# PORTFOLIO WEBSITE FOR NEONTECH

Shrikanta Paul
Full Stack Web Developer

**VERSION 4.0**

# CONTENTS

# 1. PROJECT OVERVIEW

NeonTech is a company, who are haring Tech Expert peoples like

- Web Developer
- Photographer
- Web Designer
- Videographer

So, to expend their business furthermore, they wanted to build a website-based platform.

# 2. REQUIREMENTS

## 2.1 FUNCTIONAL REQUIREMENTS

Like other corporate companies' portfolio NeonTech website has two parts, Website and Web Application System-

- **Website**
  - Home
  - Events
  - Video & Photo Gallery
    - Only Admin Panel
  - Blogs
    - Only Admin Panel
  - Achievements
    - Only Admin Panel
  - Projects
    - Only Admin Panel
  - About Us
  - Login or, Registration (Go for The Web Application)

- **Web Application System**
  - User
    - Login or, Registration
      - User Id and Password are provided by admin
    - Participate in Events
  - Admin
    - Create User
    - Create Events
    - Upload blogs
    - Upload Photos & Videos
    - Upload Achievements
    - Upload Projects

## 2.2 NON-FUNCTIONAL REQUIREMENTS

- Performance: Fast page loading, optimize media content for low data consumption.
- Security: Standard.
- Compatibility: Run in popular website and various devices.
- Accessibility: Make accessible for all kinds of users.
- SEO: Ensure search engines visibility and ranking.

## 3. DESIGN AND USER INTERFACE

- Responsive design so that it can adapts to different screen sizes.
- User-friendly navigation.

## 4. TECHNOLOGY REQUIRE

Specify the technologies, languages, and frameworks to be used –

- Framework: Modern
- Coding Architecture:
    - HTML
    - CSS
    - JavaScript
    - PHP
- Database: MySQL

## 5. MILESTONES AND REPORTING

| Milestone | Tasks | Reporting | Time |
|---|---|---|---|
| Analysis | | Submit The Design | 7 days |
| Requirement's collection | Submit to us all data | | 10 days |
| Development | | Review the work | 45 days |
| Testing | | | 10 days |
| Deployment | Must ready the server | Review final work | 5 days |
| Delivery | | Live on server | 5 days |

# 6. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software development life cycle (SDLC) is a methodical approach that ensures the accuracy and quality of the software produced. Software that satisfies customer requirements is produced using the SDLC process. The system development should be finished in the budgeted amount of time and money. A comprehensive plan that describes how to organize, create, and maintain particular software is what the SDLC entails. The SDLC life cycle contains different phases, each of which has a process and outputs that feed into the next step. Application Development Life Cycle is another name for SDLC, which stands for Software Development Life Cycle.

## 6.1 DESCRIPTION OF ALL MODELS

### 6.1.1 WATERFALL MODEL

Software engineering and product development frequently use the waterfall model, which is a linear, sequential approach to the software development lifecycle (SDLC).

The waterfall model applies logical steps in the SDLC to a project in a manner akin to how water pours over a cliff. Each phase of development is assigned specific endpoints or objectives. After they've been completed, their endpoints or objectives cannot be changed.

**When to Use the waterfall Model?**

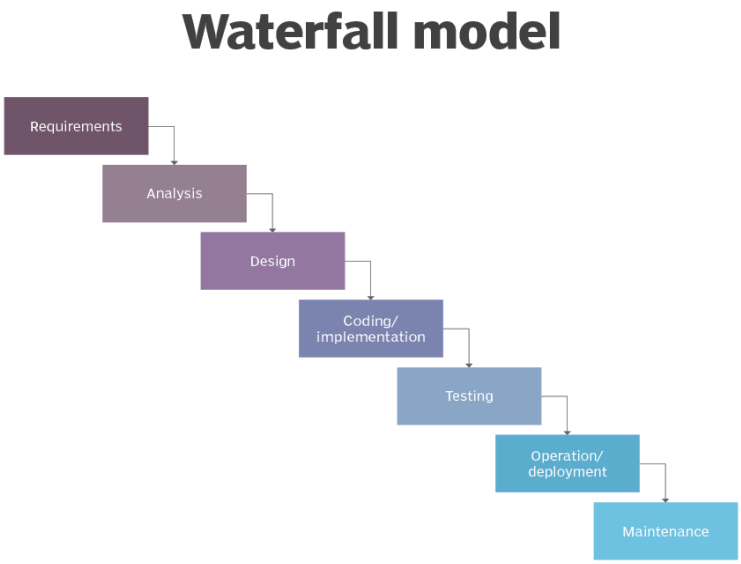
- The waterfall paradigm produces projects that are clearly defined, predictable, and with detailed documentation. They also exhibit the following traits:
- Fixed requirements, ample resources, a schedule, well-understood technology, and a low likelihood of major modifications are all prerequisites.

**Phases of the waterfall model**

The waterfall technique consists of seven steps when applied to a software development process:

- Requirements: A formal requirements document, also known as a functional specification, is created by analyzing potential requirements, timeframes, and guidelines for the project. Without stating specific procedures, this stage of project development defines and plans the project.
- Analysis: To create product models and business logic to direct manufacturing, the system specifications are examined. At this point, the viability of the available financial and technological resources is also examined.
- Design: Technical design requirements, such as the programming language, hardware, data sources, architecture, and services, are outlined in a design specification document.
- Coding and implementation: The original code is created based on the models, logic, and requirement specifications specified in the earlier stages. Before being assembled, the system is typically coded in smaller parts or units.
- Testing identifies problems that need to be fixed through quality assurance, unit, system, and beta tests. This can force a debugging stage code repetition. The waterfall process moves forward if the system passes testing and integration.
- Operation and deployment: When a product or application is considered completely functioning, it is deployed to a live environment.

- Maintenance: To improve, update, and enhance the product and its functionality, corrective, adaptive, and perfective maintenance is continuously carried out. This can entail the introduction of patch updates and new versions.




| **Advantages** | **Disadvantages** |
|---|---|
| 1. Enables big or shifting teams to advance toward a stated common goal at the requirements stage;<br>2. Forces a disciplined, structured organization;<br>3. Simplifies task organization, understanding, and compliance;<br>4. Makes departmentalization and administrative control based on a timetable or deadlines easier;<br>5. Teaches the proper coding practices of defining before implementing design and then coding;<br>6. Simplifies the process of making early system design and specification changes; and<br>7. Milestones and due dates are precisely stated. | 1. Design doesn't seem flexible; when a problem is discovered, the procedure frequently needs to be restarted.<br>2. The method doesn't take user or customer feedback into account midprocess and modifies based on outcomes.<br>3. Testing is postponed until the end of the development lifecycle by the waterfall paradigm.<br>4. It doesn't take into account error correction.<br>5. The technique struggles with requests for changes, scope modifications, and updates.<br>6. Waterfall hinders concurrent work on multiple phases by not allowing processes to overlap, which lowers overall productivity.<br>7. A workable product won't be accessible until later in the project's lifecycle.<br>8. Complex, high-risk ongoing projects are not the best candidates for waterfall. |

## 6.1.2 ITERATIVE MODEL

The iterative approach is a method to the software development life cycle (SDLC) in which initial work on the development is done based on clearly specified fundamental requirements, and then subsequent improvements are added to this base piece of software through iterations until the final system is constructed. The iterative model starts with a straightforward execution of a small set of software requirements, which iteratively improves the evolving variants until the full system is executed and prepared for redistribution. As a result, we receive a functioning piece of software relatively early in the lifespan. Each iteration, often referred to as a release, of an iterative model is produced over a certain and specified time span. As this paradigm is adaptable, bugs and faults from one iteration do not carry over to the next.

**When to Use the Iterative Model?**

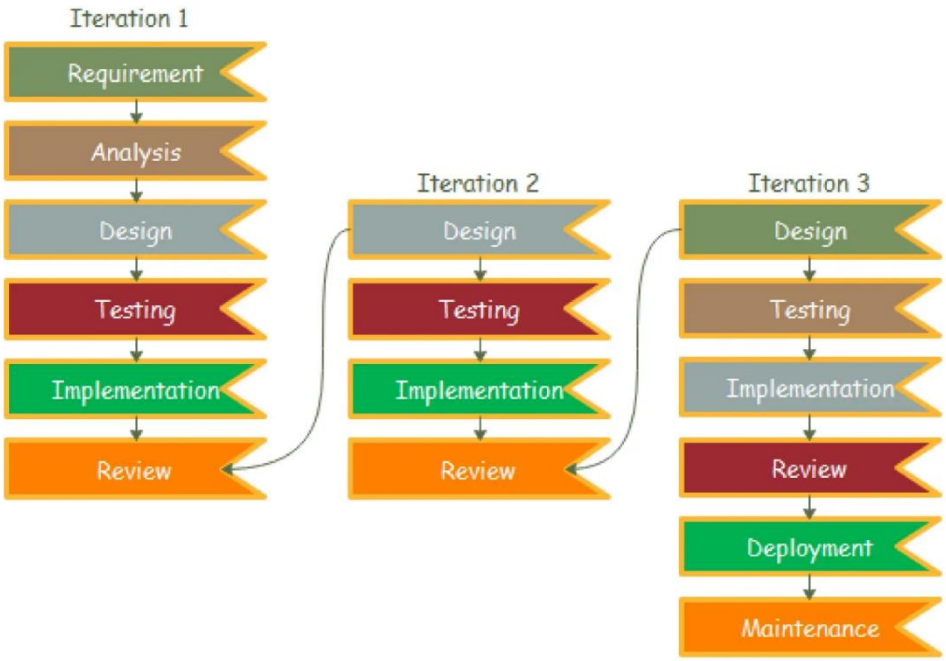
The following use cases are appropriate for the iterative model:

1. A large project can be divided into smaller components and developed by following the iterative approach.
2. At the project's outset, when the needs may be properly understood and articulated.
3. When it is necessary to take into account client feedback at every level - The main requirements are established at first, but as the development process moves forward, some functionalities are changed, and additions are suggested.
4. The development team is experimenting and learning new technology while working on the project.

**Phases of Iterative Model**

- Requirement Gathering & Analysis: In this stage of the iterative process, the business requirements are gathered. An analyst then assesses if they can be satisfied given the financial restrictions. The business requirements are described in detail at this phase, and hardware and software information about the systems is collected and evaluated for viability.
- Design: The project team obtains the entire set of requirements for beginning work in a particular direction during this phase of the iterative model. Then, in order to get clear understanding of the program design and to advance with development, they employ a variety of diagrams, including data flow diagrams, class diagrams, activity diagrams, state transition diagrams, and so on. Developers offer workable solutions in response to their inquiry. The size and importance of the project are other important considerations for determining the project's design complexity.
- Implementation: In accordance with the iterative paradigm, this is the time when the system's real coding is started. The analysis and design from the Design Stage will have an impact on this stage. All requirements, planning, and design strategies have been implemented. The developer will use established coding and metrics standards to implement the selected design. At each level of code development, they must implement a unit test, and they should work to create a fully operational, testable system for that iteration. Depending on the project, this iteration's complexity and duration will change.
- Testing: This phase involves assessing the current build iteration's compliance with a set of standards and guidelines. Performance testing, stress testing, security testing, requirements testing, usability testing, multi-site testing, disaster recovery testing, and other types of testing fall under this category. Testing is a top priority because any errors would have an impact on the software's specification, which would have an impact on the company's bottom line. The tester can develop new test cases or reuse those from earlier releases. To conduct some tests and acquire their feedback, we can also get in touch with the project's stakeholders. When a bug is fixed, a developer or tester must ensure that no new bugs enter the system as a result.
- Deployment: The program is introduced to its working environment following the completion of each phase.

- Review: Following the deployment of the product, we examine its validity and behavior in this step. In the event that mistakes are discovered, the process restarts with requirement gathering.
- Maintenance: Following the deployment of software in a working environment, there may be a need for new updates or problem fixes.




| **Advantages** | **Disadvantages** |
|---|---|
| 1. Rapid Issue Identification: Identifies flaws in functionality or design quickly, enabling prompt corrections to be made within budgetary limits.<br>2. Early functional Product: In contrast to the waterfall paradigm, a functional product is accessible early on.<br>3. Bugs Prevention: By extensively evaluating each iteration's output, problems are found and kept from spreading to subsequent iterations.<br>4. Flexible Requirements: Allows for modifications in requirements to be made at a cheap cost, while some concessions may not be possible due to structural limitations.<br>5. Customer Feedback: Quick implementation is made possible by incorporating customer feedback into every iteration.<br>6. Efficiency: Time is saved by prioritizing design and development over detailed documentation. | 1. Problems with the system architecture may result from incomplete requirement collection in the beginning. Inadequate initial requirements may lead to repeated design modifications.<br>2. Limitations for Frequent Changes: Because it could sabotage the iterative process, this approach is not recommended for projects with regularly changing needs.<br>3. For Small Projects, Not Ideal: It is difficult to divide small projects into smaller iterations since it is impractical.<br>4. Resource-Intensive: Requires more highly skilled resources than the waterfall paradigm does for analysis.<br>5. Complex management might be difficult when managing the full iterative process. |

## 6.1.3 SPIRAL MODEL

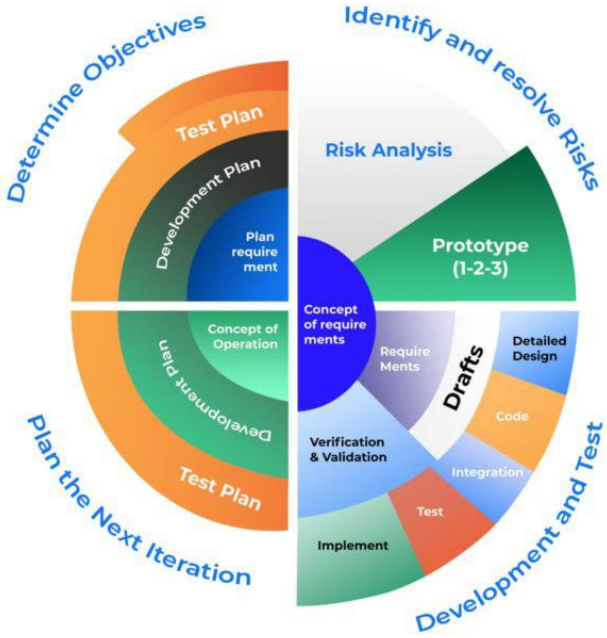
A risk-driven model of the software development process is the spiral model. It combines an iterative model with a waterfall paradigm. The Spiral Model aids in the adoption of software development components from several process models for the software project based on distinctive risk patterns, enabling an effective development process.

**When to use Spiral Model?**

1. When a project is vast in software engineering, a spiral model is utilized.
2. Spiral approach is utilized when frequent releases are necessary.
3. When it is appropriate to create a prototype
4. When evaluating risks and costs is crucial
5. Spiral approach is beneficial for projects with moderate to high risk.
6. The SDLC's spiral model is helpful when requirements are complicated and unclear.
7. If changes are possible at any time
8. When committing to a long-term project is impractical owing to shifting economic priorities

**Phases of Spiral Model**

- Planning: Planning entails estimating the iteration's budget, timetable, and resources. Understanding the system requirements is also necessary for ongoing communication between the system analyst and the client.
- Risk analysis: Risk analysis involves identifying potential risks when a risk mitigation strategy is developed.
- Engineering: Engineering entails software testing, coding, and deployment to the client location.
- Evaluation: The customer's assessment of the software. includes detecting and keeping an eye on hazards including cost overruns and timetable slippage.



| Advantages | Disadvantages |
|---|---|
| 1. Future updates or features can be implemented.<br>2. Because prototypes are constructed in small pieces, cost estimation is simple.<br>3. Recurring or continuous development aids in risk management<br>4. In spiral development, features are introduced quickly and methodically.<br>5. Customer comments are always welcome. | 1. A possibility that the budget or timeline won't be met<br>2. Large projects only require the use of spiral development, which also calls for proficiency in risk assessment.<br>3. Spiral model protocol must be closely followed for it to function properly.<br>4. Having intermediary phases increases documentation.<br>5. It is not suggested for smaller projects to use spiral software development because it could be quite expensive. |

## 6.1.4 V-MODEL

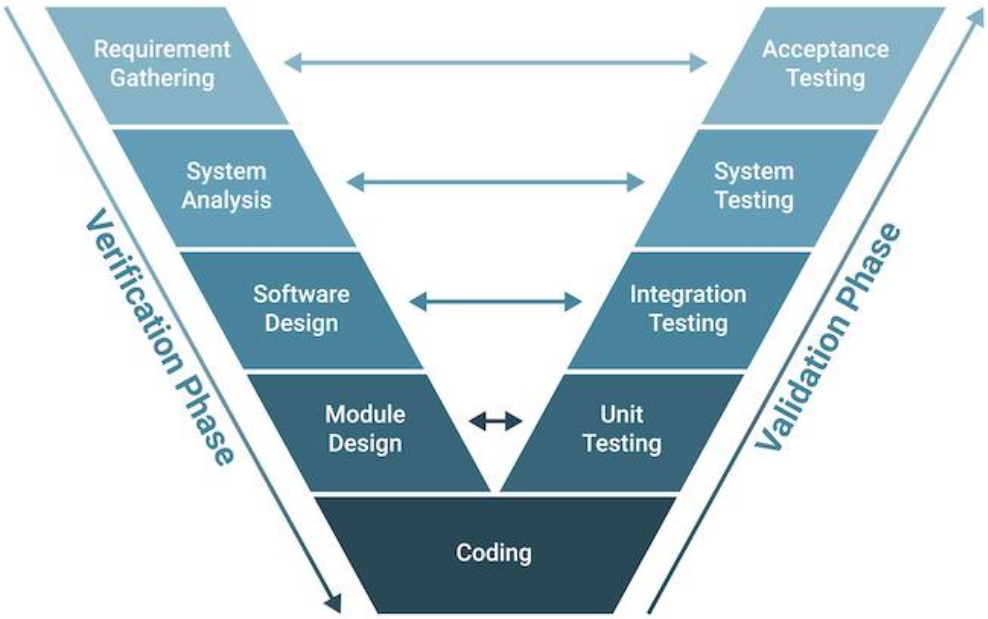
A testing phase is linked to each relevant development step in the V-Model, which is an extension of the waterfall model. This implies that there is a testing phase that is directly related to each and every phase of the development cycle. This is a very structured approach, and the start of the subsequent phase only occurs after the conclusion of the preceding phase.

**When to use V-Model?**

1. The requirements are clearly stated, fixed, and well-documented.
2. Definition of the product is steady.
3. The project team has a solid understanding of technology and knows it is not dynamic.
4. There aren't any unclear or vague requirements.
5. The undertaking is brief.

**Phases of V-Model**

- Business Requirement Analysis: Understand consumer expectations and specific requirements by doing a business requirements analysis. Based on these specifications, create acceptance tests.
- System Design: Design the entire system, including the hardware and communication infrastructure. Create a plan for the system test.
- Architectural Design: Describe the architectural requirements, the breakdown of the modules, and the data transport techniques. For integration tests, get ready.
- Module Design: Create intricate internal designs for system modules. Create unit tests to find bugs early.
- Coding Phase: Write system module code in accordance with coding conventions and standards. Perform code reviews and performance optimization.
- **Validation Phases:**
  - ✓ Execute unit tests created during the module design process to detect errors at the code level.
  - ✓ Test module coexistence and communication with integration testing.
  - ✓ Confirm the overall system's functionality and communication with the outside world.
  - ✓ Acceptance testing involves putting the product to the test in a real-world setting to find compatibility and non-functional problems.




| Advantages | Disadvantages |
|---|---|
| 1. Phases are finished one at a time under this model, which requires extreme discipline.<br>2. works effectively for smaller projects with clearly defined criteria.<br>3. Simple, clear, and convenient to use.<br>4. Because of the model's rigidity, it is simple to manage. Specific deliverables and a review process are included at each phase. | 1. Risk and unpredictability are high.<br>2. A poor choice for intricate and object-oriented tasks.<br>3. Poor model for continuing, protracted projects.<br>4. Not suitable for projects with a moderate to high risk of requirement changes.<br>5. Once an application is in the testing phase, it is challenging to go back and update a functionality.<br>6. Up until the very end of the life cycle, no functional software is developed. |

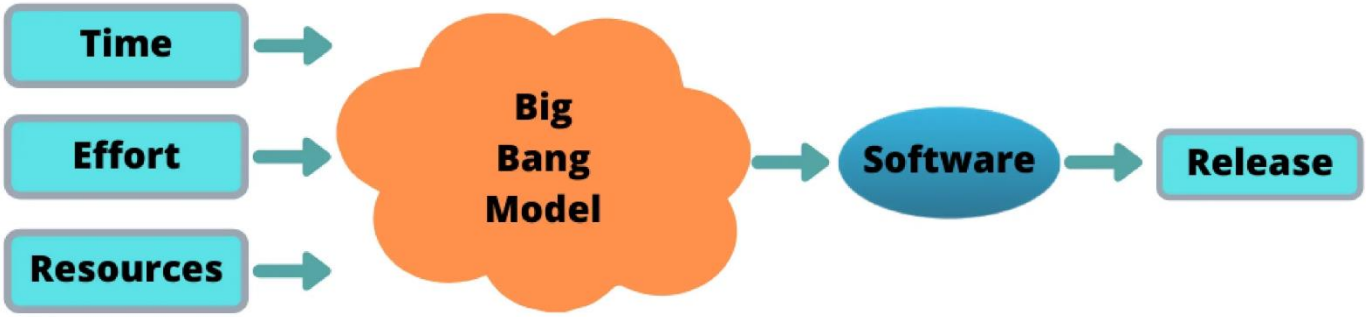
## 6.1.5 BIG BANG MODEL

Developers of the Big Bang Model don't adhere to any set procedure. With the requisite resources—in the form of inputs—development can begin. Additionally, since even the customer's criteria are not specified in this paradigm, the outcome may or may not meet their expectations.

For little projects like academic or practical ones, this model is perfect. On this paradigm, one or two developers can collaborate.

**When to use Big Bang Model?**

As we previously discussed, this paradigm is necessary when the project is modest, such as an academic or practical project. This approach is also utilized when there is a small developer team, when the requirements are not clear, and when the customer has not authorized or specified the release date.




| Advantages | Disadvantages |
|---|---|
| 1. Planning is not necessary.<br>2. Plain Model.<br>3. Requires very few resources.<br>4. simple to handle.<br>5. Developers need flexibility. | 1. High danger and uncertainty exist.<br>2. Not good enough for a big project.<br>3. If criteria are unclear, it could be highly costly. |

## 6.1.6 AGILE MODEL

Agile is a term that means quick or adaptable. The phrase "Agile process model" describes a method of developing software that is iterative in nature. Agile project management techniques divide work into smaller iterations or pieces without directly including long-term planning. The project's requirements and scope are established at the start of the development phase. Plans for the quantity, length, and scope of each iteration are spelled out in detail in advance.

**When to use the Agile Model?**

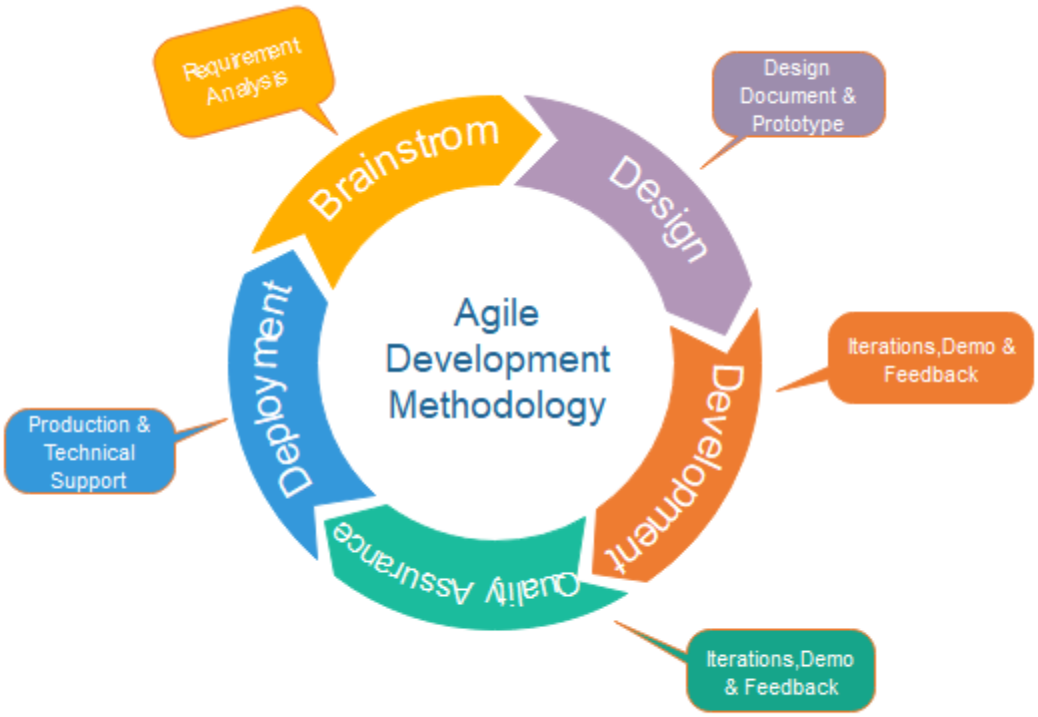
1. Any time there needs to be a lot of adjustments.
2. When a group with extensive knowledge and expertise is available.
3. To meet with a software team whenever a customer is prepared.
4. When there are few projects.

**Phases of Agile Model**

Following are the phases in the Agile model are as follows:

- Gathering needs: You must provide the criteria throughout this stage. Explaining commercial potential and scheduling the time and resources required to complete the project are important. Based on these data, you can assess the technical and financial viability.
- Create the specifications: Work with stakeholders to define requirements after identifying the project. To demonstrate the functionality of new features and how they will integrate with your current system, you can use a user flow diagram or a high-level UML diagram.
- Construction and iteration: Work starts when the team decides what is needed. Starting their project with the goal of releasing a functional product, designers and developers get to work. The product has basic, minimum functionality and will go through numerous stages of refinement.

- Testing: During this stage, the Quality Assurance team evaluates the functionality of the product and searches for bugs.
- Deployment: The team releases a product for the user's working environment during the deployment phase.
- Feedback: Feedback is the final step after a product has been released. Through this, the team collects input on the product and processes it.




| **Advantages** | **Disadvantages** |
|---|---|
| 1. Consistent delivery<br>2. Face-to-face interactions with customers.<br>3. Efficient design that satisfies corporate needs.<br>4. Changes are welcome at any moment.<br>5. It cuts down on overall development time. | 1. The absence of written documents causes uncertainty and leaves room for critical decisions made during multiple phases to be misunderstood at any time by various team members.<br>2. When a project is concluded and the developers are assigned to another project, maintaining the completed project might be challenging due to a lack of good documentation. |

## 6.1.7 RAD MODEL

A short development cycle employing an element-based construction method is emphasized by the RAD model of linear sequential software development. A development team can build a completely functional system in a short amount of time using the RAD approach if the requirements are clearly understood and stated and the project scope is constrained.

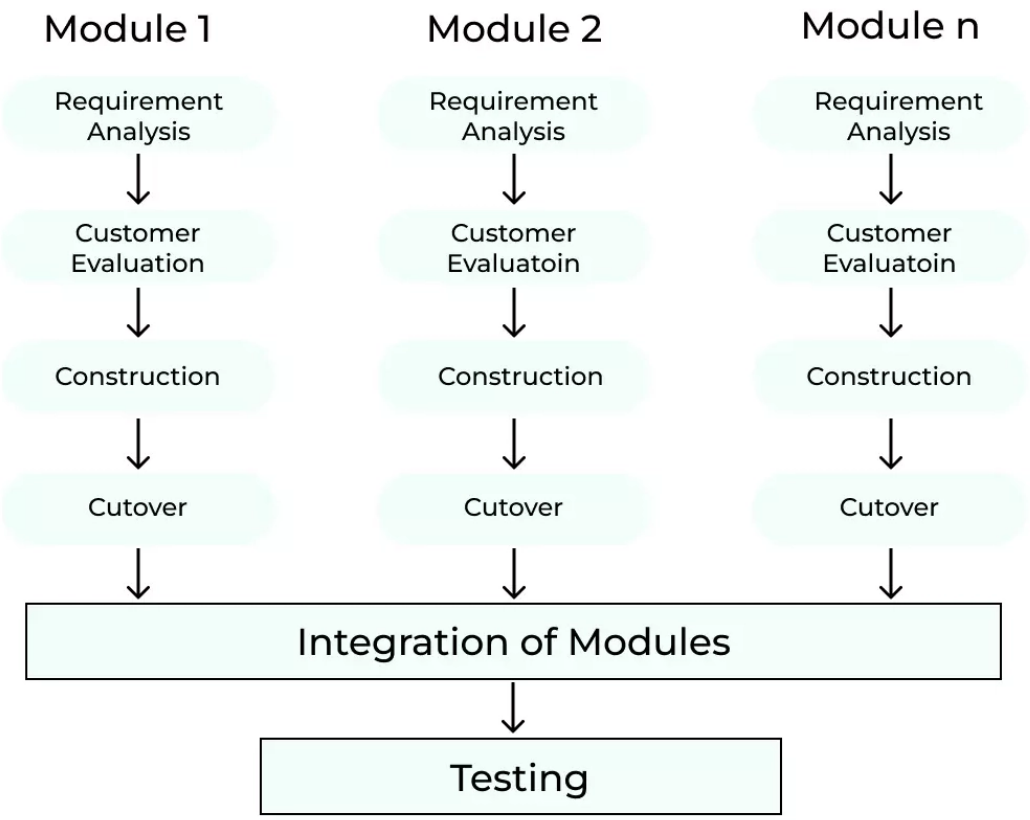
**When to use RAD Model?**

1. When the system must produce a project that can be divided into smaller units in a short amount of time (between two and three months).
2. After knowing the prerequisites.
3. If there is minimal technical danger.
4. When the creation of a system that was modularized over a period of two to three months is necessary.
5. It should only be utilized if the budget permits the use of automated tools for generating code.

**Phases of RAD Model**

- Business Modelling: By addressing issues like what data drives the business process, what data is generated, who generates it, where does the information go, who processes it, and other related issues, business modeling can define the information flow among business functions.
- Data modelling: A set of data objects (entities) that are required to support the business are created from the data gathered during business modeling. Each entity's characteristics (attributes) are recognized, and the relationships between these data objects (entities) are specified.
- Process Modelling: Process modeling is the transformation of the information objects described in the data modeling phase to create the data flow required to implement a business function. For adding, changing, removing, or retrieving a data object, processing descriptions are created.
- Application Generation: Tools that are automated and even use 4th GL approaches are used to make it easier to build applications.
- Testing and Turnover: Given that RAD places a strong emphasis on reuse, many of the programming components have previously undergone testing. Overall testing time is shortened as a result. But all interfaces must be thoroughly used, and the new component must be tested.

| **Advantages** | **Disadvantages** |
|---|---|
| 1. Change can be accommodated by this model.<br>2. A change can be adopted in this model.<br>3. Each RAD phase offers the customer the functionality that is of the utmost priority.<br>4. Dev time was shortened.<br>5. As a result, features are more reusable. | 1. The designers had to be extremely talented.<br>2. RAD incompatibility applies to all applications.<br>3. The RAD paradigm is inapplicable to smaller projects.<br>4. It's not suited due to the considerable technical danger.<br>5. essential user participation. |




## 6.1.8 PROTOTYPE MODEL

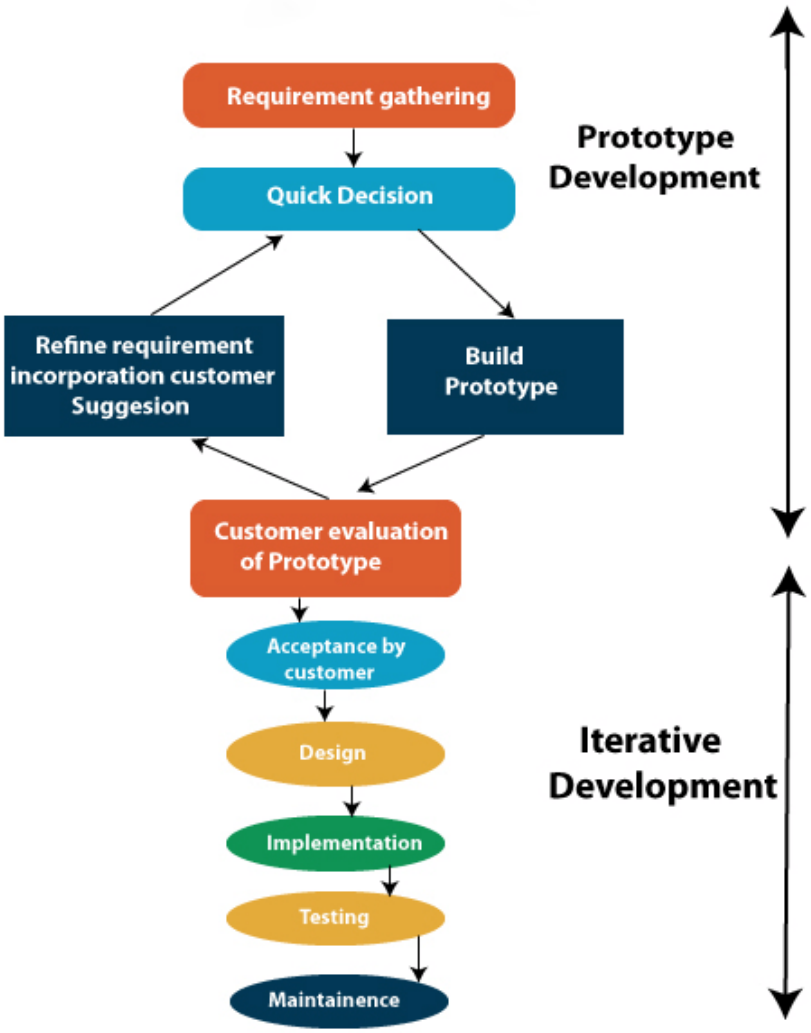
The working system prototype is a requirement of the prototype model before moving forward with the creation of actual software. The system's toy version is called a prototype. A prototype typically ends up being a very rudimentary representation of the real system, sometimes showing constrained functional capabilities, poor reliability, and ineffective performance when compared to real software. The client frequently only has a broad understanding of what is anticipated from the software solution. In a situation like this, when there is a lack of specific information on the input to the system, the processing requirements, and the output requirement, the prototyping approach may be used.

**When to use Prototype Model?**

1. When the needs for the product are unclear or unpredictable, the prototyping model should be used.
2. If requirements are changing quickly, the prototype model can also be used.
3. This paradigm can be successfully used to the creation of user interfaces, software-intensive high-tech systems, and systems with intricate interfaces and algorithms.
4. A very good option to show the product's technical viability is the prototyping model.

**Phases of Prototype Model**

- Requirement Gathering and Analyst
- Quick Decision
- Build a Prototype
- Assessment or User Evaluation
- Prototype Refinement
- Engineer Product




| Advantages | Disadvantages |
|---|---|
| 1. Minimize the possibility of inaccurate user requirements<br>2. Where the requirement is flexible or uncommitted, good<br>3. Regular, obvious processes support management.<br>4. Assistance with early product marketing<br>5. Spend less on maintenance.<br>6. As the system is created side by side, errors can be discovered considerably earlier. | 1. The final product frequently evolves from an unstable or poorly designed prototype.<br>2. substantial customer collaboration is necessary<br>3. costs the customer money and demands a dedicated client<br>4. challenging to complete if consumer withdraws<br>5. Possibly too niche and with a small market<br>6. It's challenging to predict how long the project will take.<br>7. Without adequate requirement analysis, design, customer evaluation, and feedback, it is simple to slip back into the code and make fixes.<br>8. Tools for prototyping are pricey.<br>9. To construct a prototype, specialized equipment and methods are needed.<br>10. It takes a long time to complete. |

## 6.2 THE SUITABLE SDLC MODEL FOR NEONTECH

### 6.2.1 THE MODEL USED

Given that NeonTech's website development has distinct project requirements that may benefit from a combination of several SDLC methodologies, a hybrid SDLC (Software Development Life Cycle) model may be a good option. For NeonTech's website development, the following hybrid model is suggested:

- **Requirements Gathering (Waterfall Phase)**
  - Use the Waterfall methodology to start by gathering requirements. Work closely with the customer to fully assemble and record all of the initial needs, objectives, and limitations for the website. The project scope, requirements document, and a comprehensive knowledge of the website's key features and functionalities will all be produced as a result of this phase.
- **Agile Development Phase (Iterative and Incremental)**
  - Enter an Agile development phase using an iterative and incremental methodology after the initial requirements have been acquired.
  - Break the project up into shorter, more manageable sprints, which are typically between two and four weeks long.
  - In each iteration, develop and deliver functioning increments of the website, putting the highest priority features first.
  - Regularly interact with the client or stakeholders to get their input and make any necessary improvements.
  - Give flexibility in feature development a priority while prioritizing features based on user needs and market trends.
- **Integration and Testing (Hybrid Phase)**
  - To preserve the stability and quality of the website during the Agile development period, ensure continuous integration and testing.
  - Implement the newly created features and carry out integration testing to ensure that every component functions properly.
  - To speed up testing and spot problems early, use automated testing tools.
- **User Acceptance Testing (Waterfall Phase)**
  - Return to a Waterfall-like phase for User Acceptance Testing (UAT) after all main features have been created and integrated.
  - Give the customer or end users a production-like environment to extensively test the website's functionality.
  - Make that the website performs as expected by the client and fixes any problems or bugs found during UAT.
- **Deployment and Maintenance (Hybrid Phase)**
  - Deploy the website to a live environment following a successful UAT.
  - Maintain and upgrade systems using Agile principles, making frequent iterations to handle problem repairs, enhancements, and new feature requests.
  - Continue to adapt the website to changing user demands and market trends.
- **Documentation and Knowledge Transfer (Throughout)**
  - Keep records of all project-related requirements, design choices, and technical information.
  - To ensure efficient teamwork and support, ensure knowledge transfer among team members.

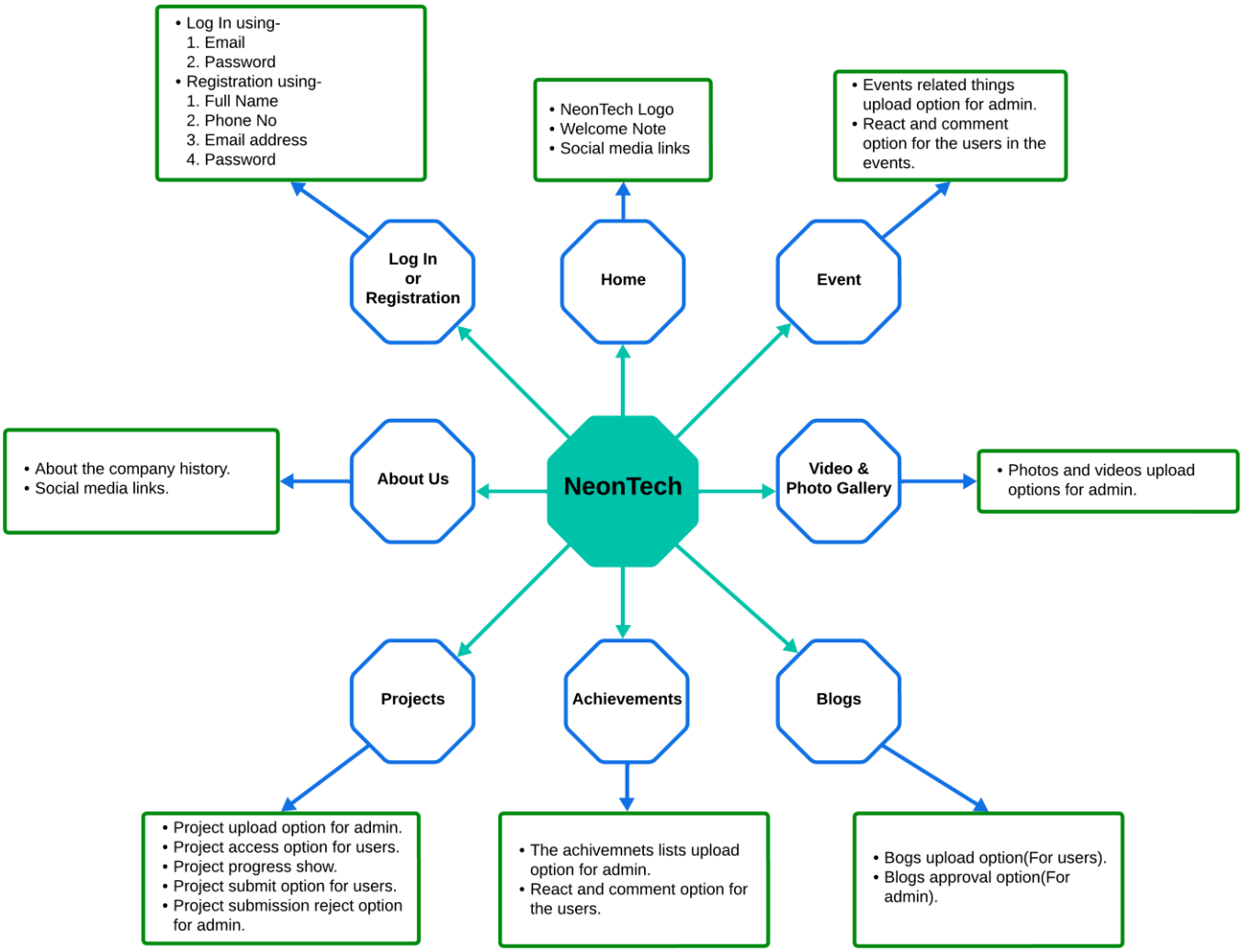
NeonTech benefits from systematic planning and thorough initial requirements (Waterfall) as well as flexibility and adaptation to changing project needs (Agile) using a hybrid model that combines Waterfall and Agile aspects. This method works for website development projects with varied levels of complexity and uncertainty because it allows for a balance between thorough planning and the capacity to react to changes.

### 6.2.2 WHY HYBRID SDLC IS GOOD FOR NEONTECH

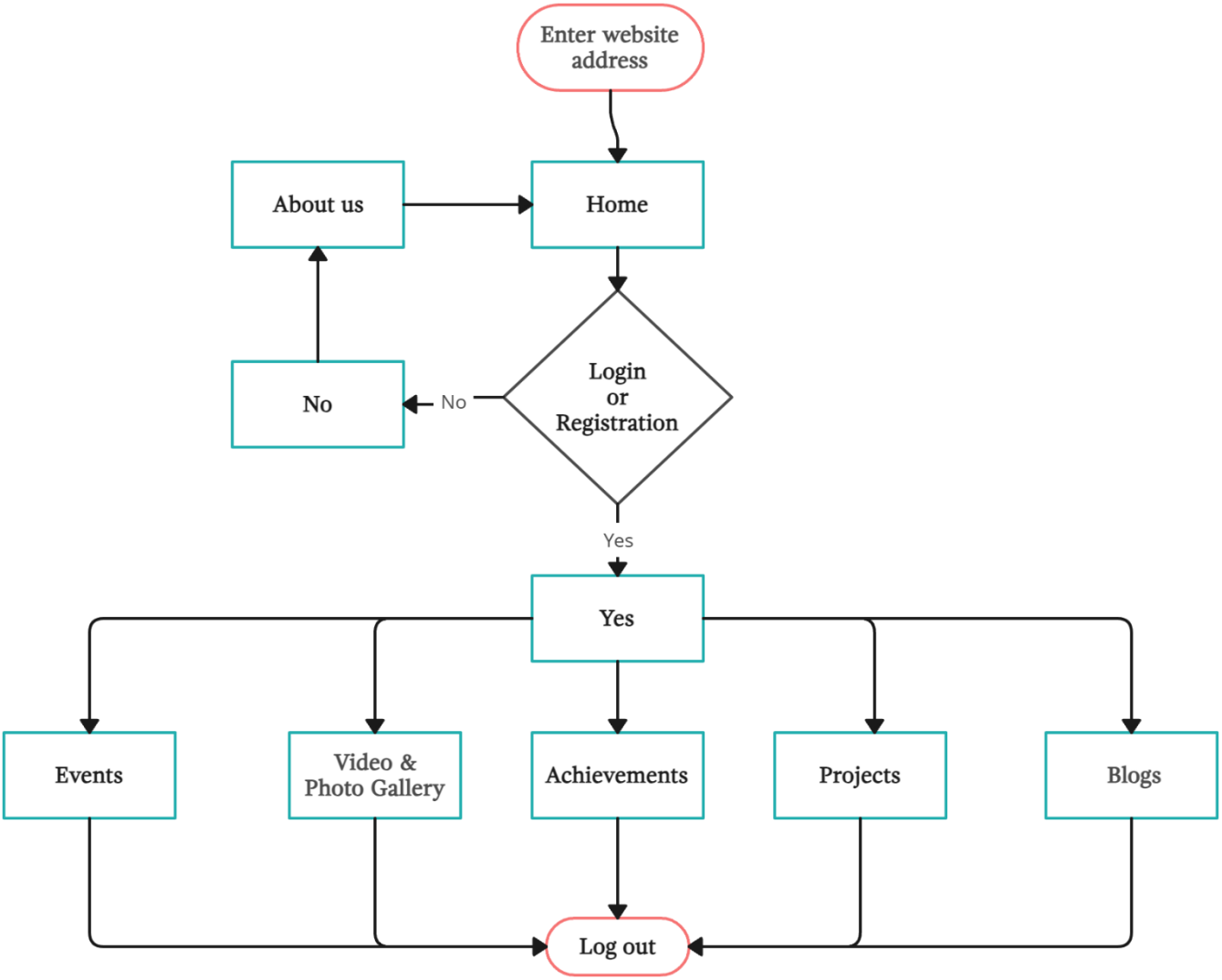
- Bespoke Approach: NeonTech can profit from a bespoke strategy that blends the virtues of the Waterfall and Agile project management approaches to meet the demands of their particular project.
- Initial Clarity (Waterfall Phase): The project's initial Waterfall phase makes sure NeonTech thoroughly collects and documents all of the project's requirements. To build a strong foundation and comprehend the website's scope, it is essential to have this first clarity.
- Flexibility (Agile Phase): Entering the Agile phase enables NeonTech to adjust to shifting needs, which are typical in web development projects. As a result, their website becomes more centered around the needs of their customers by enabling them to react swiftly to customer input and changing market demands.
- Incremental Development: NeonTech can produce functional website portions early thanks to the Agile phase's iterative and incremental approach, guaranteeing that they have a usable solution sooner. By focusing on the most important features first, this is helpful for demonstrating progress to stakeholders.
- Risk reduction: The Hybrid approach lessens the possibility that serious flaws will be found after the project has already begun by including testing into the whole development process. Better quality control results as a result.
- User Involvement: Agile iterations promote client and user involvement at every stage of the development process, generating a sense of ownership and teamwork. In doing so, NeonTech is able to make sure that the website closely meets user expectations.
- Structured Testing: The return to a Waterfall-like phase for User Acceptance Testing (UAT) ensures a thorough assessment of the website in a controlled setting. This reduces the likelihood that significant problems may emerge upon deployment.
- Balanced Documentation: The approach maintains documentation throughout the project, integrating thorough documentation from the Waterfall phase with Agile's focus on collaboration and regular communication.
- Adaptive Maintenance: Following deployment, NeonTech can continue using Agile techniques for ongoing upkeep and updates. This enables them to respond to customer feedback, make adjustments, and maintain their competitiveness in a continuously evolving digital landscape.
- Resource Efficiency: By concentrating Waterfall's formal planning on the first phase of the project and leveraging Agile's iterative development for the remaining phases, the Hybrid approach optimizes resource allocation.

In general, the Hybrid Model presents NeonTech with a balanced strategy that integrates the benefits of both the Waterfall and Agile techniques, making it appropriate for website development projects that call for a blend of organized planning and flexibility to respond to changing conditions. It can assist NeonTech in building a top-notch, client-focused website that satisfies their corporate goals while keeping adaptable to changing user requirements and market trends.

## 7. MIND MAP

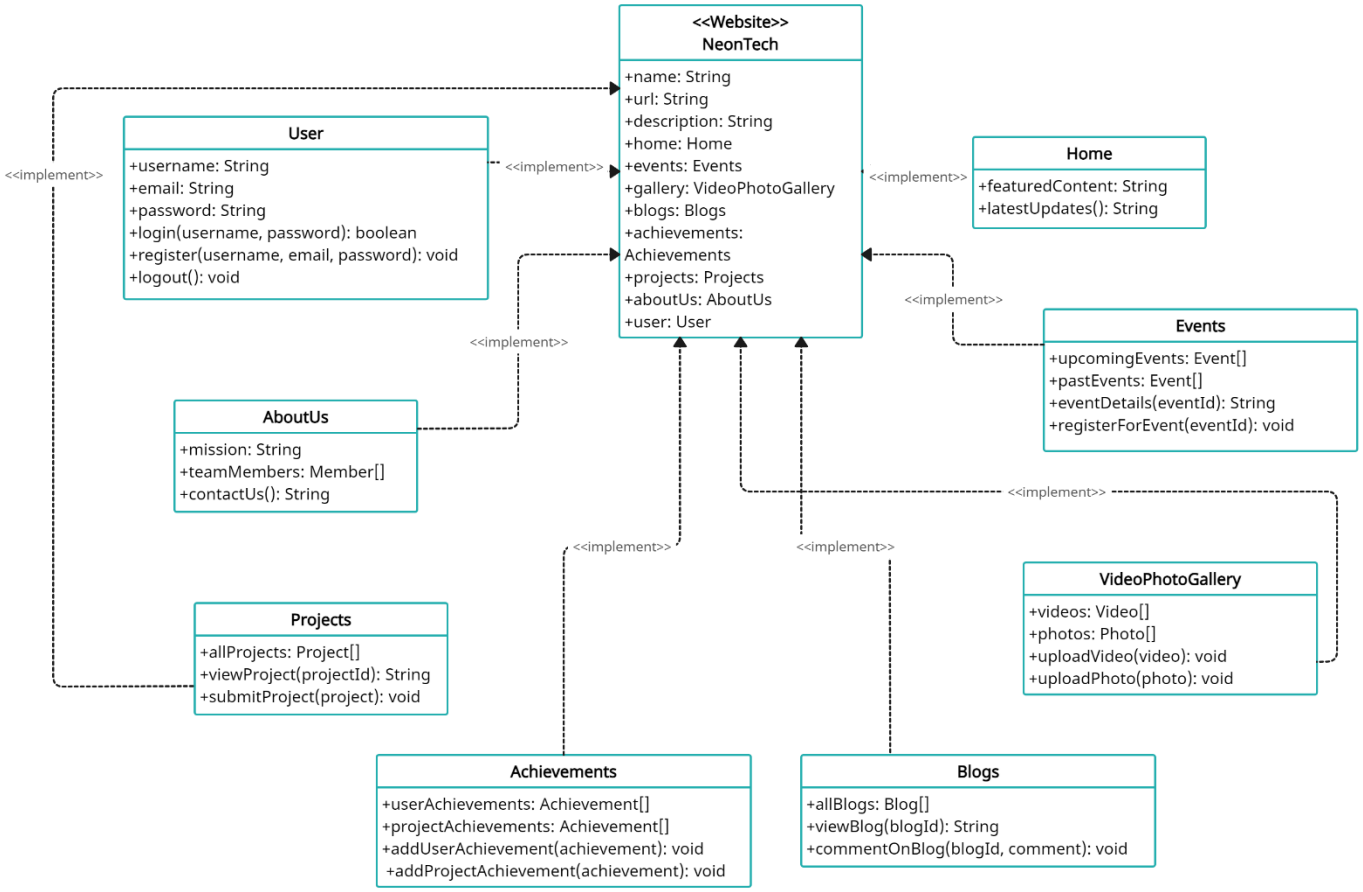

NeonTech
Log In or Registration
• Log In using-
1. Email
2. Password
• Registration using-
1. Full Name
2. Phone No
3. Email address
4. Password
Home
• NeonTech Logo
• Welcome Note
• Social media links
Event
• Events related things upload option for admin.
• React and comment option for the users in the events.
About Us
• About the company history.
• Social media links.
Video & Photo Gallery
• Photos and videos upload options for admin.
Projects
• Project upload option for admin.
• Project access option for users.
• Project progress show.
• Project submit option for users.
• Project submission reject option for admin.
Achievements
• The achivemnets lists upload option for admin.
• React and comment option for the users.
Blogs
• Bogs upload option(For users).
• Blogs approval option(For admin).

## 8. FLOWCHART


Enter website address
About us
Home
No
No
Login or Registration
Yes
Yes
Events
Video & Photo Gallery
Achievements
Projects
Blogs
Log out

## 9. USE CASE


## 10. UML



## 11. DEPLOYMENT

NeonTech website will completely base on the requirements which is given by your company. The features those are requested are already write down. If your company wants more feature, then you have to pay base on the new features.

## 12. TESTING

### 12.1 PROPOSED TESTING APPROACH

In proposing a testing approach for NeonTech compared to others, one viable option could be an Agile testing approach. Let's delve into the specifics:

**Proposed Testing Approach:**

**Agile Testing Approach for NeonTech-**

| Agile Testing Approach | Benefit | Implementation Strategy |
|---|---|---|
| Iterative and Incremental Testing | Accommodates changes efficiently | Break down testing into short cycles, ensuring alignment with development increments. |
| Collaboration and Communication | Enhances communication and issue resolution | Conduct regular meetings, such as daily stand-ups and sprint reviews. |
| User-Centric Testing | Ensures a user-friendly experience | Regularly involve users, user stories, and acceptance criteria in testing efforts. |
| Flexibility to Change | Accommodates shifts without compromising testing | Emphasize adaptability, seamlessly incorporate changes, and prioritize requirements based on evolving priorities. |
| Automation Testing | Accelerates testing and improves efficiency | Automate repetitive test cases to free up time for exploratory and user testing. |
| Continuous Monitoring | Identifies and resolves issues promptly | Implement continuous monitoring tools to track performance and identify potential problems. |
| Performance Testing | Ensures the system can handle expected load | Conduct performance testing throughout the development lifecycle to identify and address performance bottlenecks. |
| Security Testing | Protects user data and ensures system integrity | Perform security testing at each iteration to identify and remediate vulnerabilities. |
| Usability Testing | Validates user experience and satisfaction | Conduct usability testing throughout the development lifecycle to gather feedback and make improvements. |
| Release Management | Manages the release process and ensures quality | Implement a robust release management process to control and validate releases. |
| Testing Tools and Infrastructure | Employs appropriate tools and infrastructure | Utilize a variety of testing tools and infrastructure to support the testing process. |
| Reporting and Metrics | Tracks progress and identifies areas for improvement | Collect and analyze testing metrics to track progress, identify areas for improvement, and inform future testing strategies. |

| | | |
|---|---|---|
| Continuous Improvement | Continuously improves the testing process | Implement a culture of continuous improvement to refine testing practices and enhance testing effectiveness. |

**Comparison with Other Approaches:**

| Model | Characteristics | Suitability for NeonTech |
|---|---|---|
| Waterfall | Sequential, rigid, less adaptable to changing requirements | Not suitable for NeonTech's dynamic environment with evolving requirements. |
| Agile | Flexible, adaptive, iterative, allows for continuous improvement and responsiveness to changes | Suitable for NeonTech's need for quick releases, adaptability, and user-centric development. |
| V-Model | Linear, relies heavily on documentation | Not suitable for NeonTech's dynamic environment and emphasis on collaboration and quick iterations. |
| Spiral Model | Iterative but with longer development cycles | May not align with NeonTech's need for quick releases and frequent iterations. |

## 12.2 TESTING LEVELS

The methodical process of assessing a software program or system at various phases of its development life cycle is known as testing levels. Every testing stage has a distinct function, and when taken as a whole, they help guarantee the software's dependability and quality. Let's discuss the testing levels applicable to NeonTech:

**Testing Levels for NeonTech:**

| Testing Levels | Objective | Scope for NeonTech |
|---|---|---|
| Unit Testing | To validate individual units or components of the software. | Testing functions, methods, or modules within the codebase.<br>Ensuring that each unit performs as expected. |
| Integration Testing | To verify the interactions and interfaces between integrated components or systems. | Testing the seamless integration of different modules (e.g., Home, Events, Blogs) to identify and rectify any issues arising from their interaction.<br>Ensuring data flow and communication between various sections. |
| System Testing | To evaluate the entire system as a whole. | Testing the complete website to ensure all functionalities work together seamlessly.<br>Verifying the system against defined requirements and specifications. |
| User Acceptance Testing (UAT) | To ensure that the system meets end-users' requirements and expectations. | Involving actual end-users to validate the overall usability and functionality of the website.<br>Confirming that the website aligns with user needs and preferences. |

## 12.3 TESTING TYPES, TECHNIQUES AND TACTICS

To provide thorough test coverage and identify potential problems in software applications, testing types, methodologies, and tactics are essential components. Many testing kinds, strategies, and approaches can be used for NeonTech, a tech-focused website. Let's explore some relevant ones:

**Testing Types for NeonTech:**

| Testing Type | Objective | Techniques/Tactics |
|---|---|---|
| Functional Testing | To validate that each function of NeonTech operates according to specifications. | Equivalence Partitioning, Boundary Value Analysis |
| Performance Testing | To assess the responsiveness, scalability, and overall performance of NeonTech. | Load Testing, Stress Testing |
| Security Testing | To identify vulnerabilities and ensure the security of user data. | Penetration Testing, Code Review |
| Usability Testing | To evaluate the user-friendliness and overall user experience of NeonTech. | User Surveys, User Interviews |
| Compatibility Testing | To ensure NeonTech functions correctly across different browsers and devices. | Cross-Browser Testing, Device Testing |
| Regression Testing | To verify that new updates or changes do not negatively impact existing functionalities. | Automated Regression Testing, Selective Regression Testing |

**Techniques and Tactics for NeonTech:**

| Techniques and Tactics | Objective | Techniques/Tactics |
|---|---|---|
| Automation Testing | To enhance testing efficiency and coverage. | Selenium for Web UI Automation, API Testing with tools like Postman or Rest Assured |
| Exploratory Testing | To discover unforeseen issues by exploring the application. | Ad-hoc Testing, Session-Based Testing |
| Static Testing | To identify issues without executing the code. | Code Review, Static Analysis Tools |
| Risk-Based Testing | To prioritize testing efforts based on identified risks. | Risk Assessment, Focused Testing |

## 12.4 PROPOSED TESTING PROCESS

Proposing a testing process for NeonTech involves defining a systematic and comprehensive approach to ensure the quality and reliability of the website. Below is a suggested testing process tailored for NeonTech:

**NeonTech Testing Process:**

| Phase | Objective | Activities |
|---|---|---|
| Requirements Analysis | Understand and analyze the requirements for each module and functionality of NeonTech. | Review user stories, specifications, and design documents. Collaborate with stakeholders to clarify ambiguities and gather additional information. |
| Test Planning | Develop a comprehensive test plan outlining testing scope, objectives, resources, and schedule. | Identify testing types and levels suitable for NeonTech. Define entry and exit criteria for each testing phase. Allocate resources and set up testing environments. |
| Test Design | Create detailed test cases based on the requirements and design specifications. | Develop positive and negative test scenarios for each functionality. Include input combinations, boundary cases, and user flows. Ensure test cases cover various testing types such as functional, performance, security, and usability. |
| Test Execution | Execute the prepared test cases and scenarios in the designated testing environments. | Perform unit testing for individual components. Conduct integration testing to verify the interaction between modules. Execute system testing to ensure the entire website functions as intended. Implement user acceptance testing involving actual end-users. |
| Defect Reporting | Log and track identified issues to facilitate their resolution. | Document defects with detailed information, including steps to reproduce. Prioritize and assign severity levels to each defect. Communicate issues to the development team for resolution. |
| Regression Testing | Ensure that new updates or changes do not negatively impact existing functionalities. | Re-run existing test cases impacted by recent changes. Automate regression tests to expedite the testing process. Validate the stability of the overall system. |
| Performance and Security Testing | Assess the website's responsiveness, scalability, and security. | Conduct performance testing to analyze website speed, load capacity, and response times. Perform security testing to identify vulnerabilities and ensure data protection. |
| Usability Testing | Evaluate the user-friendliness and overall user experience. | Engage end-users in usability testing to gather feedback on the website's design and functionality. Make adjustments based on usability test results. |
| Release Testing | Validate the readiness of the website for release. | Confirm that all test cases have been executed and passed. Obtain final approval from stakeholders for the release. |
| Continuous Improvement | Gather insights from the testing process to enhance future testing efforts. | Conduct a retrospective meeting to discuss successes and areas for improvement. Update test cases, processes, and tools based on lessons learned. Incorporate feedback from end-users and stakeholders for future releases. |

## 12.5 MEASUREMENT IN SOFTWARE TESTING

Measurement in software testing is crucial for evaluating the effectiveness of the testing process, identifying areas for improvement, and ensuring the overall quality of the software. For NeonTech, incorporating relevant metrics and measurements is essential. Here are some key measurements in software testing:

**Measurement Metrics for NeonTech:**

| Metric | Definition | Purpose for NeonTech |
|---|---|---|
| Defect Density | Number of defects per unit size of the code. | Assess the quality of the codebase and identify areas prone to defects. |
| Test Coverage | Percentage of code or functionalities covered by testing. | Ensure critical functionalities are thoroughly tested, reducing the risk of undetected issues. |
| Regression Test Pass Rate | Percentage of regression tests that pass after a code change. | Evaluate the stability of the system and the impact of recent changes on existing functionalities. |
| Bug Fixing Time | Time taken to fix identified bugs from the time of discovery. | Assess the efficiency of the development and testing teams in addressing reported issues promptly. |
| Test Execution Time | Time taken to execute the entire test suite. | Evaluate testing efficiency and identify opportunities for optimization. |
| Test Case Effectiveness | Percentage of executed test cases that identified defects. | Assess the efficiency of test cases in uncovering issues and improve the overall testing strategy. |
| User Satisfaction with Testing | Feedback from end-users regarding the quality and performance of the software. | Gather insights into user experience and satisfaction, guiding future development and testing efforts. |
| Automation Test Coverage | Percentage of test cases automated compared to the total number of test cases. | Assess the coverage and efficiency of automated testing efforts. |

**Hierarchy of Testing Difficulty:**

In the hierarchy of testing difficulty, the complexity and difficulty of testing typically increase as we move from unit testing to system testing and then to acceptance testing. Here's a general outline:

| Test Type | Difficulty Level | Scope | Focus | Tools |
|---|---|---|---|---|
| Unit Testing | Relatively Low | Testing individual units or components in isolation. | Verify that each unit of code performs as intended. | Automated testing frameworks (e.g., JUnit, NUnit) |
| Integration Testing | Moderate | Testing interactions between integrated components or systems. | Verify data flow, communication, and integration points. | Integration testing frameworks (e.g., TestNG, Jest) |
| System Testing | Moderate to High | Testing the entire system as a whole. | Ensure the system meets specified requirements and works as intended. | Automated testing tools (e.g., Selenium for web applications) |
| User Acceptance Testing (UAT) | High | Testing the system from an end-user perspective. | Validate that the system satisfies user needs and expectations. | Manual testing, User feedback mechanisms |

**Test Plan:**

A Test Plan is a comprehensive document that outlines the testing approach, objectives, resources, schedule, and deliverables. It provides a roadmap for the testing process. Here's a suggested structure for NeonTech's Test Plan:

| Section | Description |
|---|---|
| Introduction: | Provides an overview of the NeonTech project, the purpose of the test plan, and its scope |
| Test Objectives: | Clearly defines the testing objectives for each testing phase (Unit, Integration, System, UAT) |
| Testing Strategy: | Describes the overall testing strategy (e.g., Agile) and outlines the test levels and types to be performed |
| Test Schedule: | Presents a project timeline with milestones and identifies the testing phases |
| Resource Allocation and Availability: | Details the resource allocation and availability for testing activities |
| Testing Environment: | Specifies the hardware and software configurations required for testing |
| Tools and Technologies Used for Testing: | Lists the tools and technologies that will be used for testing |
| Test Deliverables: | Defines the expected test deliverables (test cases, test scripts, reports) and establishes criteria for their acceptance |
| Test Execution: | Provides a detailed plan for executing tests during each testing phase and identifies the responsible team members |
| Defect Management: | Outlines the process for reporting, tracking, and managing defects, including escalation procedures for critical issues |
| Risks and Contingencies: | Identifies potential risks and their mitigation strategies, as well as contingency plans for unexpected issues |
| Review and Approval: | Establishes the process for reviewing and obtaining approval for the test plan, defining sign-off criteria for each testing phase |

**Test Case:**

A Test Case is a detailed document that outlines the steps, inputs, expected outcomes, and conditions for executing a specific test. Here's a suggested structure for NeonTech's Test Case:

| Field Name | Description |
|---|---|
| Test Case ID: | A unique identifier for the test case, often assigned sequentially or using a specific naming convention. |
| Test Case Title: | A concise and descriptive title that clearly conveys the purpose of the test. |
| Objective: | A clear statement of what the test is intended to verify, addressing a specific functionality or requirement. |
| Preconditions: | Any necessary conditions that must be met before the test can be executed, such as system configurations, data setups, or external dependencies. |
| Inputs: | The data, conditions, or parameters that will be provided to the system during test execution. |
| Steps: | A step-by-step list of instructions for executing the test, including specific actions, expected outcomes, and any relevant conditions or checkpoints. |
| Expected Results: | A clear definition of the expected outcomes for the test, including the desired system behavior or output. |

| | |
|---|---|
| Criteria for pass or fail: | Specific criteria that will be used to determine whether the test has passed or failed, based on the expected results. |
| Actual Results: | A space to record the actual results observed during test execution, providing details of the system's behavior or output. |
| Pass/Fail: | A final assessment of the test outcome, indicating whether it passed or failed based on the defined criteria. |
| Notes: | Any additional observations, comments, or insights gained during test execution. |

## 13. SUPPORT

- 24/7 support based on payment.

## 14. PRICING

My fee for seeing the project through from start to completion will be Thirty-Five thousand Taka only (35,000Tk).

## 15. PAYMENT TERMS

Here are the proposed payment terms:

| Payment Milestone | Percentage (%) | Payment Due |
|---|---|---|
| Acceptance of Proposal | 10% | On Proposal Acceptance |
| Website Development Agreement | 40% | On Agreement Signing |
| 70% Website Demonstration | 25% | At 70% Demonstration |
| Website Completion | 25% | Upon Website Completion |

## 16. RESPONSIBILITY

This website Ordered by **Nurul Amin**, Managing Director of NeonTech, all responsibility goes on him.

This website will Developed by Shrikanta Paul.

## 17. CONTACT ME

You can get in touch with us in any of the below ways:

**By Phone:**
+8801985064618
Shrikanta Paul

**By Email:**
paul15-4868@diu.edu.bd

I eagerly anticipate your prompt response.

## AGREEMENT SIGNED BY:

…… ……

**Client Signature**
Nurul Amin
Managing Director, NeonTech

**Authority Signature**
Shrikanta Paul