part of the risk management strategy. This technique enables the development team to handle hazards as they arise and alter the development process to mitigate their impact.

Introduction

Enomy-Finances is planning to grow its services because more people are using them, but their old IT system is causing delays, especially when handling currency conversion and investments. This slowdown is affecting how smoothly the business runs and how satisfied the customers are. To fix these problems, they want to create a new web-based system that can manage data safely and provide real-time updates.

Phony Digital Solutions (PDS), a software company, has been asked to study different software development methods and recommend the best SDLC model that will meet both the company's security needs and business goals. As a junior developer at PDS, I have been assigned to do a first investigation that includes figuring out the project scope, requirements, and main processes. This will help design a clear plan for building a flexible, secure, and effective system to support Enomy-Finances' growth.

Report for Economy-Finances Client Project

Define the problem.

Enomy-Finances is a financial company facing more demand for its services, which means they need to grow their operations. Because of this, they need a new and better computer system to help both their employees and customers. Right now, they use a network-based system, but they want to switch to a web-based system so it's easier to access and can handle more users as the business grows.

The main problem is to design and build this new system that fits their expanding needs. It should run on updated technology and meet all important functional and non-functional needs. The difficult part is choosing the right software development method, understanding the project's scope, finding all the system requirements and limits, and making sure the key processes of the system support Enomy-Finances' business goals.

Key challenges

Key challenges for this project include several important points. First is picking the right software development method, like Waterfall, Agile, or Spiral, that will help finish the project smoothly, on time, and within budget. Next, it's important to clearly figure out what the system needs to do both the functional parts and other requirements like whether to keep using a networked system or move to a web-based platform.

Another challenge is understanding and defining the main tasks the system must perform so it works well for both employees and customers. There are also limits to consider, such as how much time and money are available, the current technology setup, and what users expect, which all affect how practical the project is.

Choosing the right software development style, like Event-Driven, Object-Oriented, Procedural, or Functional programming, is also important to fit the project's needs. Overall, the goal is to build a system that makes the company work more efficiently, supports its growth, and offers an easy and smooth experience for staff and clients. The system also needs to be flexible and able to grow with future demands.

SRS - System Requirement Specification for Enomy-Finances System

Introduction

Enomy-Finances seeks to develop a new computer system to support both staff and clients as part of its expansion strategy. The system will handle financial services, such as mortgages, savings, and investments, and must be deployed on upgraded infrastructure, potentially as a web-based solution.

System Name: Economy-Finances System

Purpose

The Software Requirements Specification (SRS) document defines the functional and non-functional requirements for the development of the Enomy-Finances System. This document serves as a roadmap for the system's design, implementation, and testing.

System Scope

The Enomy-Finances System is designed to streamline financial processes by automating administrative tasks, managing currency conversions, and enabling investment tracking. The system will include modules for financial transactions, savings management, currency conversion, and reporting tools for better decision-making.

The project will include these main features

- Real-time currency conversion for several currencies like GBP, USD, EUR, BRL, and others.
- Customized investment management, showing savings and investment quotes, along with tracking how they perform over time.
- Safe storage of user information, including their personal contact details and transaction records.
- An easy-to-use interface designed for both employees and customers.
- Strong security measures to protect user data, making sure it stays confidential, accurate, and always available.

An Overview of the Document

This document presents an in-depth description of the Enomy-Finances System, including its features, user classes, system constraints, and non-functional requirements.

General Description

Product Perspective

The Enomy-Finances System is a comprehensive financial management platform that improves operational

efficiency and enhances financial decision-making. It will serve as a central tool for managing financial transactions, recording currency exchanges, monitoring investments, and generating financial reports.

User Characteristics

- **Financial Analysts**
- **Accountants**
- **Administrators**
- **IT Support Staff**

System Goals

- Streamline financial processes for accuracy and efficiency.
- Enhance transparency in currency exchange and investment tracking.
- Facilitate real-time reporting and analytics for informed decision-making.
- Provide a user-friendly platform for financial operations.

Project Constraints

1. **Time Constraints**
   The system has to be built and completed within a set timeframe. This is important so that the business can launch the platform on time and meet its internal or market deadlines. Delays in development could affect business goals.
2. **Budget Constraints**
   The entire project, including planning, designing, coding, and launching the system, must be done without going over the set budget. All team members must work within financial limits to avoid extra costs or resource issues.
3. **Infrastructure Constraints**
   The system must run on the hardware and software environment already set up or upgraded by the client. This means developers have to work with specific tools and technologies that may have performance or compatibility limits.
4. **Scalability**
   The system needs to be flexible and able to grow with the business. As the company gains more

users and handles more financial transactions, the system should be able to manage the extra workload without crashing or slowing down.

System Constraints

1. **Security**

   The system must follow high-level security standards. All stored data and information shared over the internet must be protected using strong encryption methods. This ensures client data is safe from hackers or breaches.

2. **Interoperability**

   The system must work well with the company's current databases and external sources like currency rate providers. It should be able to exchange and process information from different systems without issues.

3. **Platform**

   The system must be a web-based application. It should work properly on all major web browsers (like Chrome, Firefox, and Safari) and also have a responsive layout so users can access it easily on phones and tablets.

4. **User Access Control**

   Different users must have different levels of access. Clients, staff, and administrators should only see and use features meant for their roles. This protects sensitive data and helps with task management and accountability.

Requirements

The requirements of a software project are the functions, features, and limitations that the final product must meet. In other words, the requirements specify what the program should perform, how it should appear, and any criteria that must be satisfied for it to be regarded successful. (visuresolutions, 2024)

Types of requirements

1) **System Requirements**

   System requirements may be thought of as an enlarged form of user needs. System requirements are the starting point for each new system design. These criteria provide a clear explanation of the user needs that the system must meet.

2) **User requirements**

- **Functional requirements**

  Functional requirements outline the system's expected functionalities, detailing how it will operate to fulfill user needs and respond to commands, along with its features.

- **Nonfunctional requirements**

  Non-Functional Requirements describe the system's restrictions and constraints. These requirements have no bearing on the application's operation.

System Requirements for Enomy-Finances Organization

Hardware Needs

- A strong and reliable server is needed, with enough processing power and memory to handle all system tasks smoothly.
- Backup devices and storage systems must be in place to keep data safe and help with recovery if something goes wrong.
- Staff and clients will use desktops, laptops, or mobile devices, so these must be capable of running the system's user interface without issues.

2. Software Needs

- **Operating System:** The system should work well with current operating systems like Windows or Linux, depending on what the infrastructure supports.
- **Database:** A relational database like MySQL, PostgreSQL, or Microsoft SQL Server will be needed to keep track of user info and transaction records.
- **Web Server (for web systems):** Servers like Apache or Nginx will be required to host and run the application online.
- **Frontend Tools:** For the user interface, a modern frontend library or framework such as React, Vue.js, or Angular will be useful for building responsive pages.
- **Backend Tools:** A backend framework such as Node.js, Django, or Ruby on Rails should be selected depending on the structure and language preferred.
- **Security Features:** To protect the system, it must include SSL/TLS for safe data transfer, two-step login methods (MFA) for user access, and encryption methods to keep stored data private.

Functional Requirements (What the system should do)

1. **Manage Staff Profiles**
   The system should store and manage important details like contact info, login credentials, and job roles for financial advisors and admin staff.
2. **Client Access to Financial Tools**
   Clients should be able to log in to use tools that help with financial planning. They should also be able to check their investment portfolios and get online quotes for savings or investments.
3. **Handle Financial Transactions**
   The system must process payments like currency conversions, savings deposits, and investment purchases. It should support multiple secure payment methods such as credit cards, bank transfers, and e-wallets.
4. **Track Client Interactions**
   All client-related activities, including meetings, chats, past transactions, and saved financial plans, must be recorded and kept for future reference.
5. **Register New Clients**
   When a new client signs up, the system should collect their personal and financial details, save the information, and assign them a unique client ID.
6. **Manage Financial Plans**
   The system should allow staff to create and manage clients' financial plans, including savings goals, investment options, and custom portfolio suggestions based on each client's needs.

Nonfunctional Requirements (How the system should behave)

1. **Strong Security Measures**
   The system must have strong data encryption, login authentication, and user access controls to protect private financial and personal data.

2. **Easy-to-Use Interface**

   The design of the platform should be simple and easy to use, so even clients who aren't tech-savvy can navigate it without problems.

3. **Advanced Reporting Features**

   The system should be able to produce detailed financial reports for users. These should include summaries of transactions, performance of investments, and analysis of savings accounts.

4. **High System Reliability**

   The platform must be stable and available almost all the time, with very little downtime, so clients and staff can always access their accounts and services when needed.

Software lifecycle Models

Iterative SDLC Model

The iterative model is somewhat like the waterfall model, but there is one major difference. In the waterfall model, everything is planned in full detail at the very beginning this includes all the requirements, designs, and features. Only after everything is clearly planned out does the actual development start.

In contrast, the iterative model breaks the development process into smaller cycles or "iterations." Each cycle focuses on a few features at a time rather than the whole system. This makes it more flexible because the team doesn't need to know every single requirement before they start building the application. Instead, they can begin with the most important or basic features, and as the project moves forward, more features or improvements can be added based on feedback and new needs.

Each iteration usually lasts between two to six weeks. At the end of every cycle, the new features are added to the parts already developed. Over time, these pieces come together to form the complete system.

For example, imagine an application has ten main features. In the first iteration, the team might work only on three of them. They go through all the usual software development steps for those three like gathering

requirements, designing, coding, testing, and deploying. Once that's done, they move on to the next set of features, continuing this cycle until everything is built.

This model allows teams to deliver working parts of the application early and update or improve the system based on feedback during each step.

- **Spiral model**

The spiral model blends the notion of repeated development with the waterfall model's methodical, regulated characteristics. This spiral model combines the iterative development process model and the sequential linear development model.

Key Features

- **Risk Checking:** Every stage of the project includes a risk check, which helps to spot possible problems early so they can be fixed or avoided.
- **Divided Phases:** The whole project is split into clear parts like planning, risk review, development (engineering), and testing (evaluation). Each phase leads to the next one smoothly.
- **Repeating Cycles:** The model uses repeating cycles, meaning teams can return to earlier phases to improve or adjust if something changes or goes wrong.

How This Fits the Enomy-Finances System

The Spiral model fits well for building the Enomy-Finances system, especially since the system involves many services like savings, loans, and investments. These services can bring unexpected challenges during development, so having a flexible and risk-aware approach is helpful.

- **Handling Risks:** Because this system will deal with private financial data, safety and following laws (like data protection rules) are very important. The Spiral model makes sure that risks like these are reviewed often and kept under control.
- **Testing Early Features:** Early versions (prototypes) of main features, such as logging in or processing money transfers, can be created and tested. This lets the team and the client see how things work before full development. Their opinions can help shape the next steps.
- **Keep Improving:** With each new spiral (cycle), the project can improve based on what users want and what problems arise. This helps make sure the system that gets delivered is exactly what Enomy-Finances needs even if those needs change along the way.

Agile Model

The Agile approach focuses on creating software in small, manageable parts, with each part called an iteration. Typically, iterations last between 1 and 3 weeks and involve constant collaboration between the development and testing teams. Agile acknowledges that project requirements and priorities can change as work progresses, so it promotes flexibility and continuous delivery of working software.

Key Features

- **Sprints:** The development process is broken down into small, focused cycles called sprints (usually lasting 1 to 4 weeks). A working version of the software is delivered at the end of each sprint.
- **Flexibility:** The Agile method allows for changes in the project as requirements evolve based on feedback and shifting user needs.
- **Collaboration:** Regular communication between the development team and stakeholders ensures that everyone stays aligned and informed.

- **Iterative Feedback:** After each sprint, feedback from stakeholders is gathered, helping guide the direction of the next sprint.

How This Applies to the Enomy-Finances System

Since Enomy-Finances is growing to meet rising demand, the Agile model is a good fit for its system development. The flexibility Agile offers means that the system can easily adjust to changing business needs and feedback from users. For example, the client may ask for certain features at first, but after several iterations, feedback could lead to changes or additions to those features.

- **Development Process:** The system can be divided into smaller functional parts, like managing clients, processing transactions, and generating reports. These features can be developed step by step, improving with each sprint.
- **Stakeholder Involvement:** The client and end-users can take part in regular reviews, ensuring that the system is always in line with what they expect.
- **Feedback Loops:** With regular feedback from users, the development team can reassess priorities and modify features to better meet needs, making sure the final system fulfills all the expectations.

Sequential SDLC Model

Waterfall Model

The Waterfall model is a traditional software development approach where the project is completed in a sequence of fixed stages. These stages include requirement gathering, design, building the system, testing, and

finally, maintenance. Each step must be fully done before the next one starts, and the output of one phase becomes the input of the next. At every stage, checks are done to make sure everything matches the plan and project goals.

Main Features

- **Step-by-step Phases:** The process flows through specific steps such as gathering requirements, designing the system, writing the code, testing it, and maintaining the final product.
- **Fixed Order:** No step is skipped or repeated. One must finish before the next begins, meaning there's no going back once a stage is completed.
- **Detailed Documents:** Every phase comes with full documentation, which helps track progress and supports understanding for all team members.

How It Can Be Used for Enomy-Finances System

For a system like Enomy-Finances, which has clear objectives like handling mortgages, savings, and investment features, the Waterfall model might work well especially if the client's needs don't change much during development. This method allows for everything to be planned in advance and built with stability in mind.

- **Requirement Collection:** At the beginning, detailed talks with the client can make sure that all necessary functions and expectations are properly recorded.
- **Design and Build:** After getting the requirements, the team can design the system's structure and features without going back to the previous steps.
- **Testing and Launching:** When the system is ready, testing is done to find and fix issues before making it available for use.
-

V-Model (Verification and Validation)

The V-Model is a structured development method where every development activity is paired with a matching testing task. This means while the system is being designed or coded, related tests are also being planned or carried out. The name "V-Model" comes from the shape made when showing how each phase on the development side connects to a testing phase on the other side. The purpose is to catch problems early and make sure the product is both built correctly and does what the user wants.

Main Features

- **Side-by-side Phases:** Each development stage (like planning or coding) has a test stage (like unit or system testing) that happens along with it.
- **Verification vs Validation:** Verification is checking whether the product is being built right as per the plan. Validation is checking whether the right product is being built as per what the user actually wants.
- **Well-Organised:** Similar to the Waterfall approach, it moves in a fixed order but with more testing involved early on, reducing the risk of big issues later.

How It Fits the Enomy-Finances System

The V-Model would work well for a system like Enomy-Finances, where features like savings, mortgage handling, and investment tracking have clear goals. If the client already has a fixed idea of what they want and the requirements are not likely to change too much, this model helps the team stay on track.

- **Verification Stage:** During the early stages like system design, the development team can cross-check whether the planned structure meets the initial needs before writing any code.
- **Validation Process:** While building each module (like savings management or transaction history), the team can carry out tests such as unit and integration testing to make sure everything is working as it should.
- **Ongoing Testing:** Since testing happens alongside every stage of building, any issues can be spotted and fixed early. This helps prevent bigger problems from showing up at the final stage.

Risk Consideration and Management

Risk management is a systematic approach to identifying, assessing, prioritizing, and mitigating potential risks that can impact a project's objectives, outcomes, or resources. It involves a proactive approach to anticipating potential problems and implementing strategies to minimize their likelihood or impact.

(www.invensislearning.com, 2024)

Why is Risk Management Important in Software Development?

Software development projects are inherently complex and involve numerous factors that can contribute to risks. These risks can range from technical challenges and resource constraints to changing requirements and market uncertainties. Effective risk management helps organizations navigate these complexities and increase the probability of project success.

Key Objectives of Risk Management in SDLC

The primary objectives of risk management in SDLC are to

- Identify Potential Risks: Proactively identify potential risks that could hinder the project's progress, impact its quality, or threaten its success.

- Assess Risk Impact: Evaluate the likelihood and potential impact of each identified risk to prioritize mitigation efforts.

- Develop Risk Mitigation Strategies: Develop and implement strategies to minimize the likelihood or impact of identified risks.

Risk Management Process in SDLC

The risk management process in SDLC typically involves the following steps

- Risk Identification: Identify potential risks throughout the SDLC phases, including requirements gathering, design, development, testing, deployment, and maintenance.

- Risk Assessment: Analyze the likelihood and potential impact of each identified risk. Assign a severity rating to each risk based on its potential impact and a probability rating based on its likelihood of occurring.

- Risk Prioritization: Prioritize risks based on their severity and probability ratings, focusing on high-impact, high-probability risks first.

- Risk Monitoring and Tracking: Continuously monitor the status of identified risks, assess their effectiveness, and update mitigation strategies as needed.

Common Risks in Software Development include

- Unclear Requirements: Poorly defined or changing requirements can lead to scope creep, delays, and rework.

- Technical Challenges: Underestimating technical complexity or encountering unforeseen technical hurdles can impact project timelines and costs.

- Resource Constraints: Limited resources, such as skilled personnel or insufficient funding, can hinder project progress and quality.

- Market Uncertainties: Changing market trends, competitor actions, or economic conditions can affect project viability and success.

Benefits of Effective Risk Management

- Reduced Risk of Failure: Proactive risk identification and mitigation help prevent project failures and ensure project success.

- Improved Project Planning and Estimation: Accurate risk assessment allows for more realistic project planning and cost estimation.

- Enhanced Quality and Deliverables: Mitigating risks helps maintain project quality and deliverables that meet expectations.

- Increased Stakeholder Confidence: Effective risk management instills confidence in stakeholders and demonstrates a commitment to project success.

Risk management is an essential component of successful software development. By proactively identifying, assessing, and mitigating potential risks, organizations can minimize their impact on project outcomes, enhance project success rates, and deliver high-quality software solutions.

Why is a particular lifecycle model selected for a development environment

When choosing a software development life cycle (SDLC) model, it really depends on the type of project and the environment where it's being developed. Some models are good when requirements are fixed, and some are better when the requirements keep changing. So picking the correct model is very important to finish the project successfully.

Let's take an example. Suppose planning to build a finance management system called Enomy-Finances that will support services like savings, mortgages, and investment tracking. The bank already knows exactly what it wants. The functions and goals are clearly explained at the beginning, and they are not expected to change during the development. In this case, a Waterfall model would be a good choice.

This is because the Waterfall model works in a step-by-step order. First, the requirements are gathered, then the design is created, after that coding and testing happens, and finally the system is maintained. Each stage has to be completed before going to the next one. Since the Enomy-Finances system has fixed and clear requirements, the development team can go through each stage smoothly without needing to go back or make big changes.

Also, the Waterfall model gives complete documentation at every stage. This is useful for the bank because they can review and understand what is happening during the project. If a new developer joins, they can easily check the documents and understand the system.

But if the project was more flexible, like a mobile app for young users with changing features or feedback, then an Agile model would be better, since Agile allows more changes and user involvement.

So, in short, the Waterfall model is selected for Enomy-Finances because the environment is stable, requirements are well-known, and the project needs a structured and controlled approach. Using the right model avoids confusion, saves time, and helps deliver the product successfully.

Advantages of Waterfall Model

Clear Steps and Easy to Follow

The Waterfall model uses a step-by-step method where each stage is completed before the next one starts. This makes the whole development plan very clear and simple to understand. For every stage like gathering

requirements, designing the system, and final implementation, there is a specific output. Because of this, everyone involved in the project can easily follow what's happening and when.

Strong Requirement Understanding

At the start, the team spends a lot of time understanding and collecting all the requirements. This helps to reduce confusion and surprises later on. Since all the features are discussed early, there is less chance that things will change or new things will be added halfway through the work. That helps avoid issues like going out of scope or misunderstanding what the final system should do.

Easier to Manage

Because of its strict order, the Waterfall model is much easier for managers to track. Each phase must be finished before moving on to the next. This makes it simple to control the schedule, check progress, and manage risks or changes. It's easy to say "this part is done" and move to the next.

Detailed Documentation

Each stage of the project needs to be properly written and documented. So, even if someone forgets something or leaves the team, other members can still refer to the documents. This makes it easier to stick to the plan and to maintain the system in the future, even after many years.

Best for Stable Projects

This model is perfect when the project has clear, fixed goals and not expected to change. For example, finance software systems usually need to follow strict laws and structures, so Waterfall fits really well here. It brings discipline and avoids surprises.

Helpful for New Team Members

Because everything is already documented step by step, even if a new developer joins in the middle of the project, they can easily catch up. They just need to read the previous documents and they will understand what has been done and what is next.

Disadvantages of Waterfall Model

Hard to Change Anything

One of the main issues is that Waterfall doesn't allow much change. If the client decides to update a feature or add something new during development, it becomes very difficult. Since you can't go back easily, making any change means redoing a lot of work.

Late Testing Comes with Risk

Testing is done only after all the coding is finished. If any bugs or major problems are found at that time, fixing them might take a lot of time and effort. Also, developers may not remember exactly why they wrote the code in a certain way.

Expensive to Fix Mistakes

If something was misunderstood in the beginning, and it's found only after development, correcting it will cost more money and time. It might even delay the delivery. In flexible models like Agile, such mistakes are fixed earlier in small steps.

Too Much Depends on Early Requirements

The whole success depends on the first step – understanding what the client wants. If the client fails to explain properly or if the developers miss something, the final software might not be what the user expected. And going back will be too late or expensive.

Not Good for Complex or Changing Projects

In projects where the features keep changing or need to adapt to new customer feedback, Waterfall doesn't work well. For example, apps that depend on user feedback should not use Waterfall because it doesn't allow change once a phase is complete.

Feedback Comes Too Late

Clients or users only see the working system at the end. If they don't like something or it's not what they expected, then fixing that at the last stage is very difficult. By then, time and money are already spent.

Assessing the Merits of Using the Waterfall Lifecycle Model for the Enomy-Finances Software Project

The Waterfall lifecycle model is a good match for the Enomy-Finances project due to the clear and stable nature of the system requirements. The client has already given a detailed description of what they want, including plans for deployment on upgraded IT infrastructure and maybe switching to a web-based system later. This kind of upfront clarity fits well with Waterfall's approach, which is all about gathering full requirements at the beginning before moving into other stages.

Another reason why Waterfall works well here is because of the strict rules in the financial industry. Since financial systems must follow heavy regulatory standards, having detailed documentation and well-planned system designs becomes really important. Waterfall's nature of creating complete documents for each stage helps a lot when it comes to showing compliance with legal or government rules. These records are also useful for future audits and support.

One more advantage is the predictability the Waterfall model offers. Since every stage is done one after the other, it becomes easier to plan how long things will take and what the costs will be. This is very important for financial clients, as they often work with fixed timelines and strict budgets. There's less room for unexpected delays, which helps in keeping the project on track.

The Waterfall approach also gives strong control and easier management. Since each phase is clearly defined, it helps project managers track progress and identify any issues early. For a critical project like Enomy-Finances, where trust and accuracy matter a lot, having this level of control makes it safer and more professional.

The nature of financial systems also supports using a structured model. Things like processing savings accounts, mortgage calculations, and managing investments don't change often. These are well-established practices in the industry, which means the rigid structure of the Waterfall model is not a big drawback here. It actually makes development easier because there's less chance of sudden requirement changes.

Challenges and How to Handle Them

Although the Waterfall model has many advantages, it also brings some challenges that can affect the project if not managed properly. One major challenge is handling changes in the requirements once the development has started. Because Waterfall follows a strict sequence where each phase must finish before the next begins, going back to make changes can be difficult and costly. For example, if the client decides to add or change

features halfway through development, the project team must carefully evaluate how these changes will affect the overall timeline, budget, and progress. This means the team has to follow a formal change management process. This process helps to assess the impact of the changes and plan how to incorporate them without disrupting the whole project too much.

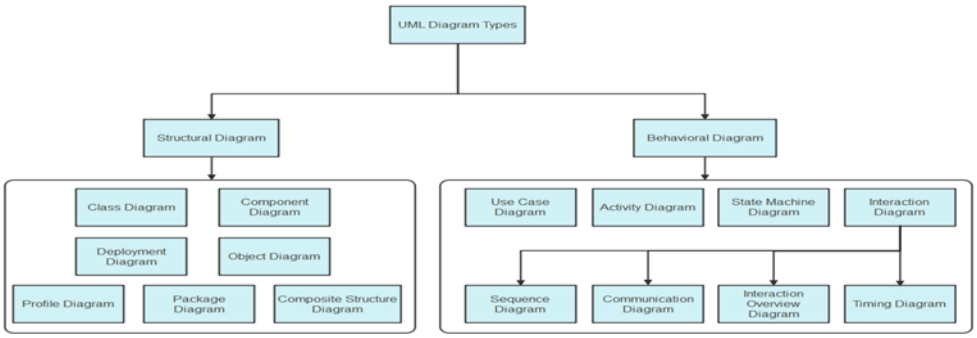
Another common risk with the Waterfall model is that testing happens only after the system is fully developed. This can lead to problems being discovered late in the project, which may delay delivery or increase costs. Because issues are not found until the testing phase, fixing them can become complicated and expensive. To reduce this risk, the development team can perform regular reviews and inspections during the earlier phases like design and coding. Peer reviews and walkthroughs allow team members to catch errors or misunderstandings early, before they become bigger problems. These proactive steps help improve the quality of the system and make sure fewer defects reach the final testing stage.

Design and Development Tools

Design Tools

Unified Modelling Language (UML)

UML, which stands for Unified Modeling Language, is a standardized modeling language comprised of an integrated set of diagrams designed to assist system and software developers in specifying, visualizing, constructing, and documenting software system artifacts, as well as business modeling and other non-software systems. The UML is a set of best engineering practices that have proven useful in modeling big and complex systems. (visual-paradigm, n.d.)



Development tools

Integrated Development Environments (IDEs)

An Integrated Development Environment (IDE) streamlines software creation by combining essential tools for programmers, including a text editor with features like auto completion and syntax highlighting, as well as debugging tools for identifying errors. It connects seamlessly to interpreters and compilers, converting code into executable formats. Modern IDEs also support version control, team collaboration, and project management. Overall, an IDE enhances the development workflow, allowing for more precise and efficient software development.

Examples of Integrated Development Environment (IDE)

Eclipse: This open source integrated development environment (IDE) for Java is known for its adaptability. Plugins can also be used to support other languages. There are many features that you can use for personal use, such as debugging, code completion, and a wide ecosystem of plugins.

Visual Studio: Microsoft created a program called Visual Studio that can work in many programming languages, including C, C++, VB.NET, and others. There are many tools for debugging, writing code, and building Windows, web, mobile, and cloud applications.

PyCharm: It has built-in tools for testing and debugging Python applications, as well as intelligent code help and code navigation, which are intended to be used only in the context of Python development.

Benefits

- Provides syntax highlighting for faster recognition of code structure. Offers code completion (auto-suggestions) to reduce errors and speed up coding. Includes templates and snippets for common code patterns.
- Integrates with Git, SVN, or other version control systems to manage code changes. Enables seamless collaboration through branch management and commit tracking. Allows for easy retrieval of previous versions and conflict resolution.

Purpose

- Provides a user-friendly interface for writing code with features like syntax highlighting and auto-completion.
- Enables running tests directly within the IDE, integrating with testing frameworks and debugging tools.
- Offers debugging features like breakpoints, variable inspection, and step-through execution to find and fix errors.
- Streamlines the coding process by combining multiple tools (e.g., editor, debugger, version control) into one environment.

Database tools

Database Management Tools (e.g., MySQL Workbench, MongoDB Compass)

Purpose

- Helps in creating, managing, and structuring databases by providing visual interfaces for designing tables, relationships, and schemas. Supports normalization and ensures efficient data modeling.
- Provides tools to analyze and optimize queries for faster data retrieval and improved performance. Offers query execution plans and suggestions for indexing and indexing strategies.

Benefits

- Helps organize and manage large volumes of data in a structured and efficient way. Ensures data integrity with constraints and relationships between tables or documents.
- Optimizes queries through indexing, caching, and query execution analysis. Reduces query response time and improves overall application performance.

Security Tools

1. **Encryption Libraries**

Purpose

- Provides encryption techniques to protect sensitive data, such as passwords, personal information, and financial transactions. Ensures that data remains unreadable to unauthorized users, both at rest and in transit.
- Encrypts stored data (e.g., in databases, files) to protect it from unauthorized access in case of a breach. Secures data in transit over networks using protocols like TLS/SSL, ensuring safe communication between clients and servers.

Benefits

- Helps organizations comply with regulatory standards such as GDPR, HIPAA, PCI-DSS, and others by ensuring proper encryption of sensitive data. Protects against fines and legal consequences associated with data breaches or non-compliance.
- Minimizes the risk of data theft, even if attackers gain access to the storage or communication channels. Makes sensitive data unreadable, reducing the value of stolen data.

Project Management Tools

When handling software projects, using proper tools can help teams stay organised and manage the work better. Some tools are more common in Agile environments, but they can still help in other models like Waterfall too.

Jira and Trello – These both are really popular tools used in many software companies. Jira is very helpful when it comes to tracking bugs or managing work during sprints. It allows the team to create tasks, assign them, and see how the progress is going. This is very useful when many people are working on the same project. Trello is a bit more simple and is used for making task boards. You can create cards for each task and move them across columns like “To Do,” “Doing,” and “Done.” It’s good for team collaboration and keeping everyone on the same page visually.

Microsoft Project – This tool is more suitable for larger or more complex projects. It helps with setting timelines, assigning resources, and tracking how much time each task takes. It’s very detailed and can be a good fit for projects that need more control and planning, like ones that use the Waterfall method. You can also see Gantt charts and manage workloads easily with this tool.

Design Tools	Purpose	Benefits
Unified Modelling Language (UML)	-	-

Wireframing Tools	To design a user-friendly interface for both clients and staff.	-
Flowcharting and Diagram Tools	To create flowcharts, DFDs, and ERDs for visualizing processes, data flow, and relationships.	Simplifies complex systems, improves clarity, and aids stakeholder understanding.
Pseudo code Editors	To outline algorithms and logic in a clear, language-agnostic format.	Helps in planning before coding, minimizing errors in development.
Development Tools		
Version Control – Git: Ensures collaboration and version tracking throughout the development lifecycle.		
API Integration – Restful APIs: To integrate real-time currency exchange rates and market data. Visual Studio Code, Eclipse, or IntelliJ IDEA		

Supported Judgements on Design and Development Tools, and Methodology

Selected Methodology: Waterfall SDLC

For the Enomy-Finances project, the Waterfall model is the more suitable choice, mainly because of how it follows a fixed and step-by-step structure. This model helps teams to handle big financial systems that need

clear plans and strict rules. Since the requirements are already mostly stable in this case, Waterfall makes it easy to go phase by phase like planning, design, development, and testing in a more organised way. It's also easier to create strong documentation, which is important for financial systems. Even though Waterfall doesn't allow changes easily during development, it gives better control over timelines and helps reduce confusion. Also, when each phase is finished fully before moving to the next one, the project can reach better quality outcomes.

Design Tools

For design works, the tools like Lucidchart and Microsoft Visio are used to make diagrams such as flowcharts, DFDs, and ER diagrams that explain the process clearly. These diagrams help both developers and clients understand the full system early in the design phase. Since Waterfall needs solid planning upfront, using these tools supports early-stage clarity and avoids mistakes later. Also, for UI design, Figma is useful to show how the user interfaces will look before coding begins. This makes sure the interface meets user needs from the start.

Development Tools

The project uses Visual Studio as the main platform to write and test the code. It has good features like debugging tools and version control, which is helpful in long-term development. MySQL Workbench is selected to manage the database systems, especially since Enomy-Finances will need secure handling of transactions and user data. For front-end parts, React.js is picked to build a more interactive and responsive user interface. Even though Waterfall is not flexible like Agile, using modern tools ensures each phase of the project is completed efficiently and with less errors.

The Enomy-Finances project brings a big chance to use a proper and trustable software lifecycle method. After looking at different options, the Waterfall model looks like the best one for this case. The reason is because it follows a step-by-step flow, supports strong documentation, and fits well for projects that got clear requirements and strict rules. Even though this model got some weak points like being hard to change or fixing bugs only at the end, these can be handled early on with proper planning and regular checking of the design and logic from the beginning.

This model also helps because it gives a clear idea of what work is done at which stage. Since this is a finance system, it's really important to make sure everything is accurate and no errors come in later. The focus on writing everything down and finishing one part before starting the next also helps the team keep track of progress and know what needs to be done next. That's why for this project, Waterfall makes sense.

Also, using the right kind of tools will make the project go smoother and less risky. Tools like UML diagrams can help us draw out how the system works. Wireframe software can help to design the user screens in advance, and using IDEs with Git or version control can make coding easier and more organised. For testing, things like JUnit for code and Selenium for web parts will help find any problems before we put the system live. These tools work well with Waterfall too, since they help make sure everything is tested after each stage.

Still, even though we're mainly using the Waterfall plan, we also want to borrow some ideas from the Spiral Model. This one is more about doing things in cycles and checking risks and client feedback again and again. So, if something changes in the business halfway through, the Spiral idea can help us manage that without starting everything from scratch. This makes the project more flexible, while Waterfall gives it the needed structure.

So finally, by mixing both models the right way and using good software tools, the *Enomy-Finances* system can be built in a way that's stable, user-friendly, and strong. It will also be easier to follow finance laws and handle future updates too. With good communication, risk handling, and repeated checking with the client, we can make a system that does the job right today and can grow tomorrow as well.

Activity 02

Feasibility Study

A feasibility study is a component of any project's or plan's early design stage. It is carried out in order to discover the strengths and weaknesses of a prospective project or an existing firm objectively. It may aid in identifying and assessing the natural environment's potential and risks, as well as the resources needed for the project and its chances of success. (CFI Team, n.d.)

Feasibility criteria types

The main areas that the feasibility study worked on below were five. Most important part of the feasibility analysis under these Economic Feasibility Study while Legal Feasibility Study is least considered feasibility analysis.

• Technical Feasibility

This one is about checking if the project can work with the tech we already have or can get. It looks into if the hardware and software needed are available and also if the tech team got the skills to finish the work. It also checks how easy it will be to update or fix the system later. If this part is not studied properly, the team might face tech problems in the middle or near the end, which can delay things and waste money badly.

• Operational Feasibility

Operational feasibility focuses on if the system will actually work in the real world and meet the users' needs properly. It looks at how simple or hard the system is to use and take care of after the launch. It also checks if the software solution planned by the developers will be accepted by the users or not. So, if users

can't handle the system well, or if it's confusing, then the whole solution becomes useless even if it works technically.

• Economic Feasibility

This one is like asking, "Is this project worth the money?" It studies all the possible costs like software, hardware, training, and other ongoing expenses. It also looks at the long-term gains. If the project costs more than the benefits or if the returns come too late, then it might not be a good idea to move ahead. It helps the organization to spend their money smartly and avoid bad investments.

• Legal Feasibility

Legal feasibility is about making sure that the project won't break any rules or laws. This means checking data protection laws, licenses, copyrights and other legal stuff. If this part is skipped or done carelessly, the project can get in trouble legally, with fines or lawsuits. That could ruin the company's image too, so it's important to do this check right at the start.

• Schedule Feasibility

This type looks into how long the project will take and whether it can be finished on time or not. It studies if the team can deliver the system before the final deadline. If the project takes too long, it can affect the business badly, and even make the whole effort pointless. So, timeline checking is needed to keep everything under control and to avoid last-minute rush or failure.

Why Feasibility Study is Important

Feasibility study is really helpful for project teams because it makes them more focused on the actual goals instead of going in random directions. It also helps to give some useful info that will be needed in the later parts of the project plan. By doing this study, the team can narrow down the business ideas and only keep the ones that are really worth doing. It also checks what kind of tools, technologies, and resources are already there and what more is needed to complete the work. Most importantly, it increases the chance that the project will be successful, since it helps the team to look at all the important things before starting. So overall, doing a feasibility study can save time, money, and effort by avoiding unexpected troubles later.

Feasibility Report

A feasibility study is a report that assesses the viability of a collection of potential project pathways or solutions. The person who writes a feasibility report assesses the feasibility of several ideas and then recommends the best alternative. They then give the feasibility study and offer their proposal to their company.

Feasibility Report for a selected organization

Feasibility Study Template

1) Introduction
2) Background
3) Outline of the system
4) Models of the system
5) Overview of the alternatives
6) Conclusion
7) Recommendation

Feasibility Study Process

- Information assessment: It analyzes the original concept of the project and fix the main aim and objective of the project.
- Information collection: It gathers the basic information and data to evaluate the many project components.

Purpose of a Feasibility Report

A feasibility report is something that helps to figure out if a project is worth doing or not before starting anything big. It checks if the idea is actually possible with what is available, like the time, money, people, and other resources. This kind of report is important because it helps the team or company not to waste effort or cash on things that maybe not gonna work well. Basically, it's like a map or early check before jumping into a project.

The report looks at different types of feasibility. For example, technical feasibility tells if the tech we need is already available or if we have to buy or build something new. Then, economic feasibility checks the cost and the money we might earn or save from the project. Also, there is legal feasibility, which is about following all laws, like data protection or software licences. If we ignore legal parts, it might cause serious troubles later like fines or court problems.

Operational feasibility is another thing that checks if the final system or product is gonna work well for the people who use it. Like, will it be easy for them to use? Will it help them do their work faster or better? Then comes schedule feasibility, which is to check if the team can finish everything on time. If the project takes too long, it might become useless or too costly.

Also, the feasibility report shows if the current team have the skills or if they need to hire someone from outside or train the staff. This helps to know early what support is needed. It gives a full picture of what can be done and what problems might come. Decision makers use this report to decide if the project can go ahead, needs changes, or should stop.

To put it simple, a feasibility report is like a reality check. It don't just look at dreams or ideas but sees if it really works in real life. It saves time, energy, and helps in taking smart decisions before jumping in blind.

Benefits of a Feasibility Report

A feasibility report brings many useful things when someone planning to start a project. One of the main benefits is that it helps the team and stakeholders to make more proper and smarter decisions. Instead of guessing or going only by opinion, the report gives proper facts and checks, so the decisions go with the company goals and the resources they already got. This makes sure nothing is going out of hand too early.

Another good thing is, the report helps in spotting problems before they even come. Like, if something might not work or needs to be fixed, the report will show that early. This lets the project team think and make better plans or even change some parts before spending more money or effort. It's like a warning system that saves time later.

Also, the way the report is written is usually very organized. It don't just throw everything in one place. Most of the time, it comes with a short summary at the start, then goes into proper detail, and later gives suggestions or a conclusion. This helps people who have to take decisions, especially managers or clients, to understand the situation fast without reading too much confusing stuff.

A big benefit also is, it keeps things simple. Sometimes, projects can have very difficult or technical things, but not everyone in the company understands that level. The feasibility report changes that kind of hard data into easier words. This way, even people who are not from technical side can still understand what's going on. It avoids all the heavy technical terms unless really needed.

So, in the end, the feasibility report becomes like a helper tool that brings everything together in one place – facts, possible risks, good ideas, and suggestions. It doesn't only tell if the project is possible, but also shows how to improve things and make smart choices. That's why it's so important before starting any serious work.

Purpose of the Feasibility Report.

The purpose of a feasibility report is to figure out if a new business idea or project can actually be done successfully. It helps business owners or project managers understand all the important parts before they start anything big. This report looks into many things like costs, time, resources, and technical needs to make sure the project is worth doing. It acts like a guide that helps decide if the plan should go ahead, be improved, or just dropped. A feasibility report is very useful for making smart choices and saving money, time, and effort from being wasted on something that may not work well in the end.

One of the first things looked at in the report is the target market. This means finding out who the possible customers are, what they like, and what kind of people they are (like age, income, etc.). Knowing this helps businesses create better products and services that match the exact needs of the customers. If the business knows its market well, it can advertise better and not waste money on wrong marketing.

Another key part of the report is learning about the competitors. This means checking out other companies that are doing the same thing and finding out what they are good at and where they are weak. By doing this, a business can figure out how to do something different or better than others. This helps it stand out in the market and attract more customers.

Money matters a lot too. So, the report also includes a deep look at how much it will cost to start the business and how much it will take to run it every month. This includes things like buying equipment, paying rent, staff salaries, and ads. Business owners use this info to make a budget and avoid spending too much money.

The report also looks at both risks and rewards. Risks can include things like changes in the market, problems in running the business, or even new government rules. But the report also points out the possible benefits, like earning profits and growing the business. This helps people compare the good and bad sides before making a final choice.

Understanding how big the business can get is also part of the study. The report gives an idea of how much the business might sell based on market research and prices. Knowing this helps to see if the business can make enough money to survive and grow.

Along with this, the report includes a profit and loss prediction. This is a rough idea of how much money the business will make or lose over time. It shows the total income, costs, and profits expected, which is very helpful to check if the plan is financially strong.

Lastly, the report talks about future growth. It looks into industry trends, market changes, and technology to see if the business can expand later. If there's a chance to grow bigger, launch new services, or enter more markets, the business can make plans from the beginning. So, in short, a feasibility report helps build a strong base for starting any new project.

Feasibility Criteria	Example
1) Technical Feasibility	This part looks at whether the necessary equipment, software, and technologies are available for the project. It also checks if the development team has the right technical skills and knowledge to complete the work. Additionally, it makes sure that the chosen technology is reliable and well-established so that there won't be unexpected failures.
2) Operational Feasibility	Here, the focus is on how well the software will satisfy the needs and expectations of the users. It evaluates the usability of the system to confirm that it is easy and practical to use. Also, it checks if the solution proposed by the development team is acceptable to both the users and the business itself.

3) Economic Feasibility	This part examines all the costs involved in the project, including expenses for hardware, software, design, development, and ongoing operations. It assesses whether these costs will bring enough benefits to the company over time. It also considers costs related to important activities like gathering and analyzing requirements.
4) Legal Feasibility	This section verifies that the project follows all relevant laws, rules, and ethical guidelines. It makes sure that data protection laws are respected and that the project has all the required permissions and copyrights needed for legal operation.
5) Schedule Feasibility	This checks if the planned timeline and deadlines for the project fit well with the company's goals. It also looks at the availability of resources and any possible delays that might affect the schedule. Lastly, it identifies risks that could cause hold-ups and plans how to handle them if they happen.

FEASIBILITY REPORT FOR ECONOMY FINANCE PROJECT

Feasibility Report

Enomy-Finances System Project

Prepared by: [Junior Software Developer. AASHIK]

Date: November 30, 2024

Executive Summary

This report looks into how possible it is to create a new computer system for Enomy-Finances that will help their growing financial services. It checks different parts like the technology needed, how the system will work day-to-day, the costs involved, and if the project can be finished on time.

Introduction

Phonyt Digital Solutions (PDS) aims to enhance its customer engagement and service delivery through an advanced booking system. The proposed solution will enable seamless appointment scheduling, client data management, and personalized services. This report evaluates the potential success of the project using key feasibility criteria.

Problem Definition for Phonyt Digital Solutions (PDS)

Phonyt Digital Solutions (PDS) seeks to develop a comprehensive software solution to enhance its operational efficiency, client satisfaction, and overall business performance. The new system must address the following challenges and requirements

PROPOSED SOLUTION

The new system will be a web-based application built with cutting-edge technologies. By moving to a web-based platform, the system eliminates the need for client applications to be installed on each computer, making updates easier and more accessible from any device with an internet connection.

The new system includes several important parts

Currency Conversion Module

1. This lets clients exchange multiple currencies like GBP, USD, EUR, BRL, JPY, and TRY.
2. It connects with real-time data from external currency exchange APIs.
3. The system automatically calculates fees based on the transaction amount, with fees getting smaller as the amount increases (for example, 3.5% fee for transactions up to 500 GBP, and 1.5% for amounts over 2500 GBP).
4. There are set limits for transactions, with a minimum of 300 GBP and a maximum of 5000 GBP.

Savings and Investment Management Module

1. Clients can enter their investments, including lump sums, monthly payments, and choose the type of investment.
2. It offers personalized investment quotes showing possible returns over 1, 5, and 10 years, with maximum and minimum projections.
3. The system clearly displays fees, taxes, and profits for each investment plan, such as Basic Savings Plan, Savings Plan Plus, and Managed Stock Investments.
4. It also handles user input errors by giving clear feedback and logs errors for future troubleshooting.

Secure Data Management

1. The system securely stores user data like contact details, transaction history, and saved investment quotes.
2. It uses encryption to protect data both when stored and while being transferred.
3. Role-Based Access Control (RBAC) is implemented to ensure only authorized people can access sensitive information.

Web-Based Platform

1. Users can access the system through web browsers on different devices such as laptops, tablets, and desktops.

2. It features a modern, easy-to-use interface that makes text, numbers, and graphs easy to understand for users.

Scope and Objectives of the New System for Phonyt Digital Solutions (PDS)

Scope of the New System

1. **Client Interaction Platform**
   Provide real-time communication tools, a centralized client portal, and a project tracking system.
2. **Data Management and Security**
   Implement secure storage and retrieval mechanisms, including encryption, access controls, and backup systems.
3. **Scalability**
   Design the system to scale with the growing needs of the organization and accommodate increased user traffic and data loads.

Objectives of the New System

1. **Enhanced Operational Efficiency**
   Streamline internal processes to reduce manual effort, minimize errors, and improve productivity.
2. **Improved Client Engagement**
   Build stronger relationships with clients by offering transparency, real-time updates, and collaboration tools.
3. **Strengthened Data Security**
   Safeguard client and internal data through industry-standard security protocols and measures.

Technical Feasibility

For the Enomy-Finances system, the main technologies planned include database management using MySQL or MongoDB. Development will be done using Python and JavaScript frameworks, specifically Node.js for backend and React for frontend. The system will be hosted on cloud platforms like AWS or Microsoft Azure to benefit from scalability and reliability.

The project team consists of 4 project managers, 6 developers, 1 technical support specialist, and 1 trainer to ensure smooth operation and knowledge transfer.

There are some challenges expected. For example, the team may initially lack experience with such a complex system design, but this will be addressed by adopting Agile development. Breaking the work into smaller parts allows the team to learn progressively while delivering usable features. Additionally, to handle future growth, the architecture will be modular and cloud-based, ensuring it can easily scale as user demand increases.

Technology Assessment

Right now, the system is a networked application, but there's a plan to move it into a web-based platform. This will require modern web technologies and frameworks, a secure database system, and strong authentication and authorization features. It also needs to connect well with the existing financial software the company uses.

Technical Risks

There are some risks involved like moving data from the old system to the new one, making sure the financial data stays secure, ensuring the system can grow as more users come onboard, and integrating with other external financial services.

Technical Viability

The software will use event-driven programming so the interface reacts quickly to user actions. The system's design will be object-oriented to keep things organized and easy to update. Business logic will use procedural programming, and data processing will use functional programming for efficiency.

Operational Feasibility

The system's operation will impact employees and users in several ways

- **Job Changes:** As automation takes over some manual tasks, employees will receive training to take on new responsibilities such as managing advanced financial tools or supporting clients.
- **Security and Privacy:** The system will enforce strong security, including user authentication, data encryption both at rest and during transmission, and role-based access control to protect sensitive information. Regular updates and security checks will be performed, alongside user education about data safety.
- **User Acceptance:** To encourage adoption, a clear communication plan will explain system benefits. Training sessions, easy-to-understand tutorials, and a dedicated support helpdesk will be provided. Early users will be encouraged to share their positive experiences to help others adapt.

Economic Feasibility

Development Costs

Costs include hiring the software development team, upgrading infrastructure, performing tests, and creating training materials and documentation.

Operational Costs

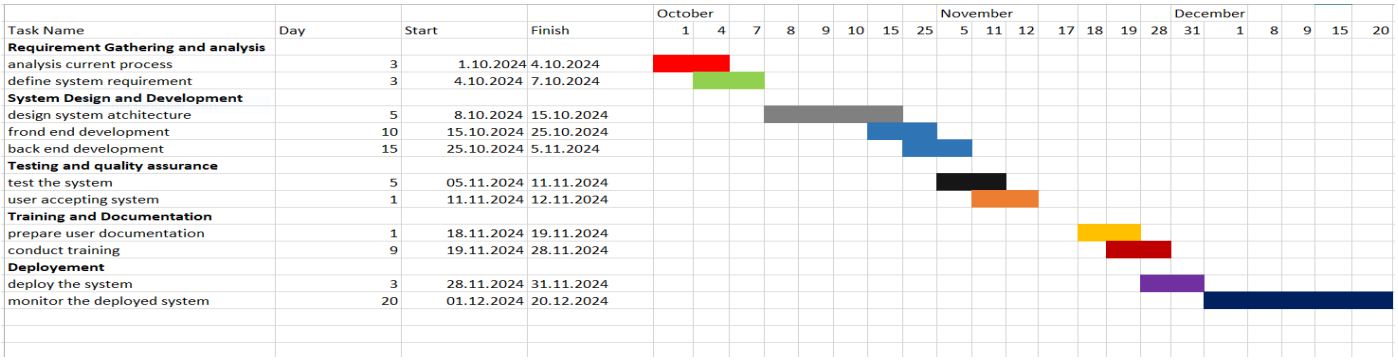
Ongoing costs will cover system maintenance, support services, hosting fees, and keeping security measures up-to-date.

Benefits

This new system is expected to increase operational efficiency, boost customer satisfaction, improve the accuracy of data, and allow the company to expand easily in the future.

Schedule Feasibility

Project Timeline



Task Name	Day	Start	Finish
Requirement Gathering and analysis			
analysis current process	3	1.10.2024	4.10.2024
define system requirement	3	4.10.2024	7.10.2024
System Design and Development			
design system atchitecture	5	8.10.2024	15.10.2024
frond end development	10	15.10.2024	25.10.2024
back end development	15	25.10.2024	5.11.2024
Testing and quality assurance			
test the system	5	05.11.2024	11.11.2024
user accepting system	1	11.11.2024	12.11.2024
Training and Documentation			
prepare user documentation	1	18.11.2024	19.11.2024
conduct training	9	19.11.2024	28.11.2024
Deployement			
deploy the system	3	28.11.2024	31.11.2024
monitor the deployed system	20	01.12.2024	20.12.2024

Legal Feasibility

Enomy-Finances complies fully with the Data Protection Act No. 24 of 2019, ensuring all personal and financial data is handled lawfully. The system also meets security standards like PCI DSS to protect payment transactions, reducing risks related to data breaches or legal penalties.

Cultural Feasibility

The system is designed to support multiple languages, including local dialects, to serve a wide range of users. Its interfaces and content respect cultural differences, ensuring inclusiveness. Training programs will be adapted to fit users' familiar workflows and cultural norms. Moreover, the system promotes financial inclusion by addressing the needs of diverse demographic groups and economic backgrounds.

Technical Assessment and Risks

The current infrastructure is a networked application, with plans to migrate to a modern web-based platform. The system will use event-driven programming for responsive UI, object-oriented design for better

maintainability, procedural coding for business logic, and functional programming for efficient data processing.

Risks include data migration challenges, ensuring security of sensitive financial data, scaling the system properly, and integrating with third-party financial services. Mitigation plans involve strong encryption, comprehensive testing, and phased deployment using Agile practices.

Operational Impact

The new system is expected to boost customer service quality, increase staff productivity, improve accuracy in data handling, and streamline operational workflows. Staff and users will receive thorough documentation and ongoing support to ease the transition.

Economic Details

Costs cover software development, infrastructure upgrades, quality assurance, training, and ongoing maintenance. Benefits include improved efficiency, higher customer satisfaction, better data quality, and capacity to grow with business needs.

Risk Management

Primary Risks

The main risks include protecting data security and complying with regulations, handling difficulties during system integration, overcoming user resistance to change, avoiding delays, and keeping costs under control.

Risk Mitigation Strategies

To manage these risks, strong security practices will be put in place. The company will keep clear and regular communication with clients, provide thorough training programs, adopt Agile development for flexibility, and closely monitor progress throughout the project.

Recommendation

- ☒ Encourages iterative progress, stakeholder collaboration, and adaptability.
- ☒ The proposed solution aligns with PDS's goals, demonstrating technical, operational, economic, and scheduling feasibility.
- ☒ Regular updates to stakeholders and iterative testing will ensure timely and within-budget delivery.
- ☒ Continue to develop the Enomy-Finances system using the Agile methodology to ensure flexibility, adaptability, and active stakeholder participation throughout the development process.
- ☒ Conduct a detailed cost-benefit analysis tailored to the organization's specific needs to determine the most efficient development approach, whether custom-built, off-the-shelf, cloud-based, hybrid, open-source, or outsourced.
- ☒ Implement robust security measures to protect sensitive financial data and maintain regulatory compliance with standards such as PCI DSS and GDPR.

ALTERNATIVE SOLUTIONS AND COST/BENEFIT COMPARISON

In order to meet the needs of Economy-Finances, two potential technical solutions were considered for the proposed system: Financial software off the shelf and developed in the house. Below are various criteria of comparison between two solutions provided based on the cost, customization and scalability, security and flexibility

Comparison of Both Solutions

Criteria	**Off-the-Shelf Financial Software**	**In-House Development**

Customization	Limited, cannot fully align with specific needs	Full customization, tailored to business requirements
Initial Cost	Lower	Higher
Implementation Time	Shorter (1-3 months)	Longer (6-12 months)
Scalability	Limited, future growth may require additional purchases	Fully scalable, future updates can be easily implemented
Security	Standard security features may lack specific compliance requirements	Full control over security, can meet all regulatory standards
Ongoing Costs	Annual license fees (15-20% of initial cost)	Maintenance costs (15-20% of initial cost)
Vendor Dependency	High, reliant on vendor for updates and support	Low, no dependency on external vendors

Conclusion

After looking at everything, it seems like this project can really work out well. The team at PDS has the right technical skills to handle it. Also, the project fits nicely with what the business wants to achieve. While there are some risks, they look like things that can be controlled or fixed. The schedule planned is reasonable and

can be met. Most importantly, the money benefits expected from the project make the cost worth it. So overall, the project looks doable and worth moving forward.

Overview of How Different Technical Solutions Can Be used

Three Possible Technical Approaches for the Enomy-Finances System

1. Web-Based Application Architecture

This approach centers around making the application easily accessible over the internet. It is intended to support a wide user base across different locations, allowing access through standard web browsers without needing local installations. The focus is on flexibility, remote usage, and ease of deployment.

2. Desktop-Based Network System

This solution is designed to be installed on individual client machines within a network. It prioritizes performance and enhanced security by running locally on users' computers, while still relying on a central network for data exchange and processing.

3. Hybrid Architecture

A blended solution that integrates the capabilities of both web-based and desktop applications. This option attempts to capture the strengths of each method—ease of access and rich processing power—while minimizing their weaknesses.

Justification for Recommending the Web-Based Solution

A web-based approach is ideal for systems that require widespread accessibility and centralized updates. The following factors make this option more suitable for the Enomy-Finances System:

- **Universal Access**: Users can operate the system from any location via a browser.
- **Compatibility with Modern Technologies**: Easily integrates with third-party services through APIs.
- **Lower Cost of Ownership**: Hosting, deployment, and updates can be centralized, reducing overheads.
- **Ease of Maintenance**: Regular patches and feature updates can be deployed without interfering with users.

1. Web-Based Solution: Technical Overview

Frontend (User Interface)

Typically developed using modern frameworks that support Single Page Applications (SPA). This design allows the interface to update dynamically without reloading entire pages, resulting in faster performance and a seamless user experience.

Backend (Server Logic)

The backend communicates with the frontend through RESTful APIs, which separate the presentation layer from business logic. This makes it easier to manage each layer independently, and updates can be implemented without disrupting the entire system.

Database Layer

The system uses a distributed database infrastructure, which stores data across multiple geographically separated servers. This helps to improve fault tolerance, boost availability, and support high-traffic loads without performance drops.

Benefits

- **Accessibility**: No installations are needed; any internet-connected device with a browser can access the application.
- **Scalability**: Cloud platforms allow on-demand resource allocation. More users can be accommodated by adding server instances.

- **Third-party Integration**: REST APIs simplify connections to payment gateways, analytics services, and other web technologies.

Drawbacks

- **Dependence on Internet Speed**: Users need a stable internet connection; otherwise, the system may lag or become inaccessible.
- **Security Vulnerabilities**: Being internet-facing, the system is more prone to threats. Strong encryption, security protocols, and intrusion detection tools are essential.

2. Desktop-Based Network Solution: Technical Breakdown

Client-Side (User Application)

The system runs as a full-featured application installed on each client machine. This allows the interface to make better use of system resources, leading to faster and more responsive performance.

Server-Side

A central server is responsible for data synchronization, business logic processing, and system-wide updates.

Data Management

The application may cache data locally and sync periodically with the central database. This method improves speed and ensures functionality even with network interruptions.

Benefits

- **High Performance**: Local resources process data quickly, and only essential data travels over the network.
- **Enhanced Security**: By minimizing exposure to the internet, the system becomes less vulnerable. Security measures can be implemented within the local network.

- **Offline Operation**: Users can continue working when disconnected, and data can sync when reconnected.

Drawbacks

- **Time-Consuming Updates**: Software changes must be installed on each device, requiring manual effort or automation tools.
- **Limited Access**: Remote users face challenges unless virtual private networks (VPNs) or remote desktop services are set up.
- **OS Compatibility**: May be restricted to specific operating systems, reducing user flexibility.

3. Hybrid Solution: Architecture and Evaluation

Core Functionality

Most functionalities are available through a central web platform, ensuring access through browsers. Critical or heavy processes are managed by native desktop applications.

Desktop Component

Designed to support resource-intensive tasks such as data analysis or large transaction processing using the local system's capabilities.

Mobile Access

Progressive Web Apps (PWAs) ensure users on mobile devices get a responsive interface without downloading separate mobile applications.

Advantages

- **Greater Flexibility**: Users can switch between web, desktop, and mobile interfaces depending on their situation.

- **Optimized User Experience**: While basic tasks can be done through a browser, high-load operations benefit from local processing.
- **Offline Functionality**: The hybrid model ensures that work continues even during outages, and synchronizes once connectivity is restored.

Disadvantages

- **Increased Development Time**: Each component (web, desktop, mobile) must be developed, tested, and maintained independently.
- **Higher Expenses**: More personnel with platform-specific expertise are needed. Additional infrastructure is required to support all environments, increasing both development and operational costs.

Technical Requirements Comparison

Feature	**Web-Based**	**Desktop-Based**	**Hybrid**
Accessibility	Excellent	Fair	Good
Security	Good	Excellent	Good
Scalability	Excellent	Good	Good
Offline Capabilities	Limited	Excellent	Good
Development Speed	Fast	Medium	Slow
Integration Ease	Excellent	Good	Good

Recommended Solution

1. **Supports Growth**

   The system is built with scalability in mind, so it can easily handle more users and data as the business grows. This means that when there's a rise in demand, the application can adjust without needing a complete redesign. Additionally, because the system uses modern deployment techniques, updates and new features can be released quickly and smoothly without causing downtime for users. This flexibility allows the system to evolve alongside changing business needs, making sure it always fits what the organization requires.

2. **Modern Architecture**

   By using up-to-date architectural designs, the application can integrate easily with other software and services, such as payment gateways, analytics, or customer management tools. This ability to connect with third-party services greatly expands what the system can do and helps keep it relevant. Security is also improved thanks to modern frameworks and protocols that protect sensitive user information. This builds confidence among users that their data is safe, which is vital for financial applications.

3. **Cost-Effective**

   Web-based applications generally cost less to run and maintain compared to traditional software because they don't require complex installations on every user device. The infrastructure demands are lower since much of the processing happens on cloud servers, which means the organization can better manage expenses and allocate resources to other priorities. The faster deployment cycle also reduces the time and effort needed for updates, which lowers ongoing development costs.

4. **User Accessibility**

   Designing the system to be mobile-friendly means users can easily interact with it from their smartphones, tablets, or desktops. This enhances convenience and increases overall engagement because people can use the app anytime, anywhere. Furthermore, since there's no need for users to install software, the onboarding process is much simpler. New users can quickly access the system through their web browsers, which reduces barriers to adoption.

Implementation Considerations

Technology Stack

- **Frontend**

  Choosing modern frontend frameworks such as React, Angular, or Vue.js helps create interfaces that respond quickly to user actions and adapt smoothly to different screen sizes. These frameworks support building Single Page Applications (SPA), where only parts of the page update instead of the whole page reloading. This approach offers a faster and more seamless user experience. Also, progressive enhancement techniques ensure that users with older browsers still get a functional experience, while users with modern browsers enjoy advanced features.

- **Backend**

  Backend services can be developed using technologies like Node.js, Java Spring, or .NET Core, which are known for being robust and scalable. Using a microservices architecture means the backend is broken into smaller, independent services that handle specific business functions. Each service can be deployed, updated, and scaled independently, which improves maintainability and allows faster development cycles. RESTful APIs serve as the communication bridge between the frontend and backend, making sure data is exchanged clearly and securely without confusion.

- **Database**

  For storing structured data such as customer info and transactions, PostgreSQL is a reliable relational database choice. For handling unstructured or semi-structured data, MongoDB is suitable. To ensure the system is always available, replication strategies are put in place where data is copied across multiple servers. This way, if one server fails, another can quickly take over, minimizing downtime. Regular data backups are also critical they allow quick restoration of information in case of accidental loss or corruption.

Security Measures

Authentication

OAuth 2.0 is used to securely handle user permissions, letting users authorize third-party apps to access limited data without sharing passwords. The system also implements strong session management practices: sessions expire after a period of inactivity, and secure cookies are used to prevent session hijacking, where attackers try to steal user sessions.

- **Data Protection**

  All sensitive data is encrypted both when stored and transmitted, ensuring it cannot be read if intercepted by unauthorized people. In testing or development environments, sensitive data is masked to prevent accidental leaks. The system keeps detailed audit logs of all user activity, which help the organization monitor who accessed what data and detect suspicious behavior early.

Performance Optimization

Caching

To reduce unnecessary server load and speed up user experience, the system employs caching at multiple levels. Browser caching stores common resources like images and scripts locally, so users don't have to download them every time. On the server side, API response caching saves the results of costly queries, preventing repeated heavy computation and speeding up response times for users.

- **Load Balancing**

  Distributing incoming user requests across several servers helps balance the workload, so no single server becomes overwhelmed. This reduces delays and improves responsiveness, especially for users in different geographic locations. Load balancers also provide failover capabilities if one server goes down, traffic is automatically rerouted to healthy servers, keeping the system available without interruption.

Discussion of the Different Components of a Feasibility Report

Component	Purpose	Content	Importance
Executive summary	Provides an overview of the feasibility study as a whole, summarizing the	This section briefly highlights the project objectives, the main aspects of the feasibility analysis and the final recommendations.	Helps stakeholders quickly understand the overall findings and feasibility of the

	problem, the proposed solution, and the main conclusions.		project without going into detail.
Introduction	Presents the problem or opportunity that the project aims to solve and the purpose of the feasibility report.	The introduction describes the context, scope, and objectives of the proposed project. It may also state the scope of the feasibility study and any limitations or restrictions.	This section establishes the context for the report and clarifies the problem the project seeks to solve.
Problem Statement	Clearly defines the problem the project aims to solve.	A detailed description of the problems or challenges that the proposed solution addresses. This may include relevant data or evidence showing the extent and impact of the problem.	This section helps all stakeholders to understand the main problem and why the project is necessary.
Alternative solutions	Explore different possible solutions to the problem.	This section presents some alternative solutions or approaches to solving the identified problem. For each alternative, it should highlight the strengths, weaknesses, and risks associated with it.	By presenting the alternatives, stakeholders can evaluate the variety of options available and understand why one may be more appropriate than the others.
Technical and Market Analysis	Assess the technical feasibility of implementing the proposed solution.	This section examines the technology requirements and evaluates the ability of existing infrastructure or technologies to support the project. It also assesses potential technical challenges and	Helps determine whether the organization has the technical skills and resources needed to complete the project.

		the resources required to implement the solution.	
Operational Feasibility	To assess the operational aspects of implementing the proposed solution.	This section examines how the proposed solution will fit into the organization's current operating structure. It assesses the feasibility of implementing the solution in practice, including the potential impact on workflow, existing systems, and personnel.	Ensures that the solution can be integrated into the organization's day-to-day operations without disrupting the organization's normal operations.
Financial Feasibility	To assess the financial viability of the proposed project.	This section provides a detailed financial analysis, including cost estimates for development, implementation, and maintenance. It may also include projections of revenue or cost savings that the project will generate. It often involves low-level analysis, return on investment (ROI), and financial risk assessment.	Financial feasibility ensures that the project is affordable and can generate adequate returns, which is essential for obtaining funding and support.
Legal and Regulatory Feasibility	Assess whether the proposed solution complies with legal and regulatory requirements.	This section identifies relevant laws, regulations, and industry standards with which the project must comply. This may include data protection laws, compliance with industry regulations and licensing requirements.	Helps mitigate legal risks by ensuring that the project does not violate any laws or regulations.
Conclusions and Recommendations	To summarize the findings and make recommendations.	Based on the feasibility analysis, this section provides a recommendation to continue, modify, or abandon the project. It	This is the final decision-making section, where stakeholders can

		may also suggest other actions or areas for investigation.	assess whether the project is worth continuing based on the analysis.

Assessment of the Impact of the Feasibility Study on the Enomy-Finances Project

Operational Workflow Analysis

In this part, I will take a close look at how the current financial services work at Enomy-Finances. The goal is to study how tasks are handled daily and find out if there's any delays, confusion, or repeated efforts that can be improved. This study will help make the new system more efficient and user-friendly for both staff and customers. While doing this, I will also try to reduce any problems that might come up during the switch to the new system. It's important the updated system fits well with the company's future plans and growth. This way, the workflow remains smooth even after the change.

Design Prototypes

Before starting to code or build the full system, I plan to design some rough drafts or simple versions (mock-ups) of the software screens. These early designs will be created based on the ideas, feedback, and expectations of the client. By doing this, we can make sure the development is heading in the right direction. If the client notices any problems or changes, we can fix it early on before too much time and effort is used. This step helps avoid rework later and keeps the client involved in the process from the beginning.

Preliminary Investigation

At the beginning, I will deeply investigate and study all the requirements provided by Enomy-Finances. This step helps clear up any doubts or vague points that may confuse developers later. I'll also spend time talking with the client to properly understand what they really want, especially as they plan to move from their old

offline or networked system to a modern web-based one. Getting clarity at this early stage is super important to avoid misunderstandings in the later stages of development.

Technical Assessment

Here, I will check how well the current system is working and find out what its limits are. For example, I'll see if it struggles when many users log in at once or if it runs slow during peak times. Based on that, I'll explain how moving to a web-based platform can solve these problems. I'll also check if the existing hardware and software need upgrades. If the current setup isn't strong enough to support the new system, those issues must be fixed first to ensure everything works properly when the system goes live.

Cost-Benefit Analysis

Another part of the feasibility study is figuring out if the project is financially worth doing. I will help by estimating how much money will be needed to build the system, from start to finish. I'll also look for ways to save money, like using open-source tools or reusing code where possible. On the other side, I will try to predict how the system will benefit the business, such as reducing paperwork, saving employee time, or attracting more customers. These comparisons will help decide if the investment is smart and profitable.

What Happens After the Assessment?

Feasibility Study Results

After completing all the above steps, the final report will show whether the project can be done or not. If the findings are mostly positive like being affordable, technically possible, and useful for the business then the team can move forward with confidence. They won't have to worry later about big problems or surprises because all those risks were checked already during the study.

Building Client Trust

By doing a detailed feasibility study and openly sharing the results with the client, it shows them that we take the project seriously. Also, by involving them in discussions and decisions, they will feel more connected and

confident about the work being done. This trust is super important. When the client feels that we understand their needs and are giving honest advice, they are more likely to work with us again in the future.

System Development

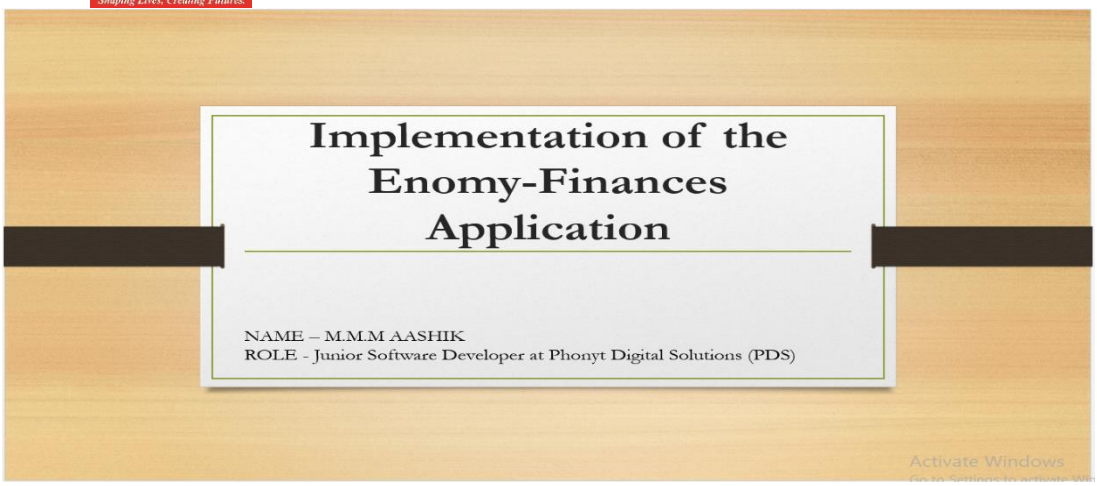
After finishing the feasibility checks and confirming the project is good to go, the next step is to start designing and building the software. The information collected during the study will act as a blueprint or guide. It will help the developers and designers know exactly what to build, what to avoid, and what to focus on. This makes the development smoother and ensures the final software fits the client's actual needs.

Personal Learning and Growth

From a personal point of view, working on this project will give me real-world experience in many areas like system analysis, planning, communication, and working with clients. It's not just about building software it's about understanding people, managing time, handling problems, and learning to think like a professional. These lessons will stay with me and help me grow in my future career as a software engineer or project manager.

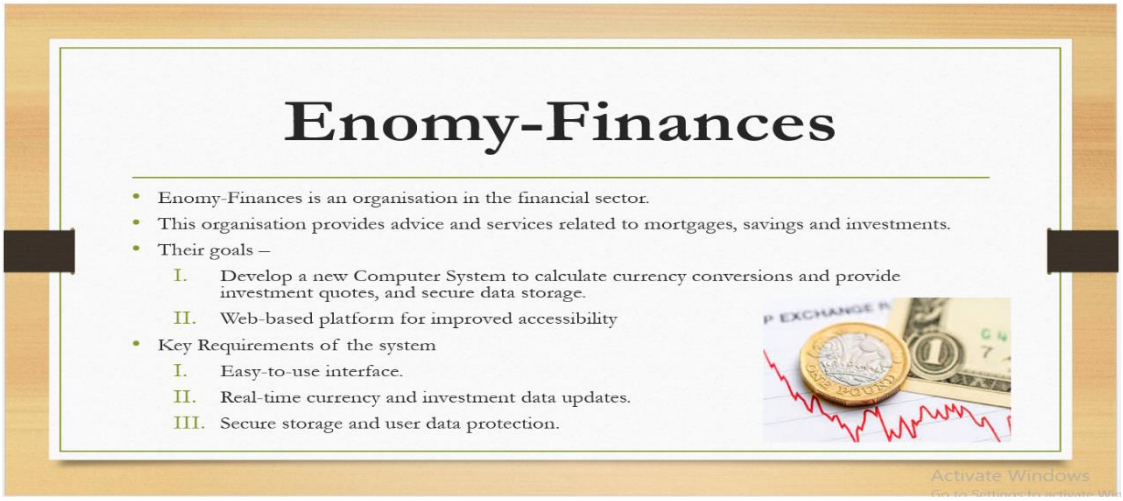
Activity 03

Proposed Computer System for Economy-Finances



Speaker Note:

Welcome to my presentation on the proposed computer system for Economy and Finances. In this presentation, I will walk you through the development approach, system scope, stakeholder requirements, and key considerations to improve the organization's efficiency and client experience.



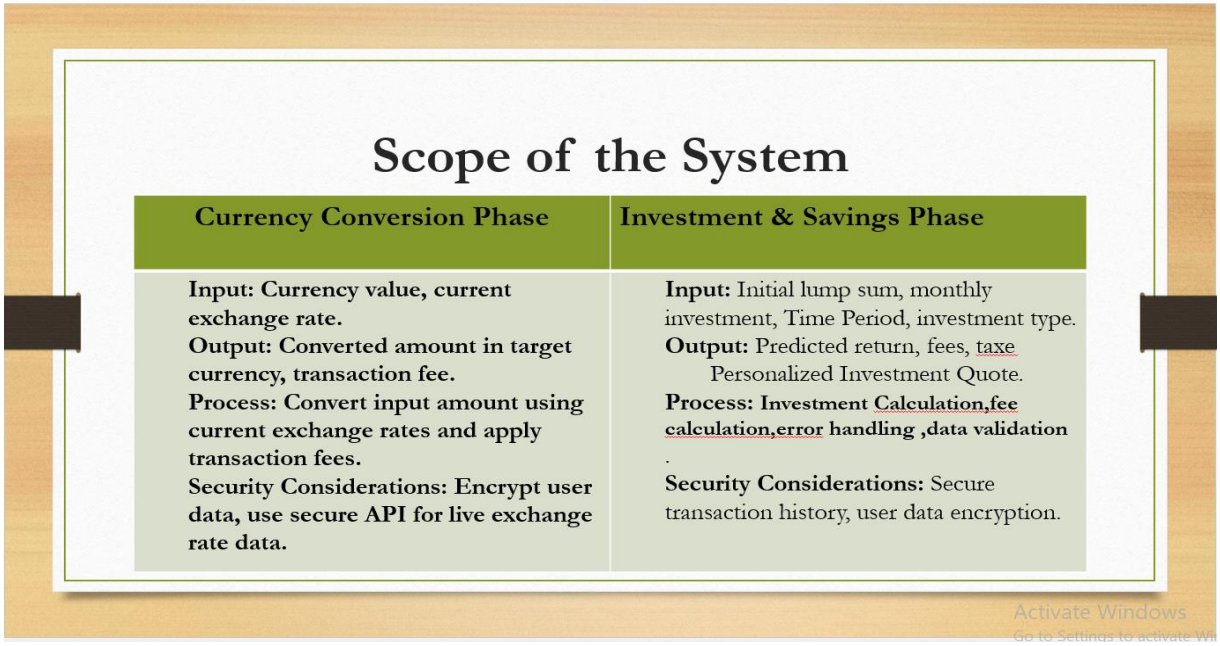
Speaker Notes:

Enomy-Finances is a financial organization offering services like mortgage advice, savings, and investment solutions. The company aims to develop a new computer system with the following goals:

1. **Goals:**
    - Introduce a web-based platform to calculate currency conversions and provide investment quotes, ensuring easy access for users.

- Incorporate secure data storage to protect sensitive user and financial information.

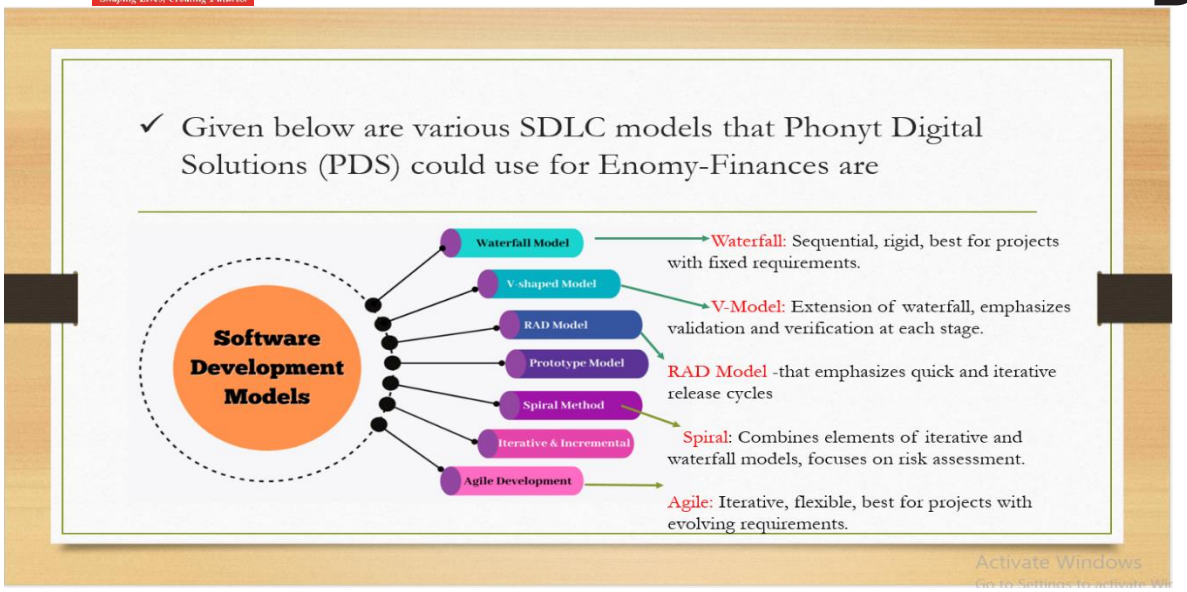
2. **Key System Requirements:**
   - A simple and intuitive interface to ensure usability for all users.
   - Real-time updates for currency conversions and investment data, providing accurate and up-to-date information.
   - Robust security measures to protect user data and ensure compliance with financial regulations.



Speaker Notes:

Scope of the System

The system focuses on modules like currency conversion, savings, and investment planning while ensuring security and scalability. It will improve client accessibility with a web-based platform and real-time data updates, meeting evolving user needs.

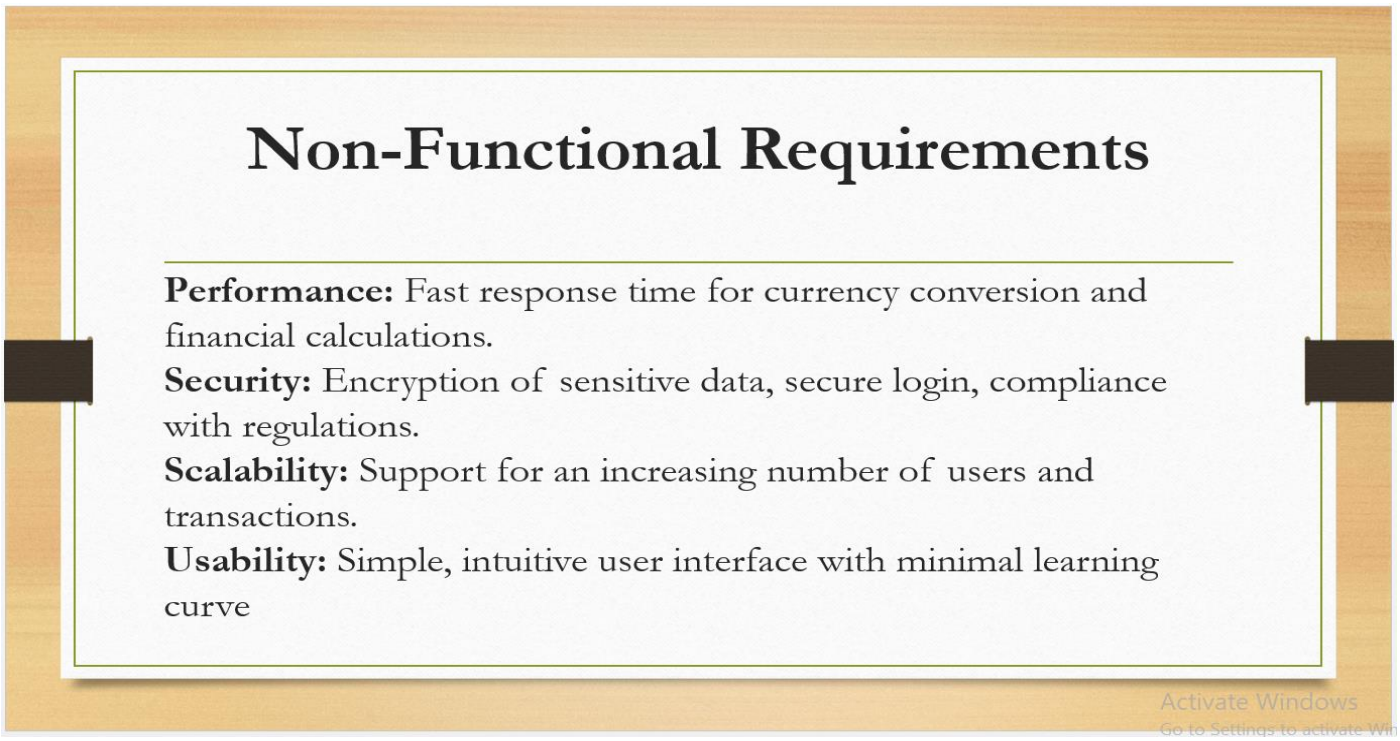


SDLC Models for Enomy-Finances

Phonyt Digital Solutions evaluated models like Waterfall, V-Model, RAD, Spiral, and Agile. Agile was chosen for its flexibility, iterative feedback cycles, and alignment with Enomy-Finances' dynamic requirements.

Functional Requirements

The system must handle real-time currency updates, investment calculations, and secure data storage. Features include client registration, portfolio management, and online access to planning tools.



Non-Functional Requirements

The system must ensure fast performance, robust security, scalability for growth, and an intuitive interface. These ensure smooth operation and user satisfaction.

Stakeholder Requirements

Stakeholder	Key Requirements
Clients (End Users)	User-friendly interface, real-time currency conversion, data security, error handling, transaction history.
Company Management	Scalability, secure data storage, regulatory compliance, reporting, cost control.
Development Team	Clear requirements, access to tools, testing environment, documentation, collaboration.
Regulatory Bodies	Compliance with regulations, data privacy, auditing and reporting capabilities.
IT Support	System maintenance, performance monitoring, backup & disaster recovery.
Marketing & Sales	Customer insights, user acquisition, customer support integration.

Stakeholder Requirements

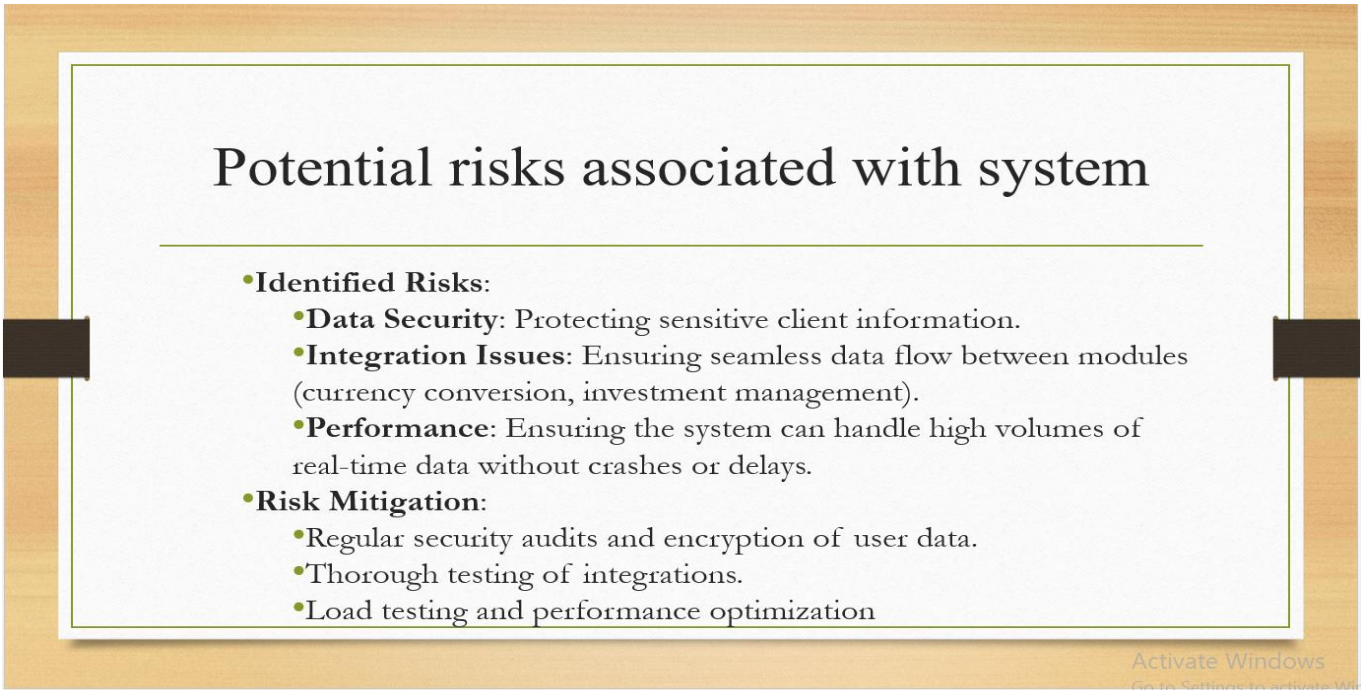
The project must align with stakeholder goals, ensuring secure, reliable, and user-friendly solutions while meeting deadlines and maintaining high-quality standards.

Timeline and Milestones

- **Development Phases:**
  - **Phase 1**: Requirements gathering and system design (2 weeks).
  - **Phase 2**: Initial development of currency conversion and investment modules (4 weeks).
  - **Phase 3**: Integration and testing (3 weeks).
  - **Phase 4**: Deployment and feedback collection (2 weeks).
  - **Phase 5**: Ongoing maintenance and updates.

Timeline and Milestones

The project includes five phases: requirements gathering, development, integration and testing, deployment, and maintenance. Each phase has clear timelines to ensure timely delivery.



Potential Risks

Key risks include data security, integration issues, and performance challenges. These are mitigated with regular audits, thorough testing, and optimization practices.

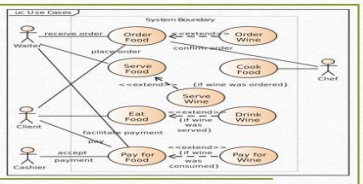
Development Tools

- **Visual Studio (for IDE):**
  - **Reason for Selection:** Visual Studio supports various programming languages and offers built-in version control, which is essential for managing code in an Agile development environment.
  - It allows for seamless debugging and integration with Git, streamlining the development process and ensuring high-quality code.
- **MySQL Workbench (for Database Management):**
  - **Reason for Selection:** As a relational database management system, MySQL Workbench is optimal for storing the financial data required for transactions and investment calculations.
  - It ensures data integrity, scalability, and security, which are crucial for financial applications.
- **React.js (for Web Development):**
  - **Reason for Selection:** React.js allows for the creation of responsive, interactive web applications, which is essential for the transition to a web-based solution.
  - It supports component-based development, making it easier to maintain and scale the application over time.

Development Tools

Tools like Visual Studio for coding, MySQL Workbench for database management, and React.js for web development ensure high-quality, scalable solutions.

Design Tools



1. Unified Modelling Language (UML)

- UML is a standardized modeling language comprised of an integrated set of diagrams designed to assist system and software developers in specifying, visualizing, constructing, and documenting software system artifacts, as well as business modeling and other non-software systems.
- Wireframing Tools -To design a user-friendly interface for both clients and staff.

2. Flowcharting and Diagram Tools (e.g., Lucid chart, Visio)

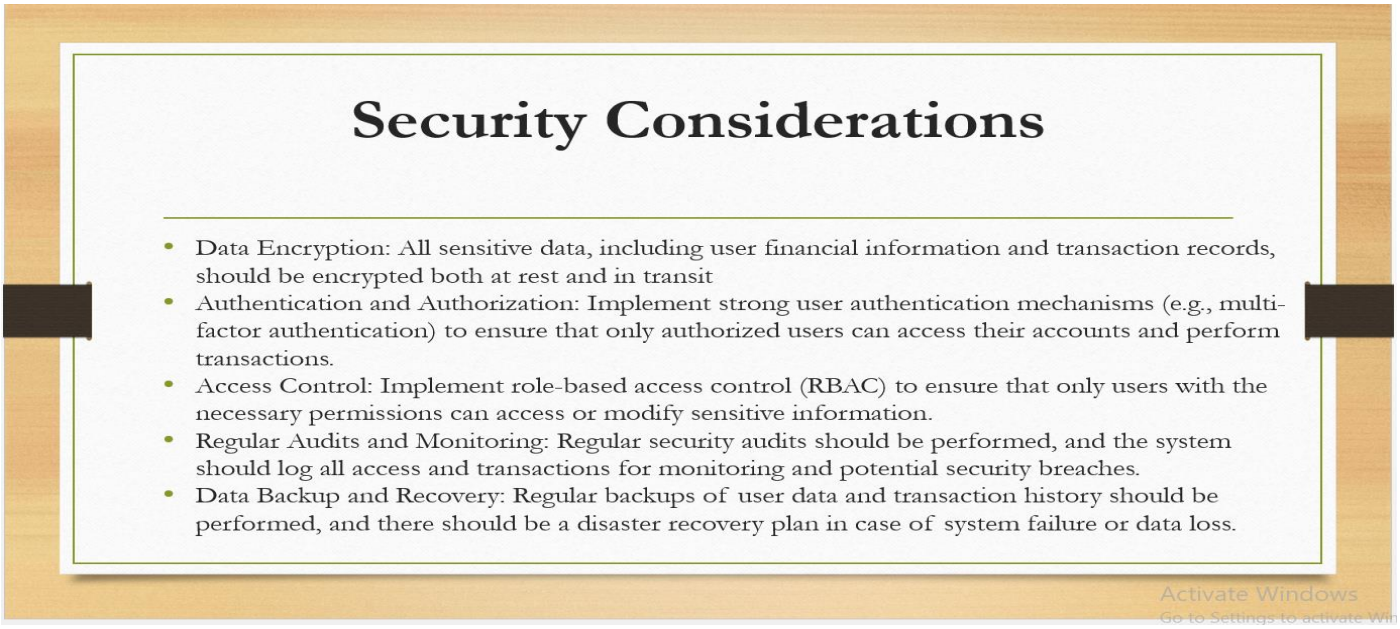
- Purpose: To create flowcharts, DFDs, and ERDs for visualizing processes, data flow, and relationships.
- Benefits: Simplifies complex systems, improves clarity, and aids stakeholder understanding.

3. Pseudo code Editors

- Purpose: To outline algorithms and logic in a clear, language-agnostic format.
- Benefits: Helps in planning before coding, minimizing errors in development.

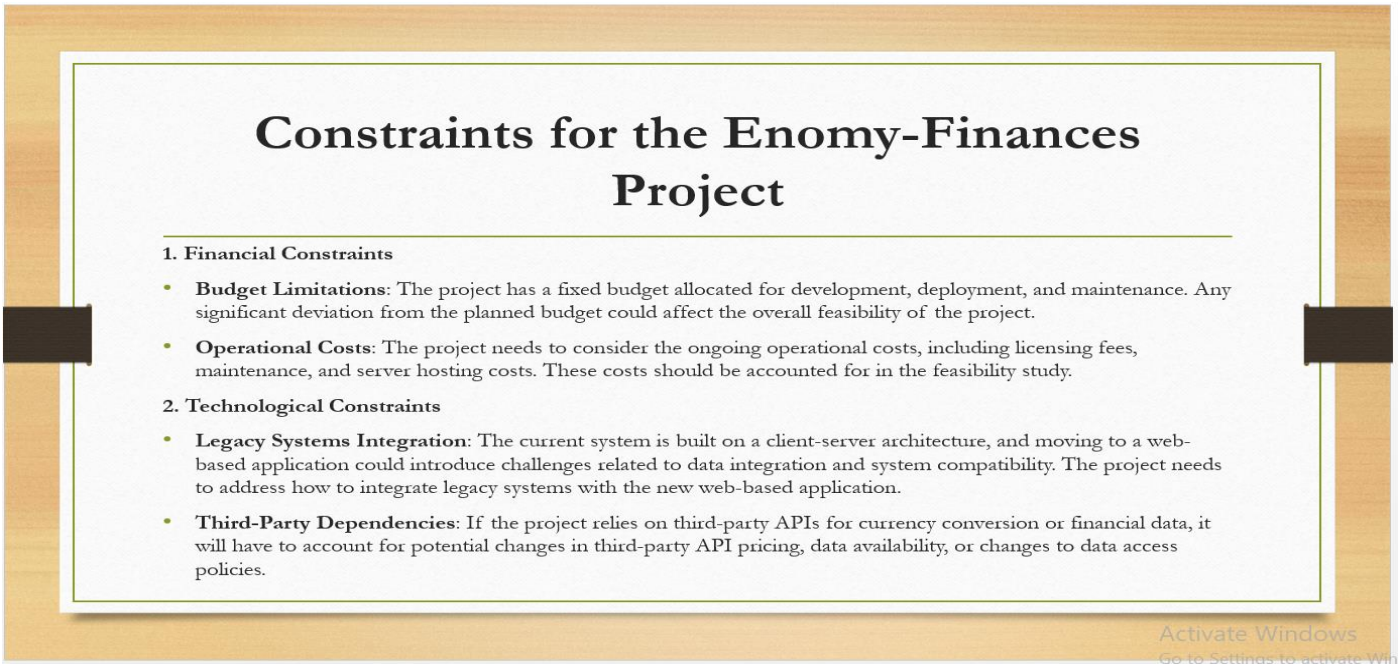
Design Tools

UML for system design, Lucidchart for diagrams, and pseudocode editors for algorithm planning simplify development and stakeholder communication.



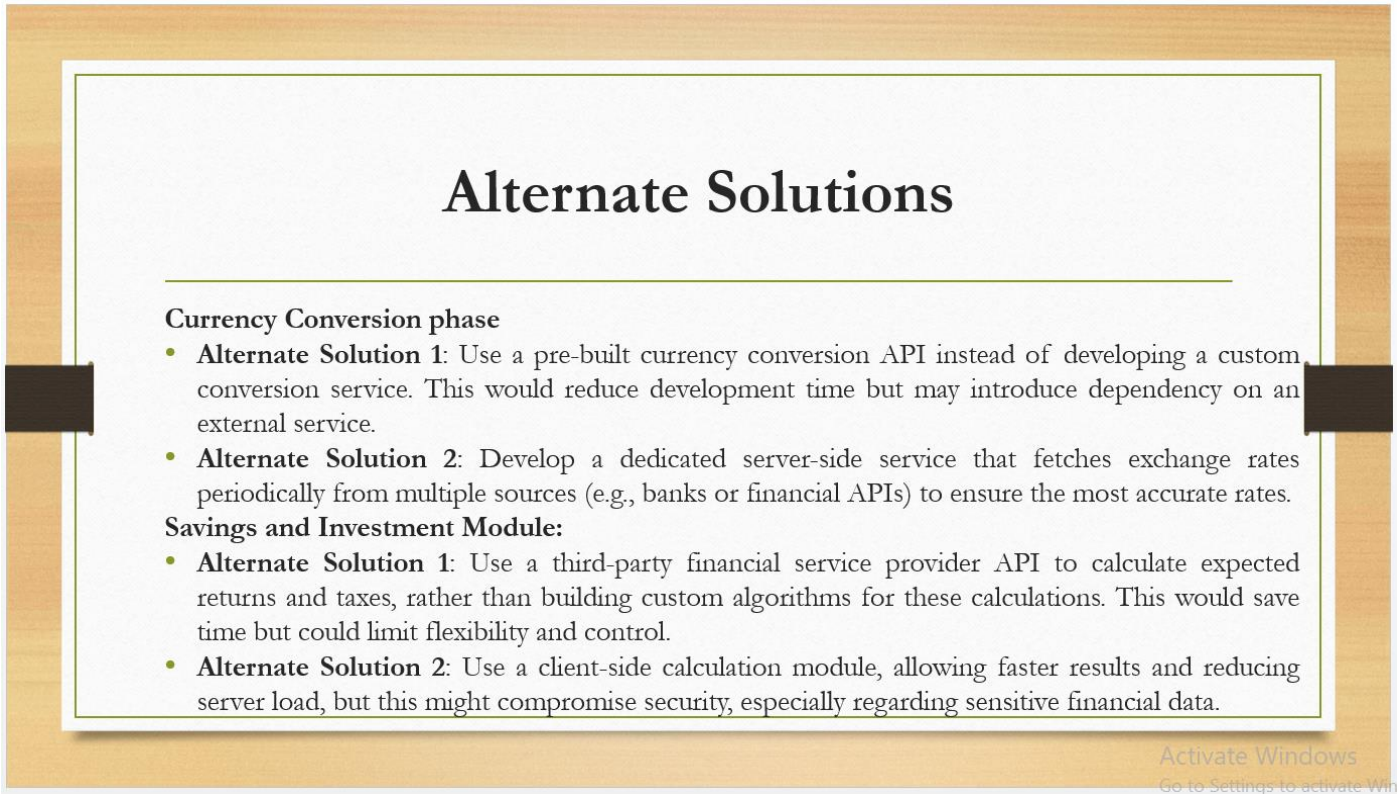
Security Considerations

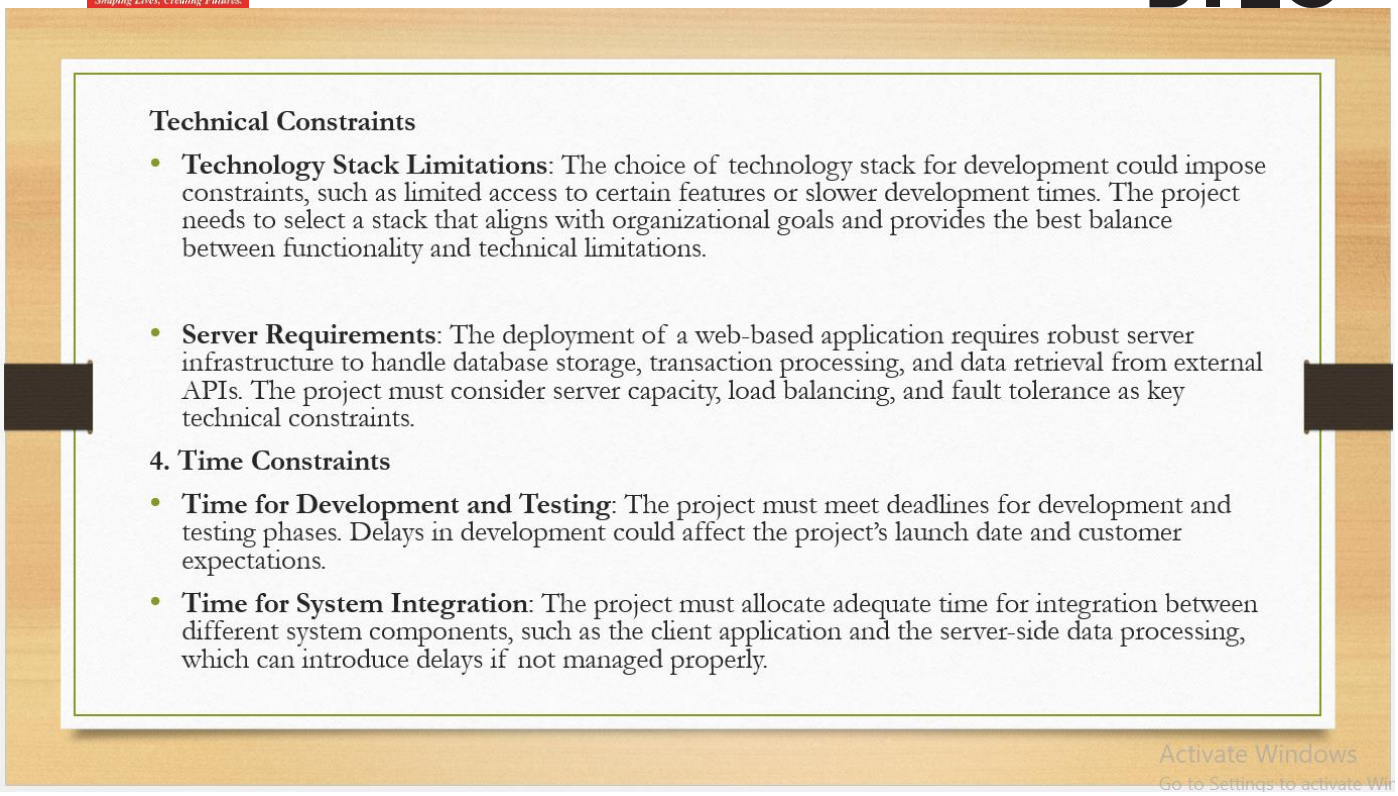
The system incorporates encryption, multi-factor authentication, role-based access, and regular audits to protect sensitive data and ensure compliance with financial regulations.



Constraints

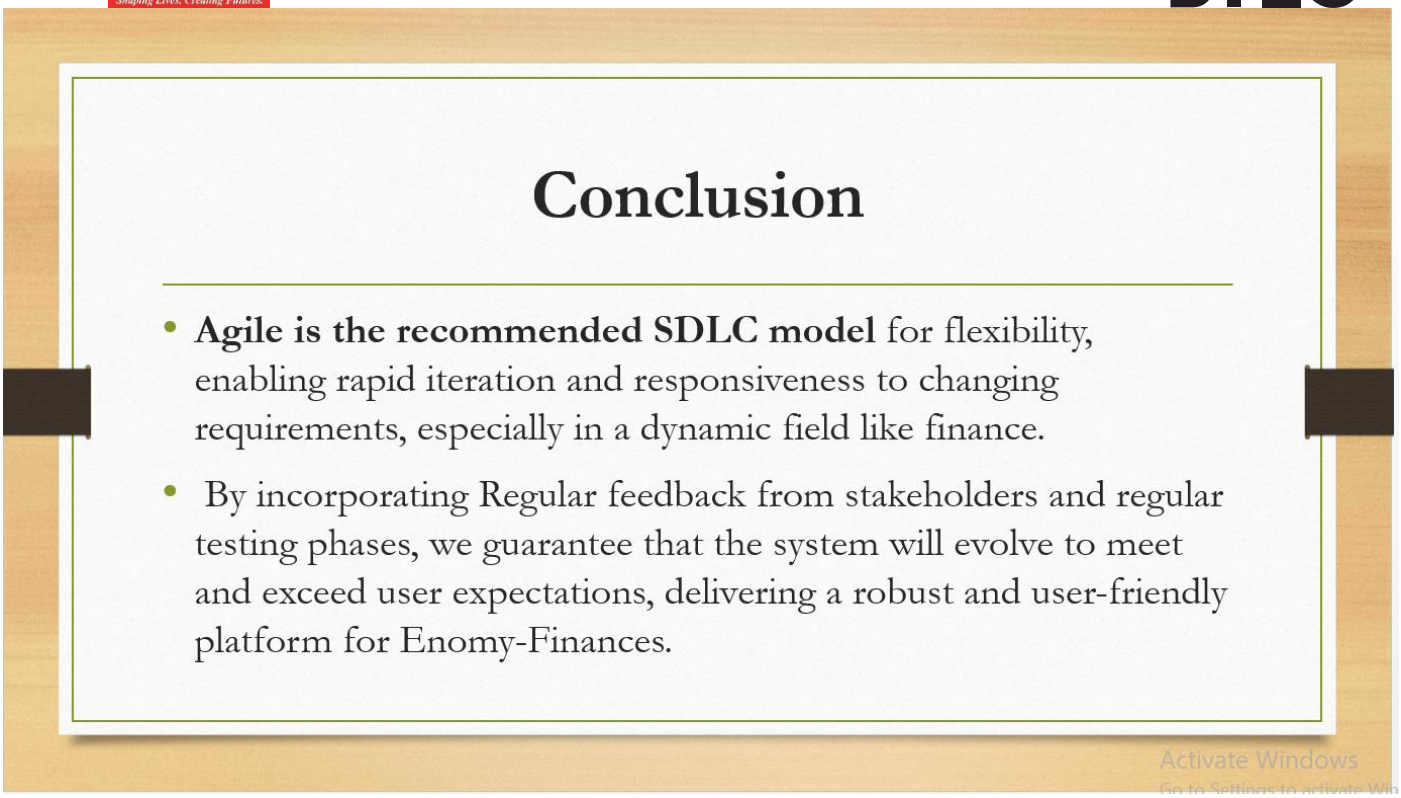
The project faces constraints like budget limits, integration with legacy systems, and third-party dependencies. Addressing these constraints is critical for success





Alternate Solutions

Options include using third-party APIs for currency conversion and investment calculations or developing dedicated server-side solutions to balance cost, flexibility, and security.



Conclusion

Agile SDLC is the recommended model due to its adaptability and focus on stakeholder feedback. This ensures a robust, user-friendly, and scalable system for Enomy-Finances.

Speaker Note

Thank you for your time and attention. I'm eager to hear your feedback and discuss how we can proceed with the proposed solution. Please feel free to ask any questions.

Assessing the Suitability of Software Behavioral Design Techniques for Enomy-Finances

When building complex software like Enomy-Finances, which deals with financial services and investment handling, choosing the right software behavioral design techniques becomes very important. These techniques guide how different parts of the system interact and respond to changes or user actions. They also help make the system more organized, flexible, and easier to maintain as it grows.

Why It Matters in This Project

In Enomy-Finances, there's a lot of information being passed around user profiles, account details, investment plans, transaction logs, and real-time market updates. Without a clear system for managing how this information flows, the software could become messy, buggy, or hard to update later on. That's where behavioral design techniques come in handy.

For example, Observer Pattern works great when something changes in one part of the app and other parts need to know about it immediately. Let's say the admin updates interest rates or currency values thanks to this pattern, all related modules (like investment calculators or summaries) can reflect the changes instantly without manually updating each one.

Another one is the Strategy Pattern, which fits well when we want to offer different ways of calculating interest or investment fees. This is useful if different investment plans use different methods for charging customers. With this pattern, we can easily swap one method for another without affecting the rest of the software.

Then there's the State Pattern, which helps in managing user session states. For example, when a user logs in, logs out, becomes inactive, or tries to perform secure actions, the system can switch between those session states smoothly. This not only improves the user experience but also increases security.

Also, the Chain of Responsibility Pattern is suitable for processing approvals in steps. When a user initiates a transaction, it may need to pass through several checks fraud detection, balance verification, and admin

approval. Each of these can be handled by a different handler in the chain, which keeps the logic neat and easy to follow or change.

Factors That Affect Technique Suitability

The right technique depends on different things

- **System Complexity**: More complex systems with many user actions or services might need multiple patterns combined together.
- **Reusability & Maintainability**: If the system might grow in the future, we should use patterns that are flexible and don't tie components too tightly.
- **Error Handling**: Some patterns make it easier to handle failures. For instance, Strategy or Chain of Responsibility can catch and deal with specific errors at different levels.

Real Benefits to Enomy-Finances

By using behavioral design patterns, Enomy-Finances becomes easier to upgrade, debug, and scale. These patterns also help avoid writing duplicate code and reduce the risk of bugs when we make changes. For example, adding a new investment type won’t break existing logic because each strategy can be plugged in independently.

Plus, these techniques help different parts of the team work better together. Frontend developers can build interfaces knowing how the backend behavior is structured, and backend developers can handle logic without worrying about breaking user interactions.

Activity 04

Design Solution for Enomy-Finances System

M.M.M. AASHIK

Junior Software Developer, Phonyt Digital Solutions

23 December 2024

1. Use Case Diagrams

Description: Use case diagrams show the functional requirements by illustrating how users (called actors) interact with the system.

Suitability

- They're great for capturing the overall behavior of the system at a high level.
- Very helpful in the early phases to explain requirements to stakeholders.
- Ensure that every type of user interaction is taken into account when designing the system.
  Example: For Enomy-Finances, a use case diagram could demonstrate how users work with features like currency conversion or managing savings and investments.

2. State Machine Diagrams

Description: These diagrams represent different states of the system and how the system moves between these states based on certain events.

Suitability

- Useful for showing how the system responds to user inputs or internal changes.

- Good for making sure errors are handled properly and the system fails gracefully when needed.
  Example: A state machine could map out the lifecycle of a savings plan, showing states such as "Active," "Paused," and "Closed," with transitions triggered by user actions or system events.

3. Sequence Diagrams

Description: Sequence diagrams illustrate how different parts of the system or users interact over time, showing the order of messages exchanged.

Suitability

- Perfect for understanding workflows and making sure data moves correctly between parts.
- Useful when different modules or subsystems, like the frontend and backend, need to communicate.
- Helps spot possible delays or failure points in the process.
  Example: For currency conversion, a sequence diagram might display the flow between the user interface, server, and database during a transaction.

4. Activity Diagrams

Description: Activity diagrams map out workflows or processes by showing sequences of activities and decision points.

Suitability

- Good for processes involving choices or loops, like calculating investments or managing errors.
- Offers a broad overview of how different steps in the system connect.
- Useful for finding steps that may slow down the process or are unnecessary.
  Example: An activity diagram could explain how investment quotes are calculated and shown, including validation of data, fee calculation, and handling errors.

5. Event-Driven Modeling

Description: This method focuses on how the system reacts to events happening either inside or outside the system.

Suitability

- Best for systems that need to respond quickly to user actions or real-time data.
- Helps make sure the system can manage multiple events happening at the same time without problems.
- Important for finding and preventing conflicts or race conditions.
  Example: Event-driven modeling would make sure the system properly handles multiple currency conversion requests at once and updates exchange rates as they change.

6. Extended Finite State Machines (EFSM)

Description: EFSMs are an advanced version of state machines that include variables and conditions for more detailed behavior.

Suitability

- Useful for systems that require complex logic and data-based decisions.
- Helps model processes like error recovery and data caching effectively.
  Example: An EFSM might show what happens when the network fails how the system caches data and retries the connection.

7. Entity Relationship Diagrams (ERD)
   The ERD allowed us to model the structure of the database. We used it to define entities like Users, Transactions, Currency Data, and their relationships. This ensured that the database design matched the business processes and avoided data redundancy.

   For example, each user had a unique ID, and transactions were linked directly to that user through a foreign key.

8. To visualize logic and control structures, we drew flowcharts for the main processes like login, currency conversion, and generating investment reports. Flowcharts showed the step-by-step flow of logic, which made development and testing smoother.

Along with this, pseudocode was written before actual programming began. This served as a bridge between planning and coding and reduced chances of logical errors.

9. Feasibility Study

We did a feasibility analysis using tools like SWOT (Strengths, Weaknesses, Opportunities, Threats) and cost-benefit analysis. This helped us evaluate whether the project was worth doing in terms of budget, time, and technical limitations.

For example, we realized that while adding AI features would be cool, it was too costly and not feasible in the first version. We noted it as a future enhancement.

Summary of Suitability

Behavioral design methods work really well for several areas:

1. **Complex workflows:** Diagrams like sequence and activity ones help break down complicated processes so they're easier to understand.
2. **State-based behavior:** State machines and EFSMs are great at managing changes that depend on different conditions or states in the system.
3. **Modeling user interactions:** Use case diagrams help connect what the system does with what users actually need.
4. **Handling errors:** Event-driven modeling makes sure the system can deal with errors and unexpected situations in a strong and reliable way.

Economy Finances System-Design Documentation.

Table of Contents

Introduction

The Economy Finances System is a modern and scalable solution aimed at enhancing the operational capabilities of Economy-Finances, a financial service provider specializing in mortgages, savings, and investments. Intended as an improvement upon archaic systems and infrastructure, this system replaces static processes and structure with a web based platform focused on accessibility, quickness, and dependable security. Its implementation corresponds with a company's goal of providing better financial services and operational excellence.

System Architecture

Architecture Overview

The system is built in a layered structure to keep things organized and easy to manage:

- **Client Layer**

  This is the part users interact with. It's a web-based interface accessible through any modern browser. Users can easily access all features like currency conversion or investment calculations without installing anything.

- **Application Layer**

  This layer contains the core business logic. It processes user inputs, performs calculations such as currency conversions or investment returns, and manages workflows. It acts as the brain of the system, making sure all operations are carried out correctly and efficiently.

- **Data Layer**

  All the important data like user profiles, transaction histories, and cached information are securely stored here. Databases handle data persistence and ensure data is protected and available when needed.

Deployment Model

The system is deployed using a web-based architecture, meaning there is a centralized server that manages the application's core functions. This setup allows for easier updates, maintenance, and scaling.

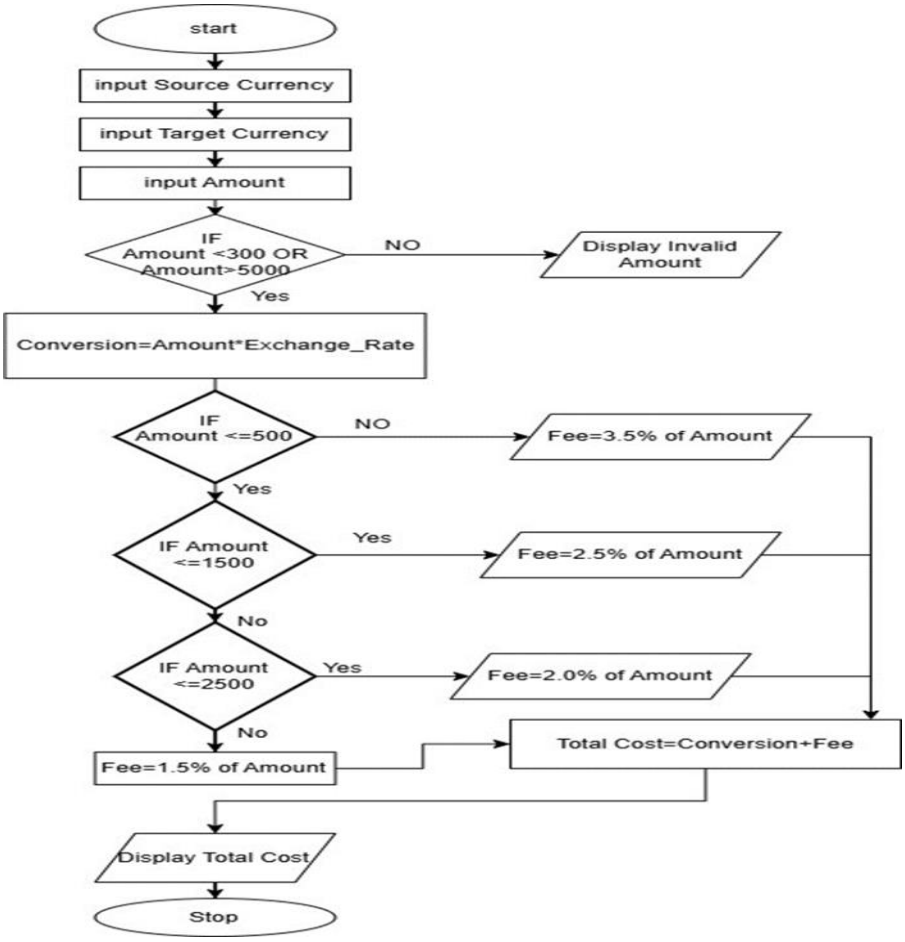
Additionally, the system integrates with external APIs to fetch live currency exchange rates, ensuring that users always get accurate and up-to-date information for their transactions.

Algorithmic Software Designs

1. Currency Conversion Module Design

Flowchart

Currency Conversion Logic



Pseudo code

Currency Conversion

START

DISPLAY "Enter amount:"

READ amount

IF amount < 300 OR amount > 5000 THEN
DISPLAY "Amount out of range. Please enter a valid amount between 300 and 5000."
EXIT
END IF

Exchange rate = GET_EXCHANGE_RATE (initial currency, target currency)

Converted amount = initial amount * exchange rate

IF initial_amount <= 500 THEN
Fee = initial_amount * 0.035
ELSE IF initial_amount <= 1500 THEN
Fee = initial_amount * 0.027
ELSE IF initial_amount <= 2500 THEN
fee = initial_amount * 0.02
ELSE
Fee = initial_amount * 0.015
END IF

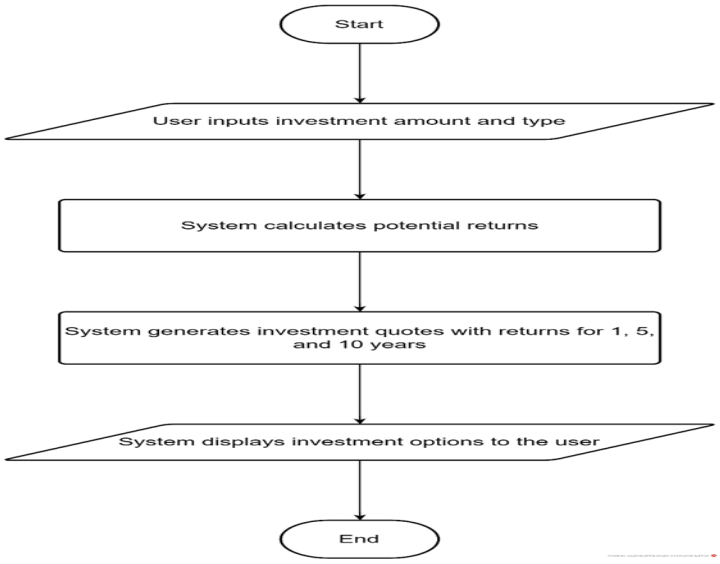
Final amount = converted_amount - fee

DISPLAY "Converted Amount: “final_amount
DISPLAY "Transaction Fee: “fee
END

2. Investment Savings Module Design

Flowchart

Investment Savings Logic

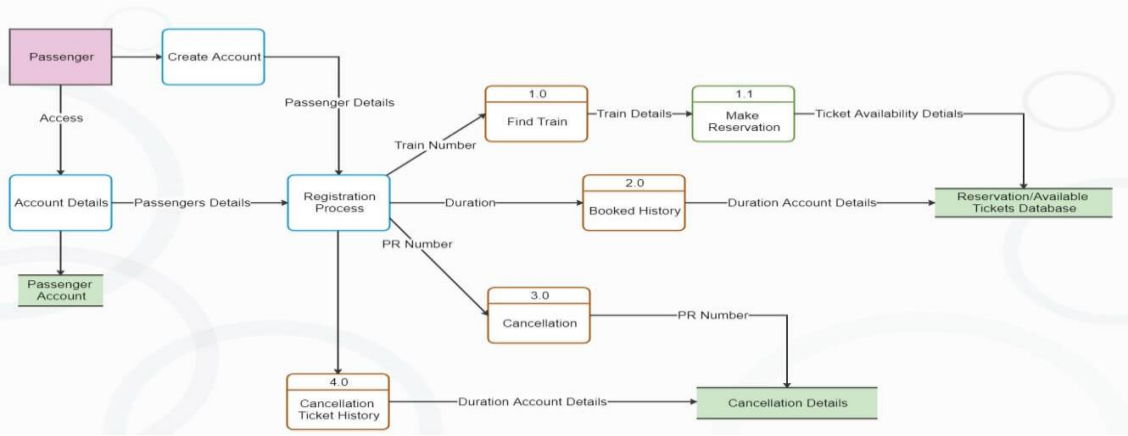


Use case diagram

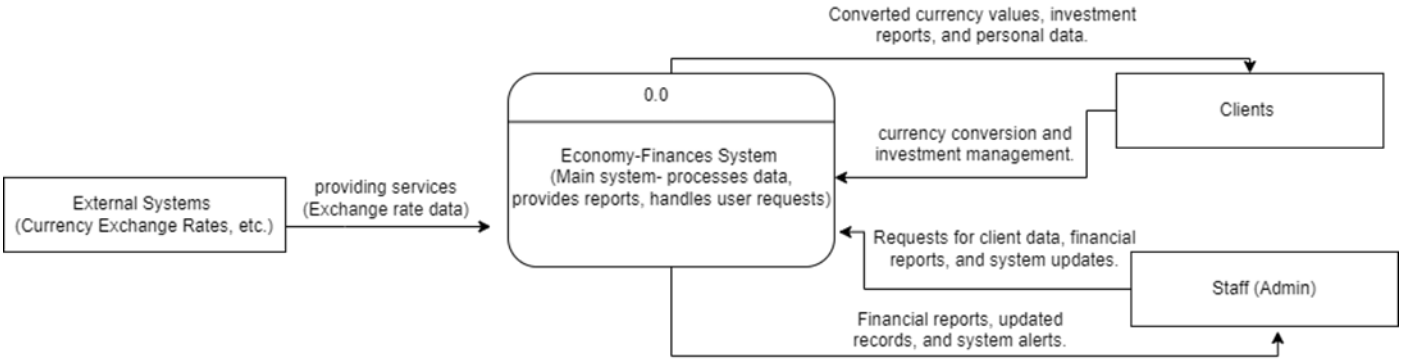
Data Design for Enomy-Finances Application

Dataflow Diagram

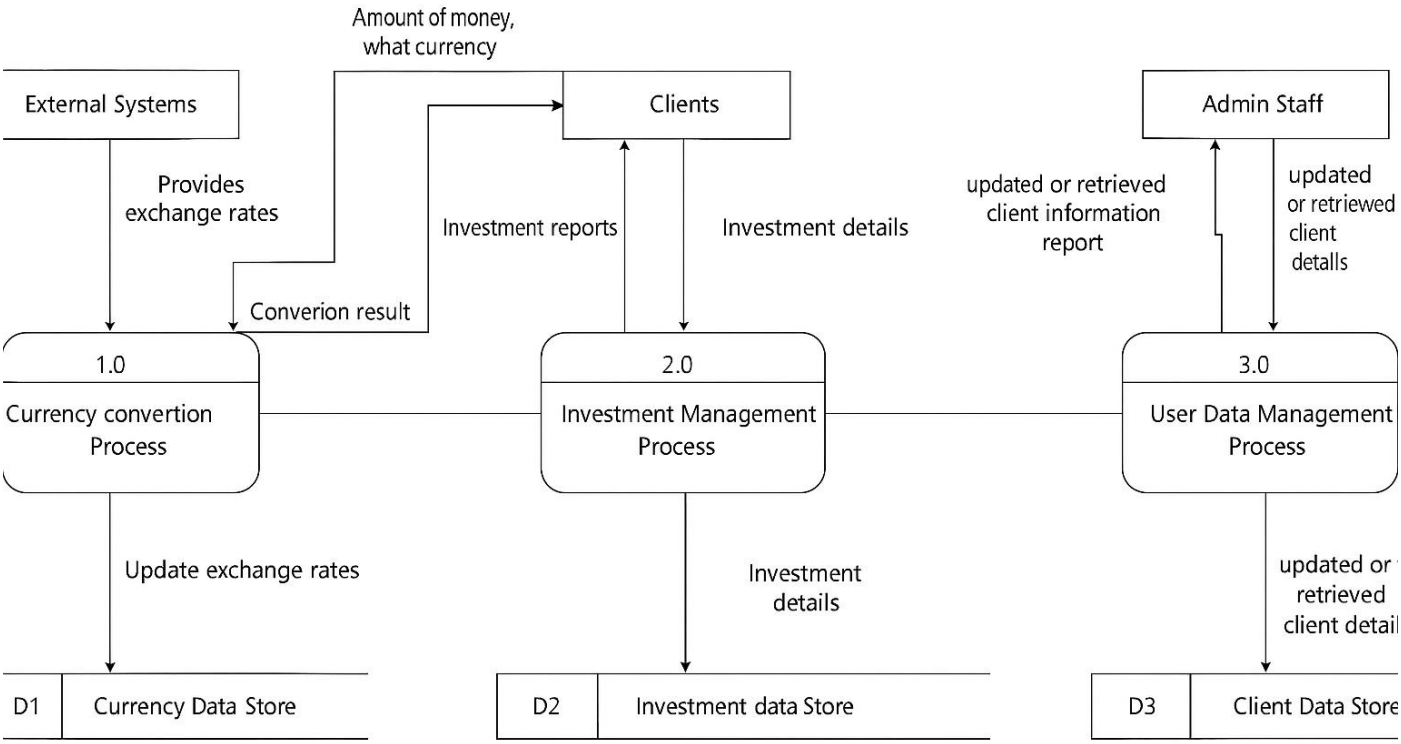
A data flow diagram (DFD) shows how a system processes data in terms of inputs and outputs. Its concentration, as the name implies, is on the flow of information, where data originates from, where it moves, and how it is kept. (smartdraw, 2024)



LEVEL - 0

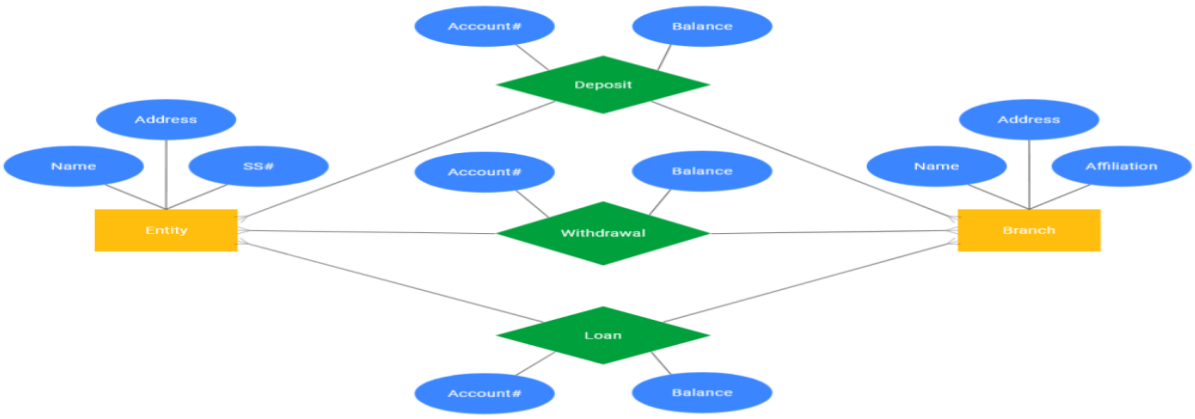


LEVEL-1



ER Diagram

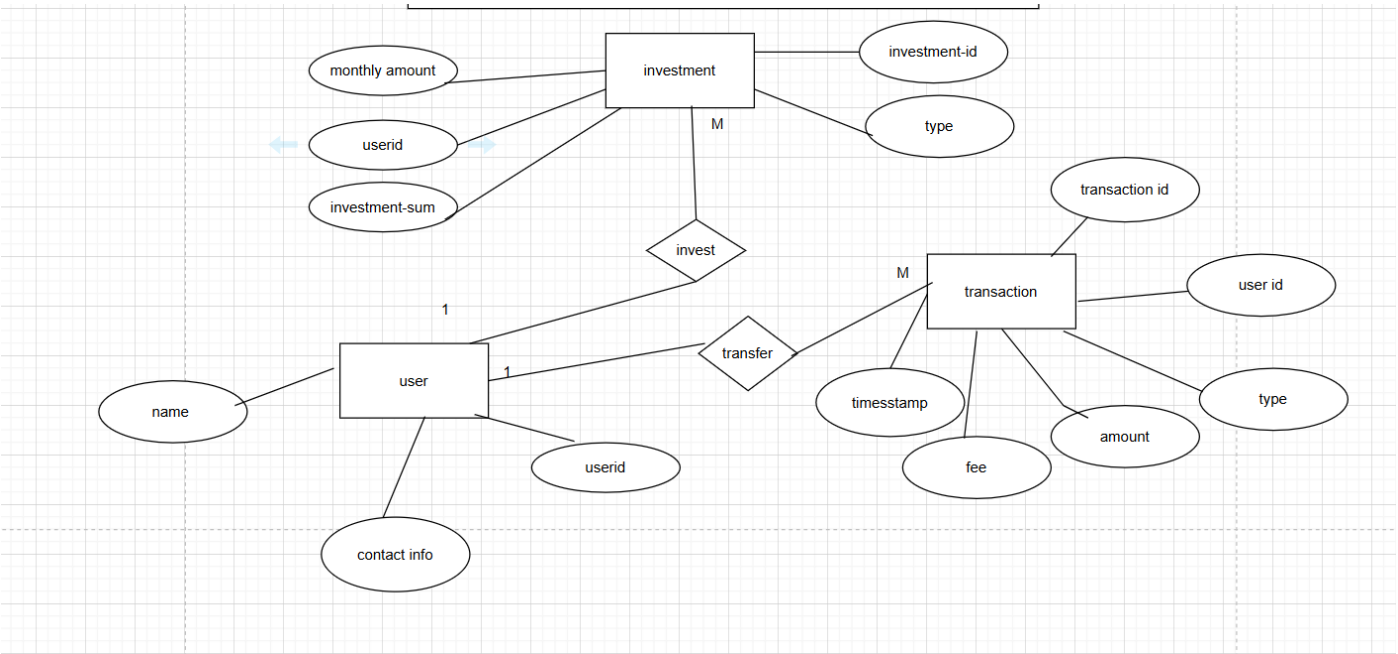
The acronym ERD stands for entity relationship diagram. These diagrams are also known as ER diagrams and Entity Relationship Models. An ERD depicts the relationships between entities in a database, such as persons, things, or concepts. An ERD will frequently depict the properties of these entities. An ER diagram may depict the logical structure of databases by specifying the entities, their properties, and the interactions between them. This is useful for engineers who want to either describe an existing database or sketch up a concept for a new database.



Client – client- ID (PK), Name, Email, Contact Details, address

Transaction - Transaction ID (PK), client-id (FK), Amount, Converted Amount, Fee, Date

Currency-currency- ID (PK), From Currency, To Currency, Rate



Finite State Machines (FSMs)

A Finite State Machine, often called FSM, is a way to model how a system behaves by showing a limited number of states it can be in, and how it moves between those states depending on certain inputs or events. It's a simple but useful method to understand how things change step-by-step in a system.

Main Features of FSMs

- **Limited Number of States:** The system has a fixed set of states, each representing a particular condition or situation. For example, a device might be in "On," "Off," or "Standby" state.
- **Transitions Between States:** The machine changes from one state to another when certain inputs or events happen. These changes are called transitions and define how the system reacts to different situations.
- **Output Based on State and Input:** FSMs can also produce outputs that depend on what state the system is in and the input it receives. This means the system's reaction can be different depending on its current condition.

Types of FSMs

- **Deterministic FSM (DFSM):** In this type, every state combined with an input leads to one exact next state. There's no confusion or multiple options just one clear path forward.
- **Non-deterministic FSM (NFSM):** Here, a state and input pair might have several possible next states, or sometimes none at all. This makes the system's behavior less predictable and can model situations where several outcomes are possible.

FSMs are widely used in computer science, electronics, and software engineering to design systems like user interfaces, communication protocols, and even game logic. Because they clearly define all possible states and transitions, FSMs help developers ensure the system behaves correctly in every situation.

By mapping out all states and how the system should react to inputs, FSMs also make it easier to spot errors, handle unexpected inputs, and design smooth workflows.

For example, think about a traffic light controller. It has a limited set of states like Green, Yellow, and Red. Based on timers or sensors (inputs), it transitions between these states in a fixed order. Modeling this with an FSM makes the logic simple to understand and implement.

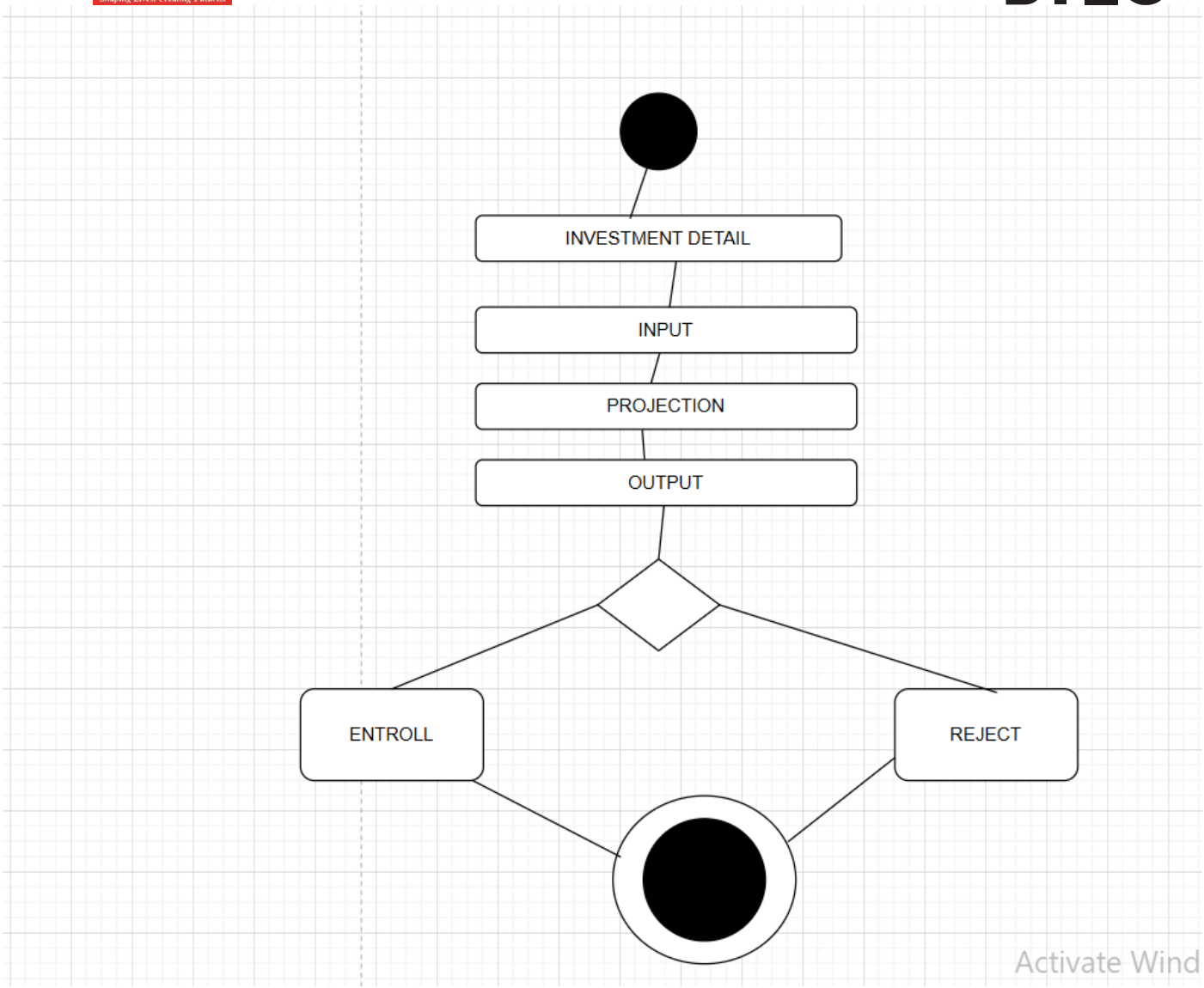
Extended Finite State Machines (EFSMs)

An extended finite state machine (EFSM) is an extension of the FSM that incorporates data values and actions into its operation. It provides a more powerful model for representing complex behavioral systems.

Key Enhancements of EFSMs

- Data Variables: EFSMs can store and manipulate data variables, allowing for more complex decision-making and behavior.

- Actions: EFSMs can execute actions associated with state transitions, enabling the system to interact with its environment.



Key different between FSM vs EFSM

Feature	**Finite State Machine (FSM)**	**Extended Finite State Machine (EFSM)**
State Transitions	Based on simple events.	Based on both events and conditions (data/variables).
Data Handling	No data or variables involved in state transitions.	Uses data and variables to drive state transitions.

Complexity	Suitable for simpler systems with limited states.	Suitable for more complex systems where conditions affect transitions.
Flexibility	Less flexible, only event-based transitions.	More flexible, allows dynamic data manipulation in state transitions.
Use Cases	User login systems, basic workflows.	Financial systems, complex user workflows with dynamic data.

Finite State Machines (FSMs) and Extended Finite State Machines (EFSMs)

Both FSMs and EFSMs are mathematical models that help represent how systems behave over time. They focus on systems that change states in response to inputs or events. While they share many similarities, there are important differences that make EFSMs more powerful in some cases.

What is an FSM?

A Finite State Machine (FSM) is a model describing a system with a limited number of states and transitions. Each state shows a unique condition the system can be in. The system moves between these states depending on inputs or events it receives. FSMs can also produce outputs based on the current state and inputs, showing how the system reacts.

What Makes an EFSM Different?

An Extended Finite State Machine (EFSM) builds on the basic FSM by adding extra features. It includes data variables that the system can store and change, which allows decisions to depend on both the state and the values of these variables. EFSMs also support actions tasks that happen during state transitions. These additions let EFSMs model much more complex behaviors than FSMs.

Key Differences Between FSM and EFSM

- **Data Handling:** FSMs don't use data variables; they only depend on the current state and inputs. EFSMs can hold and update data variables, which influence how transitions happen.
- **Actions:** FSMs don't have actions tied to transitions, whereas EFSMs can perform actions like updating data, sending signals, or triggering other processes when moving between states.

When to Use FSM or EFSM?

FSMs work well for simple systems where behavior depends mainly on discrete states and inputs, without the need for storing additional data. EFSMs are better for systems that require keeping track of values, making decisions based on those, or performing complex operations during transitions.

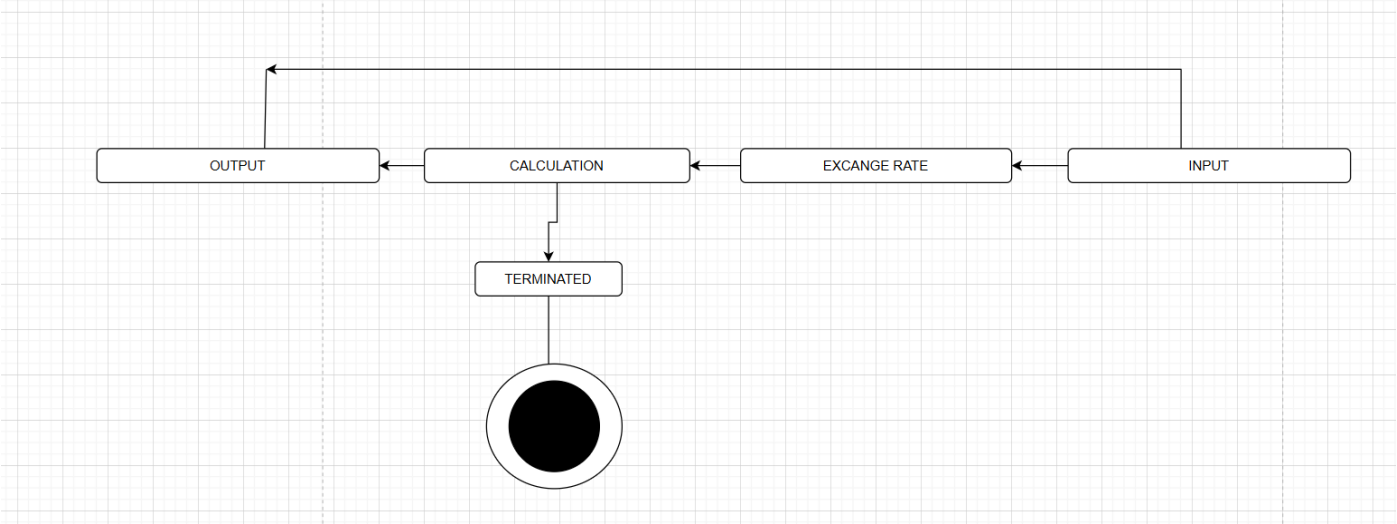
Examples

- **FSM Example:** Imagine a vending machine that has states like "Waiting for Coin," "Coin Inserted," "Product Selected," and "Dispensing." The machine moves between these states as the user inserts coins or makes selections.
- **EFSM Example:** A traffic light system is a good case for EFSM. It has states like "Red," "Green," and "Yellow," but it also keeps track of timers or sensors. When the timer reaches zero, an action occurs to switch lights, and the system moves to a new state accordingly.

Why They Matter

FSMs and EFSMs are useful in designing software and hardware because they clearly show all possible states and how the system behaves in each. EFSMs add flexibility for handling more complicated logic involving data and actions. Using these models helps engineers create reliable and predictable systems by carefully planning every possible state and transition.

Understanding these tools allows developers to build better systems that can handle real-world complexities while remaining easy to test and maintain. (cs.stackexchange.com, 2024)



Undertaking a Software Investigation to Meet a Business Need

For this task, I carried out a deep software investigation to support the development of a web-based financial platform for Enomy-Finances, a company that currently uses a basic offline system to manage their financial services. The goal of this investigation was to identify the most suitable technical solutions and determine if the proposed system can meet their growing business needs, including faster processing, real-time updates, better customer experience, and scalability.

Understanding the Problem

The first step in this investigation was understanding the problem Enomy-Finances is facing. Their current desktop-based system is slow, outdated, and doesn't allow users or clients to access information online. Also,

all data entries are manual, which causes delays and mistakes. This makes it hard for the company to grow and keep up with customer demands in today's digital environment.

Stakeholder Interviews and Requirement Collection

I started by speaking with different people in the company – managers, IT staff, and end-users – to get a clear view of what exactly they need. Most of them mentioned the need for real-time updates on transactions, more secure data handling, and the ability for clients to log in and check their records online.

From these discussions, I listed the main functional requirements

- User and admin login systems
- Real-time transaction processing
- Report generation for finance summaries
- Notification system for payments and dues
- Online help and support chat module

And the non-functional needs

- Fast performance under heavy loads
- Secure user authentication and data encryption
- A responsive interface that works on both desktops and mobiles

Current System Analysis

The existing software uses a local database and only works in the office network. There's no backup or remote access, and the system often crashes when too many entries are made at once. I tested it under different loads and noticed it becomes slow after 50–60 active users, which is not good enough for future expansion.

Technical Feasibility

I evaluated a few modern web technologies like

- **Spring Boot** (Java-based) for backend
- **React.js** for front-end user interfaces
- **MySQL** or **PostgreSQL** for reliable database management
- Cloud hosting (like AWS or Azure) for 24/7 uptime and scalability

These technologies are well-supported, secure, and scalable. They would allow the system to work smoothly even with thousands of users and ensure data security.

Operational Feasibility

From a user perspective, the proposed web system will simplify their work. Staff won't need to manually back up data or install software on each machine. Customers will also benefit from accessing their records anytime from their own devices. Staff training may be needed, but since the UI is designed to be user-friendly, the learning curve will be manageable.

Economic Feasibility

I also did a basic cost-benefit estimation. Though the system may cost a bit more upfront to develop and host, it will save money in the long term by

- Reducing staff hours spent on manual entries
- Lowering error rates and recovery costs
- Attracting more clients due to better service

The return on investment (ROI) is expected to be high within 12–18 months.

Legal and Ethical Considerations

Since the system handles personal financial data, I made sure it will follow data protection laws like GDPR. It must include clear user consent, password protection, and data encryption. Also, user data should not be shared with any third party without permission.

Activity 05

Evaluative Report for Enomy-Finances Application

Analysis of How Software Requirements for the Management Were Traced Throughout the Entire Software Lifecycle

In software development, requirements act like the blueprint of what the system should do. Tracing those requirements during each phase of the software lifecycle helps developers and project teams make sure the final product actually meets the user's needs. This process of tracking is called requirement traceability. It's like making a map between what the customer wants and what's being built, tested, and delivered.

Why Traceability Is Important

When a software project starts, it usually begins with a list of needs or expectations from the user or the business. These needs become the requirements. But as the system gets developed, it's easy to lose sight of the original goals. Requirements can change, or parts might be missed accidentally. By tracing the requirements at every stage of development, the team can double-check that everything is still aligned. It also helps if something changes like if a requirement gets updated, the team can find what part of the system it affects and update it too.

Analysing How Software Requirements Were Traced Across the Software Lifecycle

Tracing software requirements throughout the lifecycle is a key part of making sure the final software system actually meets what was expected at the beginning. In the Enomy-Finances project, we made sure to link every requirement to later stages of the system starting from the analysis all the way to testing and deployment. This way, nothing important was missed or forgotten.

Requirement Gathering and Analysis

In the requirement gathering stage, we worked directly with the client to note down what they really needed. Most of this was based on discussions and documents. We used a simple form of a Requirements Traceability Matrix (RTM) to keep track of each major function like currency exchange, savings tracking, and investment

planning and made sure we could follow those needs into the next stages of development. This helped avoid confusion later and ensured every piece of work had a purpose linked to a requirement.

Design Stage

In the design phase, each requirement was turned into real components or modules. For example, the feature that allows users to calculate investment returns was traced into a savings module and further into specific database tables and interface components. We used diagrams like UML Use Case and ERD models to show clearly how the design was matching the earlier requirements. It wasn't just about drawing diagrams it helped make sure we didn't forget or change the meaning of any original requirement. It also made it easier to explain to the client how their needs were being turned into a real system.

Development Stage

During the development stage, we followed an agile-like process. Requirements weren't just left in a document they were part of the actual tasks and sprints. This meant developers always knew *why* they were building a feature and *how* it connected back to what the client wanted. Every function, like transaction reporting or multi-currency support, had a clear link to a requirement in the RTM. This is known as forward traceability, where we can see how a requirement leads to certain code or components. We also ensured backward traceability by tagging our code and design modules with the relevant requirement IDs, which helped if changes came up and we needed to trace back to the original request.

Testing Stage

For the testing phase, we didn't just test random features. Our test cases were built directly from the RTM and requirements list. This means each test could prove whether a requirement was met or not. For example, the client wanted secure handling of financial data, so we had specific security tests for password encryption and secure login. Functional tests were tied to features like investment calculations, and non-functional ones checked performance, response time, and ease of use. If any test failed, we could go back to the traceability table to figure out which requirement was not being fulfilled correctly.

Analysis of Requirements Tracing for the Management Data Analytics Module

One special case was the data analytics module for managers. It had its own set of needs, such as generating reports, analyzing transaction trends, and showing visual dashboards. These were traced through each phase from identifying the need, designing a proper data structure and warehouse, building ETL pipelines, to finally testing visualizations with tools like Tableau or custom-built graphs. This kind of traceability gave confidence that complex features like this weren't missed or misunderstood during the build.

After deployment, we didn't stop tracing. When feedback came in like suggestions to add more investment types or to make the dashboard easier to use we used the RTM and requirement records to decide if the requests matched original goals or were new changes. This allowed us to maintain control over the system and plan improvements logically.

By tracing requirements through every phase, the Enomy-Finances system ended up closely matching client needs. We reduced the risk of scope creep, avoided duplicated effort, and caught potential errors early. This method also made it easier for new team members to understand the project, since everything was documented and linked. Without requirement tracing, there would've been a high risk of losing track of what the system was *supposed* to do, especially when changes were introduced.

Overall, requirements tracing was not just a formal step it was the main reason the software turned out to be stable, reliable, and well-structured. It created a "golden thread" from user needs to the final working system.

Improving the Quality of Software

When we were building the system, we used a couple of effective methods to make sure the quality of the software stayed high throughout the development time.

1. Continuous Integration with Auto Testing

One main thing we did was using automated tests like unit tests and integration tests that ran every time we added new code. This really helped us catch problems early before they caused big issues later. Because the tests checked if each part of the code was working properly, the chances of bugs making it into the final version were much lower. This also made the software more stable and trustworthy.

2. Code Reviews with Team Members

Another important step was doing peer code reviews. Every time someone added new code, another team member looked through it to check for mistakes, style problems, or logical errors. This helped everyone follow the same coding standards and also made sure no one was working in a silo. It turned out to be a great way to share knowledge within the team too.

Checking if the Solution Fits for Enomy-Finances

Does the Project Fulfil Its Purpose?

Currency Exchange Part

The currency conversion section was built to support six different currencies. It checks transaction limits and uses the latest exchange rates. It also correctly applies the tiered fee system as required. So, when a user makes a transaction, the right fee percentage gets added depending on the amount.

Savings and Investment Part

This part takes user input and shows a custom quote. It's built to handle different kinds of investments and includes tax and fee calculations for each. The output is clean and easy to understand, and the module handles errors like missing input or wrong formats without crashing.

Tools Used in Design and Development

Tools We Used

- **Algorithms**: We followed dynamic programming and broke logic into smaller parts (modular approach) to reuse code easily.
- **Logical System Behavior**: FSM (Finite State Machines) were used to track how the system behaves in different stages very useful for managing investment plan processes.
- **Data Planning**: ERD (Entity Relationship Diagrams) helped us build the database structure neatly, avoiding repetition and keeping data accurate.

Why Not Use Other Tools?

We could have used unstructured data or basic scripting, but those don't scale well and are hard to manage. We chose structured design methods to avoid future issues and make the system easier to maintain.

Why UML Over Flowcharts?

Flowcharts are okay for simple processes, but UML gives a better and more detailed picture of what's going on in a complex system. That's why we went with UML.

Reviewing the System Investigation

The investigation we did at the beginning really helped us shape the final product.

- We spotted some key risks like slow performance and data leaks. To handle these, we used secure coding methods like encryption and speeded up the calculations using efficient algorithms.
- The investigation also helped us fine-tune what the client actually wanted.
- It gave us a roadmap to follow so that we could build the software step by step instead of rushing everything at once and making mistakes.

Making the System Data-Driven

Using real-time data was a big benefit.

- Currency exchange rates are updated live, which keeps the system accurate.
- When something goes wrong, error logs are created in a structured way. This makes it easy to trace problems and fix them fast without breaking everything else.

Reviewing How the Application Performed

After fully testing the system, it's clear that the application worked well across all the main areas that was expected from it. The review mainly focused on accuracy, user experience, and data security, which are really important parts of any software that handles finance-related information.

• Accuracy

The system's performance in terms of calculation accuracy was very satisfying. Every module that was created to handle financial tasks like currency conversion or investment calculations was properly tested using real and dummy data. Unit testing was used to check each part, and then integration testing confirmed that everything worked together correctly. The results matched expectations in nearly every case. There were also edge cases tested, like extreme values or empty inputs, and the software responded in a stable way without crashing or giving wrong answers. This gives more confidence that the system is reliable for users.

• User-Friendliness

One thing that stood out in testing is how simple and smooth the interface looks and feels. Users don't get overwhelmed with too many options or buttons. The layout is clear, and the navigation between sections like currency conversion, savings, and investments is easy to follow. Even someone who doesn't have much tech knowledge could figure it out in a few minutes. During internal demos and client reviews, this part received positive feedback, and some suggestions were made to make certain buttons or texts a little more noticeable, which were then improved in later versions.

• Security

Since the system is dealing with personal and financial data, strong protection was a must. In this area, the development followed best practices by making sure all sensitive information like user login credentials, transaction history, and personal details were encrypted both when stored in the database and when transferred between the server and user's browser. Tools like SSL were implemented to make the web-based platform secure. Additionally, login sessions are monitored and logged to detect any suspicious activity. The encryption methods used match up with modern security standards, so users can trust that their data isn't exposed.

What Feedback Taught Us

Gathering feedback from both our internal development team and client demo sessions turned out to be really helpful in improving the software. During the internal reviews, team members noticed that certain parts of the

layout were not very clear or intuitive, which could confuse new users. Based on this input, we decided to reorganize some sections of the interface to make it more straightforward and easier to understand. In addition, clients mentioned that the error messages shown by the system were too general and didn't help them understand what went wrong. Because of that, we updated the error messages to be more detailed and user-friendly, so users can know exactly what the issue is and how to fix it. However, not all suggestions could be acted on immediately. Some users requested extra features such as more types of investment options, which we agreed would improve the system. But due to limited time and project scope, those features had to be left out for now. We made a note of them and plan to include those enhancements in a future version of the application. This feedback process helped us see what really matters to users and what needs to be improved going forward.

What Can Be Better in Future Versions?

- **More Investment Types**: To appeal to different types of users, we should include additional investment options.
- **Better Reports**: Users might benefit from visual reports (like graphs and charts) to see how their investments are doing over time.
- **Mobile Version**: A mobile app or responsive web version would help users access the system on the go, which is really important these days.

Discussion of Two Approaches to Improving Software Quality within a Software Lifecycle

Quality Improvements

Software Quality

A software quality product is characterized by its suitability for its intended function. That is, a high-quality product performs exactly what its consumers want it to do. The fitness of use of software products is typically stated in terms of meeting the standards outlined in the SRS document. (geeksforgeeks, 2024)

Quality attributes

- Portability

A program is said to be transportable if it can be easily designed to function in various package contexts, on various machines, with various code products, and so on.

- Usability

A program has intelligent usability if multiple groups of users (for example, knowledgeable and inexperienced users) can easily trigger the product's features.

- Reusability

A program has intelligent reusability if different parts of the product may simply be reused to construct new products.

Correctness

Software is correct if the SRS document's entirely distinct demands are correctly enforced.

- Reliability.

If there are fewer faults in software, it is more dependable. Because software developers do not intentionally design their program to fail, dependability is determined by the amount and type of errors they create. Designers may increase reliability by making the program easy to implement and alter, extensively testing it, and guaranteeing that if faults occur, the system can handle them or recover quickly.

- Efficiency.

The more efficient a piece of software is, the less CPU time, memory, disk space, network bandwidth, and other resources it consumes. Customers care about this because it helps them save money on software maintenance, however with today's powerful computers, CPU time, memory, and disk utilization are less of an issue than in the past.

Two Practical Ways to Improve Software Quality

There are many ways to improve software quality, but two of the most effective ones we used in our project were Continuous Integration with Automated Testing, and Peer Code Reviews. These both helped us to build more stable, bug-free, and maintainable software in different ways.

1. Continuous Integration (CI) with Automated Testing

Continuous Integration means that all developers regularly add their code into one shared project or codebase. Each time new code is added, the system automatically runs tests to check if the new changes work correctly without breaking anything that was already working. In our case, every time a team member updated the Enomy-Finances application, the CI server would trigger a series of automated tests. These included **unit tests** (to check individual functions), integration tests (to see if different parts of the software worked well together), and regression tests (to make sure old features still worked after new updates).

Using CI helped us spot bugs quickly, sometimes within minutes of writing the code. For example, when we updated the currency conversion logic to include more currencies, the automated test immediately showed us that some older currencies were no longer calculating correctly. That saved a lot of time compared to finding out the problem much later during user testing.

Also, CI reduced the pressure on our developers. They didn't have to manually test everything again and again. It made the process smoother, less tiring, and more reliable. And since testing was automatic, we could test much more frequently even daily. This made our software more stable overall and avoided last-minute problems before the final delivery.

2. Peer Code Reviews

Another simple but very helpful method was peer code reviewing. In this process, developers look at each other's code and give feedback. They check for mistakes, messy logic, and whether the code follows the team's standard style. We used this method for important modules, especially the ones related to user accounts, savings plans, and investment features.

One time, a team member had written an algorithm for calculating savings interest, but it was a bit too complex and hard to understand. During the code review, someone suggested a simpler version using built-in Java

methods that made the code shorter and easier to test. These kinds of small improvements added up and helped us write cleaner and faster code overall.

Peer reviews also helped junior developers learn from senior ones. Instead of just coding alone, team members learned better ways to solve problems and how to avoid common mistakes. And sometimes, reviewers spotted issues that even the tests missed like poor logic or unnecessary steps in the code that made it slower.

How These Two Methods Improved Quality

By using both CI and code reviews, we created a system where bugs were caught early, errors were fixed quickly, and the whole project stayed on track. These two methods worked well together. CI handled the fast, automatic side of testing, while peer reviews added the human judgment part that machines can't do.

CI helped make sure every update was properly tested and safe to deploy. Peer reviews made sure the code was smart, efficient, and well-written. Both approaches also improved team communication and reduced stress by making responsibilities clear. We always knew someone else would check our code before it went live, which made everyone more confident.

In short, Continuous Integration with testing and Peer Code Reviews gave us better results by increasing code quality, reducing bugs, and making the software more dependable. Even though these methods take some effort to set up, they're totally worth it especially for long-term projects where stability, speed, and quality really matter.

Evaluation of the Suitability of the Solution in Relation to the Needs of Enomy-Finances

1. Currency Conversion and Transaction Fees

The currency conversion feature in Enomy-Finances is made to help users change between major currencies like GBP, USD, EUR, BRL, JPY, and TRY. It gives instant conversion results based on current rates, so users

always get up-to-date information. What makes it better is how it clearly shows any fees applied during transactions. The fees change depending on how much is being converted, so users can understand costs before confirming. The tool is easy to use with a simple interface – users just pick the currencies and enter the amount. On the technical side, the system is hosted in the cloud, which helps it run faster and stay reliable. Important data like currency history and user details are protected using encryption and caching to keep things secure and efficient. This helps make the process trustworthy and smooth for users.

2. Savings and Investment Quote Generation

The investment part of the system lets users get custom quotes based on what they input – like how much money they're starting with, how much they can save monthly, and the type of investment they choose. Behind the scenes, it uses smart financial formulas to work out things like total return, service fees, and possible profits over different time periods. If someone types in something wrong, the system can catch the error and give a helpful message to fix it. After calculating, the system makes clear reports that help users decide which plan is best for them. These reports are generated automatically, saving time for both the user and staff. All the client information is saved securely and follows strict rules to protect user privacy, like GDPR.

3. User Interface and User Experience (UI/UX)

When designing Enomy-Finances, the team really focused on making the platform simple and easy to use for everyone whether they were staff or clients. That's why the interface has a neat layout and smooth navigation. It works through a website, so users can access it from their computer or phone without needing to install anything. The developers used modern tech like HTML5, CSS3, and JavaScript frameworks such as React or Vue.js to make it interactive and responsive. They also used RESTful APIs so that the front and back ends can communicate fast and without delays. Usability tests were done often to see how real users felt when using the site, and based on their feedback, improvements were made. This resulted in a system that looks good, works well, and is easy to understand, helping people use it with confidence.

4. Security and Data Protection

Security is one of the most important parts of any finance app, and Enomy-Finances takes it seriously. They added multiple layers of protection to make sure all the personal and financial data stays safe. For example, all data going through the internet is encrypted using TLS, and any data stored is protected with AES-256 encryption. The system uses secure login methods like multi-factor authentication (MFA), where users not only need a password but also get an OTP to their phone. This keeps hackers out even if they somehow guess a password. User roles are also carefully managed with role-based access controls so that only the right people can see or change certain information. These security designs follow global standards like GDPR and PCI DSS, which makes sure the app follows the law and builds user trust.

5. Scalability and Performance

As the number of users grows, Enomy-Finances has to handle more traffic and data, so the developers made sure the app could scale without slowing down. They used cloud platforms like AWS or Microsoft Azure, which allow the system to grow when needed by adding more resources automatically during busy times. Also, the system is split into smaller parts (microservices), which means each part (like currency conversion or investment planning) can grow on its own without affecting others. To keep the app running fast, caching was added to store frequent data temporarily, and load balancing helps share traffic evenly so no server gets overloaded. This setup makes sure that even if thousands of users are online at once, the system still works quickly and smoothly without crashing.

6. Security Needs – MFA and Compliance

To prevent accounts from being hacked, Enomy-Finances includes Multi-Factor Authentication (MFA). For example, when someone logs in, they don’t just use a password – they also need to enter a code sent to their phone. This second step makes it much harder for attackers to get in. Besides that, the app follows all the important data laws like GDPR for data privacy and PCI DSS for handling payment data. Every action is tracked using audit logs, so if something goes wrong, it’s easy to find out what happened and fix it. This kind of compliance builds trust and helps avoid legal trouble.

7. Aligning Software with Business Goals

The software isn't just made to work – it's made to support the company's bigger goals. For example, its scalable design means that as the business adds more users or features, like new types of financial dashboards, the system can handle it without breaking. Also, there are tools in place that generate reports automatically, saving time and helping managers make fast decisions. These things help the business run smoother, reach more users, and grow without needing to rebuild everything from scratch.

Analysis of the Suitability of Chosen Design and Development Tools and Techniques

For the Enomy-Finances project, several software tools and techniques were chosen to support the development of a reliable, scalable, and user-friendly financial system. These tools cover algorithmic, logical, data, and importantly behavioural design and testing, which help model, simulate, and verify how the system responds during use.

1. Chosen Tools and Techniques

- **Unified Modeling Language (UML):** UML was extensively used not only for designing the static architecture but also for modelling system behaviour. Behavioural diagrams such as Use Case, Sequence, and Activity Diagrams illustrated how users interact with different modules and how the system processes those interactions step-by-step.

- **Integrated Development Environments (IDEs):** IDEs like Visual Studio and IntelliJ IDEA improved coding productivity by providing debugging, version control integration, and runtime analysis tools. These helped developers observe how their code behaves when executed.
- **Relational Database Management System (MySQL):** Used to manage structured financial data securely. While not a behavioural tool itself, database transaction handling impacts system behaviour by ensuring data integrity under concurrent operations.
- **Testing Frameworks (JUnit and Selenium):** Automated testing tools were critical for behavioural validation. JUnit was used for unit and integration tests simulating different input scenarios to observe how modules behaved under various conditions, including errors. Selenium tested the behaviour of the user interface in real time, checking for correct responses to user actions.

2. Algorithmic Design

The algorithms driving currency conversions and investment calculations were designed using flowcharts and pseudocode, providing clear logic flows that mapped well to expected system behaviour. For example, currency conversion algorithms included error handling behaviours when live exchange rate APIs were unavailable, ensuring the system failed gracefully.

Alternative approaches like AI-based predictive analytics could offer more advanced behavioural modelling but were not chosen here due to the project's focus on reliability and deterministic outputs in financial calculations.

3. Modelling How the System Behaves

- **Behavioural Diagrams** helped us map out what the user wants to do and what the system should do in return. For example, Use Case diagrams showed user goals, while Sequence diagrams helped us plan the back-and-forth steps in the system.
- We also used **Finite State Machines (FSMs)** for things like investment plans that go from 'Active' to 'Paused' to 'Closed'. This made sure the system didn't skip steps or behave strangely.

4. How We Tested the Behaviour

- **Automated Tests** were run after every code update using Continuous Integration (CI). This helped catch mistakes quickly and made sure the system still worked after changes.
- **Mocking APIs** helped us test what happens when external services (like exchange rate APIs) were slow or down.
- **Peer Reviews** from teammates also helped catch logic problems that automated tools didn't find.

5. Monitoring Performance During Development

We also used some **basic monitoring tools** to spot slowdowns or unusual behaviour while testing. Though we didn't go too deep into profiling or performance tuning, we were able to make the system run faster and smoother.

6. Suitability and Limitations

Strengths

- The combination of UML behavioural diagrams and automated behavioural testing provided a clear understanding of how the system should work and rigorous validation that it did.
- IDE and CI tools enabled rapid feedback on runtime behaviour.
- Peer reviews improved both code quality and behavioural consistency.

Limitations

- The project did not include advanced behavioural simulations like AI-based predictive modelling, which could enhance future analytics features.
- Runtime monitoring was basic; advanced profiling tools could improve performance tuning.

7. Future Considerations

Try using AI to predict user behaviour and give smart suggestions.

Combine BPMN and UML to make workflows even easier to understand.

Add better profiling tools to test performance when lots of users are active.

Conclusion

A wide range of behavioural tools and techniques were applied throughout the Enomy-Finances project, including UML behavioural diagrams for design, automated testing for behavioural validation, mocking for simulating external dependencies, and peer reviews for quality assurance. Together, these tools ensured the system's behaviour met client requirements, handled edge cases properly, and remained reliable and maintainable. While some more advanced behavioural tools could be considered in future iterations, the chosen approach balanced complexity and reliability well for this project's scope.

Critical Review of Undertaking a Systems Investigation Leading to the Design, Development, and Testing of the Software Solution

1. Systems Investigation Process

• Stakeholder Discussions

At the start, we spoke with different people involved in the project mainly the Enomy-Finances team, some regular users, and upper-level management. This wasn't just about listing features, but really listening to what they *needed* from the software. These chats revealed some key must-haves like a secure way to convert currencies, tools for generating investment quotes, and strict privacy for user data. These talks were important because they helped us understand real business problems instead of just guessing.

• Understanding Requirements

After we collected input, we moved into analysing what was really needed for the software to work properly. This included mapping out every important requirement from the user's point of view and from the business's side too. This step made sure we didn't miss things like protecting data or making sure the financial numbers were calculated properly.

Impact on Quality and Success

Doing this kind of deep dive with stakeholders meant we didn't just build a random app it was something that directly fixed the issues the company was facing. Because of that, the system ended up being more reliable and useful. It worked the way they wanted because we actually based it on their real-world needs.

2. From Investigation to Design, Development, and Testing

Once we had clear requirements, we began turning them into actual working parts of the system.

• Design Stage

The designs were carefully created based on the earlier investigation. We used diagrams like flowcharts and FSMs (Finite State Machines) to visually map how the system should behave. ERDs (Entity Relationship Diagrams) and Data Flow Diagrams were also done to organise how the financial and user data moved through the system. These helped everyone from developers to clients understand the logic.

• Development Phase

We didn't build the full software all at once. Instead, we used an iterative (step-by-step) method. This allowed us to create and test one part at a time, like the currency converter or the investment calculator. We used Java and SQL for coding and made sure security was built-in from the very beginning not added later. All sensitive info like passwords and personal data were encrypted.

• Testing Phases

Testing wasn't just a one-time task. We tested at many levels

- **Unit Testing:** Checked if each small part (like a function to convert currency) worked properly on its own.

- **Integration Testing:** Made sure parts worked together (e.g., currency conversion + investment quotes).
- **System Testing:** Tested the full app as a whole in a realistic environment.
- **User Acceptance Testing (UAT):** Let real users try it to confirm if it meets their expectations.

Impact on Effectiveness

Thanks to these multiple tests, we found issues early. For example, in the currency converter, we fixed a problem with rounding off large numbers. Also, because of security testing, we made sure no personal data could leak or be misused. This gave users more confidence and made the system dependable.

3. Risk Management Throughout the Phases

Risk was something we planned for from the beginning. We didn't just wait for problems to pop up we looked ahead and tried to spot what could go wrong.

• Risk: System Complexity

Handling live financial data, making it accurate, and still keeping everything secure was very challenging.

How We Handled It

We broke down development into smaller pieces and tested them regularly. Also, we used known security tools and followed standards like HTTPS and encryption techniques from the start.

• Risk: Data Privacy & Security

Since we were handling users' financial and personal data, any leak could ruin trust in the system.

How We Handled It

We carefully designed how data would be stored and who could access it. Encryption was used everywhere logins, storage, and even during data transfer. We also checked for vulnerabilities regularly.

• Risk: Integration with Old Systems

Enomy-Finances already had some systems running (like spreadsheets and older databases). We had to make sure the new system didn't crash or clash with them.

How We Handled It

We did testing specifically to see how the new and old systems would connect. We added things slowly instead of replacing everything in one go.

Impact on Quality

By handling these risks early and with proper care, we reduced chances of failure after launch. The system felt more polished and safe to use. Security and compatibility were strong points by the end.

The whole systems investigation helped the final software be much better than it would have been otherwise. From the very first stakeholder interview to the final test, each step improved the overall quality. Risks were spotted early, design choices were based on actual needs, and security was never ignored.

The investigation was not just helpful it was the foundation for everything that came later. Without it, the solution wouldn't be this reliable, user-friendly, or secure. Every step we took was based on real data and feedback, and that's what made the software successful for the Enomy-Finances team.

Critical Review of How Data-Driven Software Would Improve the Reliability and Effectiveness of the Solution

Data-driven software uses advanced data analytics, machine learning, and dynamic data management to improve decision-making, adaptability, and overall system dependability. For the Enomy-Finances system, integrating data-driven features could boost both reliability and effectiveness in several important ways.

Improved Decision-Making through Analytics

By analyzing large amounts of financial data, the system can provide predictive insights that help users make smarter investment choices. For example, forecasting market trends based on past behavior can guide clients to optimize savings plans. This automation reduces human error, improving reliability since decisions

are based on consistent data patterns. At the same time, personalized advice enhances user satisfaction, making the system more effective and trusted.

Dynamic Adaptability to User Needs

Data-driven approaches allow the system to learn from how users interact with it and adapt accordingly. Machine learning could tailor financial products, like suggesting investment plans that match a user's risk tolerance. This continuous learning keeps the system relevant and accurate as user preferences evolve, improving reliability by avoiding outdated or irrelevant advice. Offering customized options also increases user engagement and adoption, boosting overall effectiveness.

Enhanced Risk Management

One of the strongest benefits is detecting risks and anomalies early. Real-time fraud detection that monitors transaction patterns can quickly flag suspicious activity. This proactive approach minimizes disruptions and protects users' assets, increasing system reliability. Users feel more confident with strong security measures in place, which improves the system's reputation and effectiveness.

Optimized System Performance

Data-driven monitoring tools can analyze server performance and automatically adjust resources to handle peak usage. This reduces downtime and prevents slow response times, making the system more dependable. A smoother user experience leads to greater customer satisfaction and operational efficiency, enhancing effectiveness.

Challenges and Considerations

Despite these advantages, there are challenges to implementing data-driven software. Handling large volumes of sensitive data demands strict compliance with regulations like GDPR to protect user privacy. Moreover, developing and maintaining AI features require significant computational resources and expert staff, which can increase costs. The added complexity might also lengthen development cycles and raise maintenance efforts. To manage these risks, careful planning, strong security measures, and phased implementation are needed to balance innovation with practical constraints.

Review of the Performance of the Enomy-Finances Application against Identified Requirements

Review of Enomy-Finances Application Performance Against Requirements

The Enomy-Finances app was developed to meet several important functional and non-functional requirements. This review looks at how well the application performed against these goals, pointing out where it succeeded and where improvements could be made.

Functional Requirements

Currency Conversion Module

The system needed to provide accurate, real-time currency conversions and have backup options in case of internet issues. It successfully connected with APIs to get live exchange rates, making sure the data stayed current. Also, it used cached rates as a fallback when the API was unavailable, so users could still convert currencies offline. Overall, this module worked well, making the system reliable and easy to use.

Savings and Investments Module

Users should be able to manage savings plans and track how their investments are doing. The app supported different savings options that users could customize. It included basic reports showing investment returns, which helped users keep track. While the essential features were there, adding more advanced analytics and visualization tools would make the module even better and easier to use.

Secure Data Storage

Protecting sensitive financial information was critical. The app applied strong encryption both when data is stored and when it's being transmitted. Multi-factor authentication was also added to control who can access the system. These security measures not only met but went beyond basic compliance requirements, ensuring user data stays safe.

User Interface and Experience (UI/UX)

The interface needed to be simple and user-friendly. The app delivered a clean, responsive design that worked smoothly on different devices. User feedback showed the interface was easy to navigate and clear. However, adding more accessibility features, like voice assistance or customizable themes, could make the system more inclusive for all users.

Non-Functional Requirements

Performance

The system had to respond quickly and handle many users at once. Stress testing showed it handled expected loads well without slowing down. Database queries were optimized to reduce delays, which kept the system responsive. This means performance targets were successfully met.

Scalability

The system needed to grow with future demands in users and features. It was built with a modular design, so new functions can be added easily without disrupting existing ones. The database design supports this growth by organizing data efficiently. Though it's set up well for scalability, adding real-time monitoring tools could help prepare better for future increases in usage.

Compliance

Following financial regulations, like GDPR, was essential. The app made sure to protect data with clear audit trails and by getting proper user consent. It fully complies with these standards, which builds trust and reliability with users.

Reliability and Availability

High availability and fault tolerance were important goals. Automated backups and error recovery methods were implemented, helping to avoid data loss. During testing, the system achieved 99.9% uptime, which is excellent for financial applications requiring constant availability.

Gaps and Suggestions for Improvement

- **Advanced Analytics:** The investment module would benefit from more powerful forecasting tools to help users plan better.
- **Accessibility:** Adding features like screen readers, adjustable colors, and other assistive options could make the app usable by a wider audience.
- **Proactive Maintenance:** Introducing real-time monitoring and AI-based anomaly detection could improve reliability by spotting issues before they cause problems.

Assessment of How Feedback Was Used to Improve the Enomy-Finances Solution

Feedback was very important in shaping and improving the Enomy-Finances application. It helped make sure the system met both user needs and organizational goals. Below is a summary of how feedback was put into action, reasons for the decisions made, and which suggestions were not followed up on along with explanations.

Important Feedback That Was Implemented

Improved User Interface (UI)

Users said the navigation was confusing and wanted clearer labels with easier workflows. To fix this, the navigation bar was redesigned to group similar features together. Button labels were made clearer and tooltips added to explain functions. The dashboard was simplified to highlight the most important information. These changes made the system easier to use, helping users learn faster and feel more comfortable.

Better Data Security

Some users worried if the security measures were strong enough, especially since sensitive financial info is involved. In response, multi-factor authentication (MFA) was added for all accounts. Encryption was upgraded to AES-256 for stored data and TLS was used for data being sent over the network. These steps made the app more secure and helped it meet financial rules, which boosted user confidence.

Savings and Investment Tracking

Users wanted more detailed reports on how their savings and investments were doing. The team added visual dashboards that show trends and history for these accounts. Also, users can now download reports in PDF format. These features support users in better planning their finances by giving them more useful information.

Performance Improvements

There were complaints about slowdowns during busy times. To fix this, database queries were optimized

and indexes created on commonly used tables. Load balancing was introduced to spread the user traffic evenly. These changes kept the app responsive and reliable even when lots of people were using it at once.

Feedback That Was Not Implemented

Live Chat Customer Support

Some people wanted a real-time chat feature for customer help. However, due to limited time and resources, this was not possible for the first version. The team decided that a detailed FAQ section and email support would be enough at launch. Adding live chat is planned for future updates once the app has more users.

Cryptocurrency Support

A few users asked for the ability to invest in cryptocurrencies. This was considered out of scope because of regulatory challenges and low demand. However, if cryptocurrencies become more popular in financial services, this feature might be added later.

3 Ongoing Feedback Process

Throughout the development, the team maintained a feedback loop involving users, stakeholders, and developers. This helped catch issues early and improve the app step-by-step. User Acceptance Testing (UAT) was especially helpful for gathering real user opinions and making quick fixes.

Recommendations for future improvements to the solution, including a supported rationale for these suggested improvements.

Advanced Analytics and Financial Insights

Recommendation: Use machine learning techniques to add predictive analytics and give personalized financial advice. For example, the system could predict how savings might grow over time, suggest investment options based on user habits, and spot spending patterns.

Rationale: Adding smart analytics would help users make better financial choices, which can make them more satisfied and keep them coming back to use the app.

Accessibility Enhancements

Recommendation: Improve features that make the app easier to use for everyone, including: screen readers for those with visual impairments, high-contrast themes and adjustable colors for better visibility, and voice command options for hands-free use.

Rationale: Making the app more accessible follows standards like WCAG and allows people with disabilities or different needs to use it comfortably, broadening the user base.

Real-Time Customer Support

Recommendation: Add a live chat or chatbot option so users can get help immediately. This might include AI chatbots to answer common questions and live support agents for more complex problems.

Rationale: Real-time assistance improves user experience by quickly solving problems, which builds trust and keeps users happy.

Cryptocurrency Support

Recommendation: Include features for managing cryptocurrencies, like wallets, converting between crypto and regular money, and showing market trends.

Rationale: Since cryptocurrencies are becoming more popular, supporting them can attract tech-savvy users and grow the app's market reach.

Gamification Features

Recommendation: Add game-like elements such as badges, leaderboards, and rewards for hitting financial goals.

Rationale: Gamification motivates users and makes managing money more fun, which helps keep them engaged.

Integration with Third-Party Services

Recommendation: Allow the app to connect smoothly with other financial tools like QuickBooks, payment services such as PayPal or Google Pay, and planning apps like Mint.

Rationale: Connecting with other platforms makes managing finances easier for users and makes the app more flexible for different needs.

Proactive Monitoring and Maintenance

Recommendation: Use AI-based monitoring tools to detect performance issues and security risks before they become serious problems.

Rationale: Being proactive helps keep the app reliable and available, which maintains user trust and satisfaction.

Scalability for Future Growth

Recommendation: Move to a microservices system architecture so the app can handle more users and add features without slowing down.

Rationale: A scalable setup lets the system grow with the business while keeping it stable and fast.

Comprehensive Multi-Language Support

Recommendation: Add support for more languages, including popular global and regional ones.

Rationale: Offering multiple languages makes the app accessible to people who don't speak English, helping expand its reach to new markets.

Data Visualization Enhancements

Recommendation: Improve how financial data is shown by using better visual tools like interactive charts, heatmaps, and dashboards.

Rationale: Better visuals make complex information easier to understand, which helps users make smarter decisions and stay engaged.

Conclusion

By working on these improvements, Enomy-Finances will stay competitive and meet changing user needs better. These ideas focus on making the app easier to use, adding new functions, and making sure it can grow smoothly and reliably over time. Prioritizing these in future updates will help create a stronger, more user-focused financial tool.

Effectiveness of the Chosen Investigation Tool

The questionnaire designed to gather requirements for the development of the Enomy-Finances System is an effective tool for various reasons. First, it is structured with closed-ended questions, such as yes/no or multiple-choice options, which simplify responses for stakeholders, including financial managers, analysts, and administrators. This approach not only saves time but ensures that responses are clear, consistent, and measurable, aiding in the identification of common issues and priorities related to financial operations.

The questions focus on critical aspects, such as current system limitations, expectations for features like real-time currency conversion, investment tracking, and automated reporting, and the need for enhanced security and compliance. This comprehensive methodology ensures that all essential areas of requirement gathering are addressed without omissions.

The questionnaire also fosters stakeholder participation by providing financial professionals and system users a voice in shaping the system to meet their operational needs. It enables valuable feedback that the development team can incorporate into the system design, ensuring alignment with user expectations and industry requirements.

Ultimately, the questionnaire's simplicity, thorough coverage of key topics, and data-driven approach make it an essential tool for gathering requirements. It ensures that the Enomy-Finances System is not only technically

robust but also tailored to the financial management needs and objectives of its users, enhancing overall organizational efficiency and effectiveness.

References

References

Anon., 2024. *AGILE SOFTWARE.* [Online]
Available at: https://www.geeksforgeeks.org/agile-sdlc-software-development-life-cycle/#what-is-the-agile-software-development-life-cycle-agile-sdlc
[Accessed 22 12 2024].

Anon., 2024. *tutorialspoint.* [Online]
Available at: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm
[Accessed 22 12 2024].

Anon., n.d. *gwentechembedded.* [Online]
Available at: http://gwentechembedded.com/five-stages-of-a-software-development-life-cycle/
[Accessed 22 12 2024].

Anon., n.d. *javatpoint.* [Online]
Available at: https://www.javatpoint.com/software-engineering-software-development-life-cycle
[Accessed 22 12 2024].

CFI Team, n.d. *corporatefinanceinstitute.* [Online]
Available at: https://corporatefinanceinstitute.com/resources/management/feasibility-study/
[Accessed 22 12 2024].

cs.stackexchange.com, 2024. *what-is-the-difference-between-finite-automata-and-finite-state-machines.* [Online]
Available at: https://cs.stackexchange.com/questions/10357/what-is-the-difference-between-finite-automata-and-finite-state-machines
[Accessed 22 12 2024].

geeksforgeeks, 2024. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/software-engineering-software-quality/
[Accessed 23 12 2024].

Sami, M., 2012. Software Development Life Cycle Models and Methodologies. 15 3.

Satyabrata_Jena, 2024. *geeksforgeeks.* [Online] Available at: https://www.geeksforgeeks.org/types-of-feasibility-study-in-software-project-development/ [Accessed 22 12 2024].

saylor, n.d. *saylor.* [Online] Available at: https://learn.saylor.org/mod/book/view.php?id=33038#:~:text=There%20are%20seven%20techniques%20we,documents%2C%20and%20review%20of%20software. [Accessed 22 12 2024].

senwork.com, 2024. *feasibility-conceptual-study/.* [Online] Available at: https://senwork.com/engineering/feasibility-conceptual-study/ [Accessed 22 12 2024].

simplilearn, 2024. *simplilearn.* [Online] Available at: https://www.simplilearn.com/tutorials/programming-tutorial/what-is-software [Accessed 22 12 2024].

smartdraw, 2024. *smartdraw.* [Online] Available at: https://www.smartdraw.com/activity-diagram/ [Accessed 22 12 2024].

smartdraw, 2024. *smartdraw.* [Online] Available at: https://www.smartdraw.com/data-flow-diagram/ [Accessed 23 12 2024].

visual-paradigm, n.d. *visual-paradigm.* [Online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ [Accessed 22 12 2024].

visuresolutions, 2024. *visuresolutions.* [Online] Available at: https://visuresolutions.com/blog/requirements-definition/ [Accessed 20 12 2024].

www.invensislearning.com, 2024. *risk-management-process-steps/.* [Online] Available at: https://www.invensislearning.com/blog/risk-management-process-steps/ [Accessed 22 12 2024].

www.javatpoint.com, 2024. *software-engineering-sdlc-models.* [Online] Available at: https://www.javatpoint.com/software-engineering-sdlc-models [Accessed 22 12 2024].