**Assignment – 2**

| Student Name/ID Number: |  |
| --- | --- |
| Unit Number and Title: | Develop Project Proposal |
| Academic Year: | 2022-2023 |
| Unit Assessor: | Arvinder Kaur - Mentor |
| Project Title: | Develop Enterprise Applications |
| Issue Date: | 06-23-2023 |
| Submission Date: |  |
| Internal Verifier Name: | Arvinder Kaur |
| Date: | 06-23-2023 |

| **Learner declaration** |
| --- |
|  |
| I certify that the work submitted for this assignment is my own and research sources are fully acknowledged.  Student signature: Date: 06-23-2023 |

1. **System Requirements**

**1.1. Software Development Life Cycle (SDLC)**

-It is a systematic and structured approach used to develop software applications. The SDLC defines a series of phases or stages that guide the entire software development process, from the initial concept and requirements gathering to the final deployment and maintenance of the software.

**How does SDLC work?**

-The Software Development Life Cycle, sometimes known as SDLC, is a framework that directs the creation of software programs. In order to guarantee that the software is developed effectively and satisfies the desired criteria, it adopts an organized and systematic method. Here is an explanation of the SDLC process:

1. **Requirement Analysis-** The first step in SDLC involves gathering and analyzing the software requirements. This includes understanding the needs of the stakeholders, identifying the functionalities the software should have, and documenting the requirements.
2. **System Design-** Once the requirements are gathered, the system design phase begins. It involves designing the architecture of the software, specifying the components and their interactions, and creating a high-level design document. The design phase lays the foundation for the implementation phase.
3. **Implementation-** In this phase, the actual coding and development of the software take place. Programmers write the source code based on the design specifications and best coding practices. This phase may involve various programming languages, frameworks, and tools depending on the project requirements.
4. **Testing-** After the software is implemented, it goes through a series of testing phases. Different types of testing, such as unit testing, integration testing, system testing, and user acceptance testing, are performed to identify any defects or issues. Testing ensures that the software functions as intended and meets the specified requirements.
5. **Deployment-** Once the software passes the testing phase, it is deployed or released for use. This involves activities like installation, configuration, and making the software available to users. Deployment may include setting up servers, configuring databases, and ensuring the software runs smoothly in the production environment.
6. **Maintenance-** After deployment, the software enters the maintenance phase. It involves monitoring the software's performance, addressing any issues or bugs that arise, and making necessary updates or enhancements. Maintenance may include bug fixing, adding new features, and providing ongoing support to users.

Overall, SDLC offers a structured and controlled approach to software development, guaranteeing that the software is produced quickly, precisely, and to a high standard.

**1.2. SDLC Models**

**1.2.1. Waterfall Model**

* **What is Waterfall Model?**

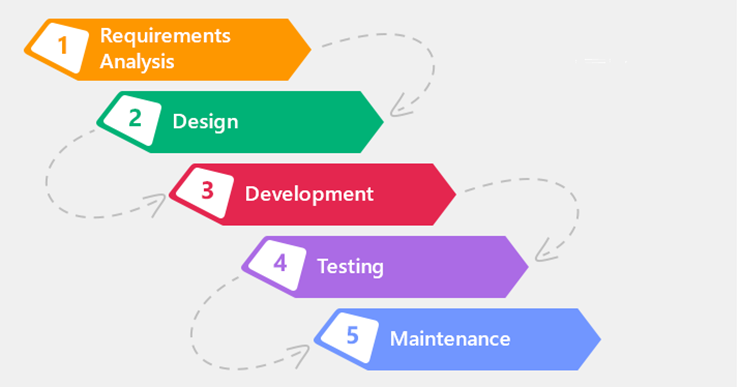
-It is a conventional strategy for developing software that takes a sequential and linear approach. Because each stage of the development process cascades into the next in a way akin to a waterfall, it is known as the waterfall model. Once a phase is ended, it usually doesn't return to the previous phase. The phases are finished one after the other.

* **Characteristics**

-The Waterfall model, a sequential software development methodology, is characterized by the following key features:

1. **Sequential and Linear-** The Waterfall model follows a sequential and linear approach, where each phase is completed before moving on to the next. There is a clear flow of activities from one phase to another, resembling a waterfall.
2. **Well-defined Phases-** The model consists of distinct and well-defined phases, including requirements gathering, system design, implementation, testing, deployment, and maintenance. Each phase has specific objectives and deliverables.
3. **Emphasis on Documentation-** The Waterfall model emphasizes comprehensive documentation throughout the development process. Documentation includes requirements documents, design documents, test plans, and user manuals. This documentation helps ensure clarity, traceability, and knowledge transfer between project stakeholders.
4. **Fixed Requirements-** The model assumes that the requirements are stable and well-understood at the beginning of the project. It follows a "big design upfront" approach, where the requirements are gathered and documented before moving to subsequent phases. Changes to requirements are discouraged once the development process begins.
5. **Limited Customer Involvement-** In the Waterfall model, customer involvement is typically limited to the initial requirement gathering phase and user acceptance testing. Customers provide requirements and feedback during these stages, but their involvement in the development process is reduced in subsequent phases.
6. **Minimal Iteration-** The Waterfall model typically involves minimal iteration or going back to previous phases. Once a phase is completed, it is expected to be stable and not revisited. This characteristic makes the model less flexible to accommodate changes or evolving requirements.
7. **Appropriate for Predictable Projects-** The Waterfall model is suitable for projects where the requirements are well-defined, stable, and predictable. It works well for small or straightforward projects that have clear objectives and limited chances of requirement changes.
8. **Sequential Progress Tracking-** The Waterfall model allows for easy progress tracking and milestone-based evaluation. Each phase has defined deliverables, and progress can be measured by assessing the completion of these deliverables.
9. **High Upfront Planning-** The Waterfall model requires thorough planning and analysis upfront. The entire project is typically planned and estimated before the development starts. This planning includes defining the scope, schedule, and resource requirements.
10. **Limited Risk Mitigation-** The Waterfall model carries a higher risk of discovering issues or defects late in the development process. As testing is conducted towards the end of the project, potential risks or problems may only be identified at that stage, leading to potential delays or rework.

* **Waterfall Model Structure**

****

* **Where do we use Waterfall Model?**

-Waterfall Model must only be used on the following circumstances:

1. **Well-Defined Requirements-** When the project has clear, stable, and well-understood requirements from the beginning, the Waterfall model can be a suitable choice. It works best when the requirements are unlikely to change significantly during the development process.
2. **Sequential Dependencies-** If the project has strict sequential dependencies, where one phase must be completed before moving to the next, the Waterfall model can be effective. It is useful when there is a clear order and hierarchy of tasks that need to be followed.
3. **Predictable and Small-Scale Projects-** The Waterfall model is often used for small-scale projects with limited complexity. If the project is relatively simple and the development team has previous experience or expertise in similar projects, the Waterfall model can be a practical choice.
4. **Regulatory or Compliance Requirements-** In certain industries or projects with strict regulatory or compliance requirements, the Waterfall model is preferred. It allows for comprehensive documentation and traceability, which are essential for compliance purposes.
5. **Fixed Budget and Schedule-** When the project has fixed budget and schedule constraints, the Waterfall model can be beneficial. It enables upfront planning and estimation, which helps manage resources and schedule adherence.
6. **Stakeholder Preference:** If stakeholders, such as clients or management, prefer a linear and sequential approach with well-documented deliverables, the Waterfall model may be chosen based on their requirements and expectations.

The Waterfall model may not be appropriate for projects with changing or confusing requirements, where frequent modifications are anticipated, or where ongoing customer interaction and feedback are required, it is crucial to emphasize. Iterative models or more flexible approaches like Agile are frequently preferred in these situations.

* **Pros and Cons**

| **Pros** | **Cons** |
| --- | --- |
| Clear Structure and Documentation | Limited Flexibility and Adaptability |
| Upfront planning and estimation | Late discovery of issues leading to delay or reworks |
| Easy progress Tracking | Limited customer involvement |
| Reduced customer involvement | Higher Risk of project failure |
| Resource Optimization | Long development cycles and limited early feedback |
| Well-suited for stable requirements | Difficulty in managing requirement changes |
| Regulatory compliance | Lack of collaboration with team or stakeholders. |

**1.2.2. Spiral Model**

* **What is Spiral Model?**

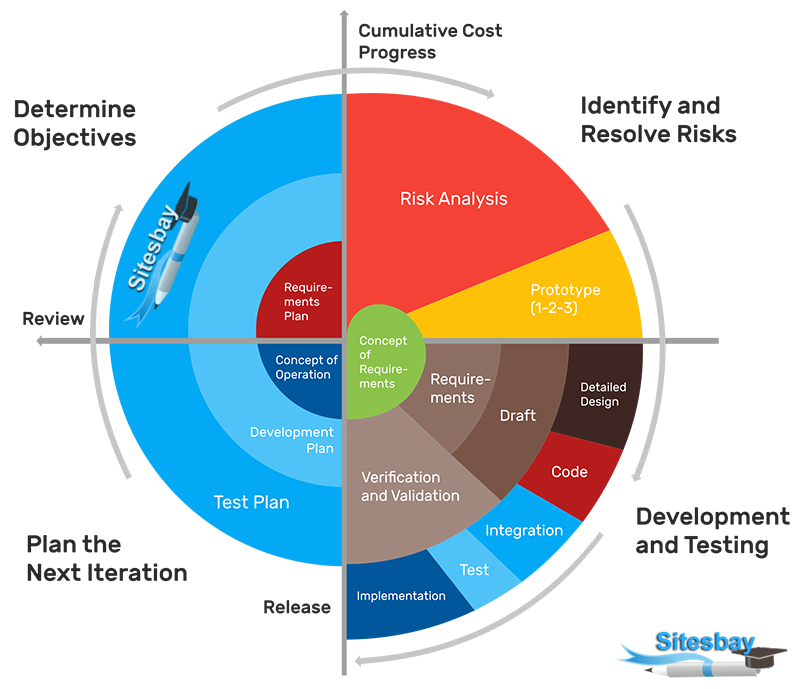
-The Spiral Model is an SDLC approach that blends features from the waterfall model and iterative development. It was created by Barry Boehm in 1986 and is specifically beneficial for complex projects with significant risks and uncertainties. This model highlights the iterative aspect of development and integrates feedback loops to tackle evolving requirements and risks.

* **Characteristics**

- The Spiral model is characterized by the following key features:

1. **Iterative Approach-** The Spiral Model takes an iterative approach, with each iteration or spiral representing a stage in the software development process. Planning, risk analysis, engineering, and evaluation are common phases.
2. **Risk Management-** Throughout the project, the model focuses a major emphasis on risk management. Each iteration begins with a risk analysis and assessment, allowing the team to identify and minimize any hazards early on.
3. **Progressive Elaboration-** The Spiral Model supports progressive elaboration, which means that project requirements and solutions are developed and evolved through time. As the project continues, this gives for a greater grasp of the system's intricacies and requirements.
4. **Feedback Loops-** The model includes feedback loops, which allow stakeholders to assess the system at each iteration and provide comments for future changes. This iterative feedback system aids in the management of changing requirements and the reduction of uncertainty.

* **Spiral Model Structure**

****

* **When do we use Spiral Model?**

-The Spiral Model is best suited for complex projects with high risks and uncertainties. It is typically used in the following scenarios:

1. **Large-Scale Projects-** The Spiral Model can successfully manage risks and fulfill developing requirements throughout the project's lifecycle when dealing with large-scale software development projects.
2. **Uncertain Requirements-** If the project requirements are not well-defined or change often, the Spiral Model supports developing requirements through iterative development and feedback loops.
3. **High Risk Projects-** Projects involving significant risks, such as safety-critical systems or projects involving new technologies, can benefit from the Spiral Model's risk management strategy. It enables the early detection and reduction of dangers.

* **Pros and Cons**

| **Pros** | **Cons** |
| --- | --- |
| The model provides a systematic approach to risk management, enabling early identification and mitigation of potential risks. | The Spiral Model requires experienced project managers and a competent development team to handle the complexities of risk management and iterative development. |
| The iterative nature of the model allows for flexibility in accommodating changing requirements and incorporating stakeholder feedback. | The iterative nature of the model may result in increased time and cost if the project scope expands significantly during the iterations. |
| The Spiral Model supports incremental development, which allows for early delivery of working software and faster feedback from stakeholders. | The model requires thorough documentation and review at each iteration, which can be time-consuming. |

**1.2.3. V Model**

* **What is V-Model?**

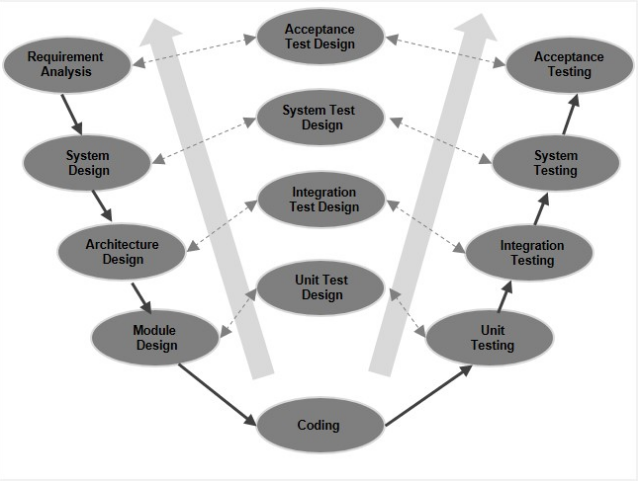
It is an SDLC model where execution of process happens in a sequential manner, in a V-shape. It is also known as Verification and Validation Model.

* **Characteristics**

- The V-Model is characterized by the following key features:

1. **Sequential and phased approach-** The V-Model follows a sequential and phased approach to software development. It consists of several distinct phases, each corresponding to a specific activity or deliverable. The development phases typically progress in a top-down manner, starting from the requirements phase and culminating in the deployment or maintenance phase.
2. **Emphasis on documentation-** The V-Model places a strong emphasis on documentation throughout the development process. Detailed documentation is prepared at each phase, including requirements specifications, design documents, test plans, and test cases. This documentation ensures clarity, traceability, and facilitates effective communication among project stakeholders.
3. **Verification and validation at each phase-** The V-Model incorporates the concept of verification and validation at each phase. Verification ensures that each phase's deliverables meet their specified requirements and adhere to the defined standards. Validation involves checking whether the final software product meets the user's needs and expectations. The corresponding testing phase for each development phase is an essential part of this process.
4. **Phased testing approach-** The V-Model features a phased testing approach, with testing activities aligned with each development phase. Testing activities start early in the lifecycle, with test planning and test case development taking place alongside the corresponding requirements and design activities. This approach ensures early detection and resolution of defects, reduces risks, and improves overall software quality.
5. **Clear and well-defined requirements-** The V-Model assumes that the project requirements are well-defined and understood before the development process begins. It relies on detailed requirements documentation to guide the subsequent phases of development and testing. The requirements serve as the foundation for design and testing activities, and any changes or modifications to requirements can have a significant impact on subsequent phases.
6. **Traceability and control-** The V-Model emphasizes traceability and control throughout the development lifecycle. It ensures that each phase's deliverables are traceable back to the corresponding requirements and that changes or modifications are managed effectively through a formal change control process. This traceability helps in maintaining project visibility, managing risks, and facilitating effective decision-making.
7. **Linear progression-** The V-Model follows a linear progression from requirements to testing, with each phase building upon the previous one. This sequential nature of the model allows for a clear understanding of dependencies and enables a structured and disciplined development process.

* **V Model Structure**

****

* **When do we use V-Model**

-V-Model is typically used in the following scenarios:

1. **Well-defined and stable requirements-** The V-Model works best when the project requirements are well-documented, stable, and unlikely to change significantly throughout the development process. It is suited for projects where the scope is clear and there is a low probability of requirement modifications.
2. **Sequential and structured approach-** The V-Model follows a sequential and phased approach, where each phase has a corresponding testing phase. It is suitable when there is a need for a structured and disciplined development process that ensures the systematic verification and validation of each phase.
3. **Emphasis on early planning-** The V-Model places a strong emphasis on planning and documentation. It requires detailed planning and preparation before the development starts, ensuring that all requirements, design, and test cases are well-defined in advance.
4. **Quality assurance and risk management-** The V-Model provides a framework for comprehensive testing and quality assurance activities. Each phase of the development process is paired with a corresponding testing phase, allowing for early detection and mitigation of defects and risks.
5. **Regulatory compliance-** In industries with strict regulatory requirements, such as healthcare or aerospace, the V-Model is often preferred. Its systematic and well-documented approach aligns well with regulatory standards and facilitates the traceability of requirements, design, and test artifacts.
6. **Waterfall-like projects-** The V-Model is sometimes used in projects that have a waterfall-like nature, where the development progresses in a linear manner from one phase to another. The V-Model provides a more robust testing framework compared to traditional waterfall models.

* **Pros and Cons**

| **Pros** | **Cons** |
| --- | --- |
| This is a highly-disciplined model and Phases are completed one at a time | High risk and uncertainty |
| Works well for smaller projects where requirements are very well understood. | Not a good model for complex and object-oriented projects. No working software is produced until late during the life cycle |
| Simple and easy to understand and use | Not suitable for the projects where requirements are at a moderate to high risk of changing |
| Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process | Once an application is in the testing stage, it is difficult to go back and change a functionality. |

**1.2.4. Agile Model**

* **What is Agile Model?**

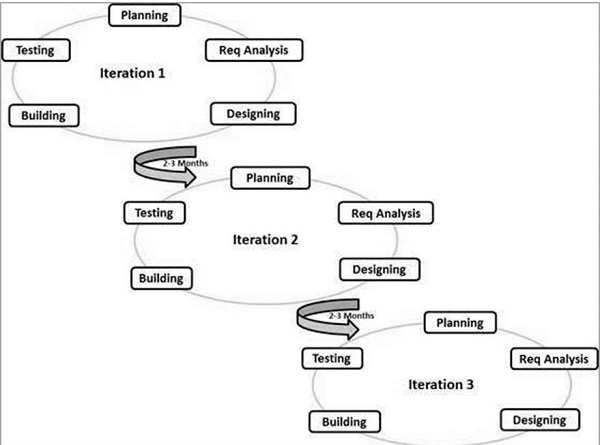
-It is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.

* **Characteristics**

-These are the characteristics of the Agile Model:

1. **Iterative and incremental development-** Agile follows an iterative and incremental development approach. Instead of attempting to define and deliver the entire software product upfront, it breaks the work into smaller iterations or sprints. Each iteration involves the development, testing, and delivery of a working product increment, providing value to stakeholders early on.
2. **Flexibility and adaptability-** Agile is designed to accommodate changing requirements and priorities. It recognizes that requirements can evolve over time, and it embraces changes throughout the development process. Agile teams are encouraged to be flexible, adaptive, and responsive to customer feedback, ensuring that the end product meets the evolving needs of the stakeholders.
3. **Customer collaboration and involvement-** Agile emphasizes close collaboration between the development team and the customer or product owner. Customers are actively involved in providing feedback, clarifying requirements, and prioritizing features. Regular meetings, such as sprint reviews and daily stand-ups, facilitate communication and collaboration, ensuring that the development aligns with customer expectations.
4. **Self-organizing cross-functional teams-** Agile promotes the formation of self-organizing and cross-functional teams. These teams consist of members with different skills and expertise, such as developers, testers, designers, and business analysts. They work collaboratively and take collective responsibility for delivering high-quality software increments.
5. **Continuous delivery and feedback-** Agile encourages the frequent delivery of working software increments. This enables stakeholders to see tangible progress and provide feedback early in the development process. Feedback is collected and incorporated into subsequent iterations, ensuring continuous improvement and customer satisfaction.
6. **Emphasis on quality and testing-** Agile places a strong emphasis on delivering a high-quality product. Testing activities are integrated throughout the development process, with the aim of identifying and resolving defects as early as possible. Agile teams employ various testing techniques, such as automated testing, unit testing, and continuous integration, to ensure the stability and reliability of the software.
7. **Transparent and visible progress-** Agile promotes transparency and visibility in project progress. Information radiators, such as task boards or burndown charts, are used to display the status of work, tasks, and progress. This enables all team members and stakeholders to have a clear understanding of the project status and promotes accountability and collaboration.
8. **Continuous improvement and learning-** Agile fosters a culture of continuous improvement and learning. At the end of each iteration, the team reflects on their performance, identifies areas for improvement, and implements changes in subsequent iterations. Agile methodologies, such as Scrum, often include dedicated retrospective meetings to facilitate this continuous learning process.

* **Agile Model Structure**

****

* **When do we use Agile Model?**

-Agile Model is typically used in the following scenarios:

1. **Rapidly changing requirements-** Agile is beneficial when the project requirements are likely to evolve or are not fully known at the beginning. The iterative nature of Agile allows for continuous feedback and adjustments to accommodate changing needs.
2. **Customer involvement-** Agile encourages close collaboration with customers or stakeholders throughout the development process. It allows for regular feedback and incorporates customer input to ensure that the end product meets their expectations.
3. **Small to medium-sized teams-** Agile methodologies, such as Scrum or Kanban, are often well-suited for small to medium-sized development teams. These methodologies promote self-organization and cross-functional collaboration, enabling teams to work efficiently and deliver incremental value.
4. **Complex projects-** Agile is effective for complex projects that require frequent testing and validation. By breaking down the work into smaller, manageable units (iterations or sprints), teams can identify and address potential issues early on, reducing risks and improving overall quality.
5. **Time-critical projects-** Agile methodologies allow for the delivery of working software in short iterations. This enables stakeholders to prioritize features and allows for earlier release of valuable functionality, even if the entire project is not yet complete.
6. **Innovation and experimentation-** Agile supports an environment of innovation and experimentation. It encourages teams to try new ideas, learn from failures, and make continuous improvements based on feedback and insights gained during the development process.

* **Pros and Cons**

| **Pros** | **Cons** |
| --- | --- |
| Promotes teamwork and cross training | Not suitable for handling complex dependencies |
| Functionality can be developed rapidly and demonstrated. | Depends heavily on customer interaction, so if customer is not clear, it can be driven in the wrong direction. |
| Enables concurrent development and delivery within an overall planned context. | There is a very high individual dependency, since there is minimum documentation generated. |
| Good model for environments that change steadily. | Transfer of technology to new team members may be quite challenging due to lack of documentation. |

**1.3. Selected SDLC Model**

* **Agile Model**

**Front-end Tech**

| **HTML** | **Pros:**   * Easy and simple to understand * Widely Supported * Lightweight and Efficient to deploy and load * Flexible and extensible   **Cons:**   * It only utilizes Static Pages * Limited functionality * Security Vulnerabilities |
| --- | --- |
| **ReactJS** | **Pros:**   * Virtual DOM * Performance * Scalability * SEO-friendly * Easy to learn   **Cons:**   * Learning Curve * Documentation * Can be difficult to debug |
| **VueJS** | **Pros:**   * Easy to learn * Flexible * Performance   **Cons:**   * Doesn’t have as many plugins as React * Not widely used as React * Documentation will be a bit difficult to find |
| **AngularJS** | **Pros:**   * Easy to learn * Utilizing the MVC architecture * Dependency Injection * Two-way data binding   **Cons:**   * Performance issues on larger projects * Complexity * Not as popular as React * Not as well-documented as React |

**Back-end Tech**

| **ExpressJS** | **Pros:**   * Lightweight framework * Flexible * Wide Community Support * Efficient   **Cons:**   * No full-stack framework support * No strict project structure feature * Manual implementation for security features |
| --- | --- |
| **Spring Boot** | **Pros:**   * Easy setup and configuration * Automatic dependency management * Includes health-monitoring, metrics and externalized configuration features * Seamless integration with Spring projects * Streamlined development process through embedded servers   **Cons:**   * High storage requirements for Spring Boot JAR files * Auto-configuration may break or infuriate specific project requirements * Slow start-up time |
| **Spring MVC** | **Pros:**   * Utilizes the Model-View-Controller architecture pattern * Seamless integration with Spring Security * Powerful data binding and validation capabilities   **Cons:**   * Steeper learning curve * Frequent updates that break old code * Limited out-of-the-box support for reactive programming |

**Database**

| **MySQL** | **Pros:**   * Easy to use * Utilizes Multi-threaded concurrency * Support for multiple users * Widely supported * Built-in network access   **Cons:**   * Hard to set-up * Slightly resource intensive than SQLite * Not as scalable as other RDBMS systems |
| --- | --- |
| **PostgreSQL** | **Pros:**   * Complies with the ACID standard for high degree of data integrity * Flexibility * Scalability   **Cons:**   * Complexity * Difficult documentation * Not as supported as MySQL |
| **SQLite** | **Pros:**   * Easy to set-up * Can be integrated to the project with 1 file * No reliance on an external RDBMS system running from a server   **Cons:**   * No built-in Security Measures * Support for multiple users * Only 2GB of database size * Slower performance when utilizing more complex applications |

**IDE**

| VS Code | Pros:   * Free * Easy to use * Wide plugin support * Visuals for Github modifications * Easy to use Terminal   Cons:   * No built-in easy Maven management without relying on plugins * Resource Intensive * No database management feature built in |
| --- | --- |
| **IntelliJ IDEA** | **Pros:**   * Fast maven importing * High Reliability * Support for high-quality plugins * Built-in Database Management System * Responsive project explorer * Interface is easy to use * Built-in Visuals for Github modifications * Easy to use dependency management system   **Cons:**   * Paid * More intensive on resources * Poor Customer Support |
| **STS** | **Pros:**   * Free * Light on resources * Open-source   **Cons:**   * Maven importing takes longer and very unreliable * Complicated User Interface * Extremely slow Project Explorer * No easy visuals for Github modifications * No built-in database management system |

**Selecting Technologies:**

1. **Front-end Tech:**

* JSP
* CSS
* JavaScript

1. **Back-end Tech:**

* Spring Boot 2.7.12

1. **Database:**

* MySQL Community Edition 8

1. **IDE:**

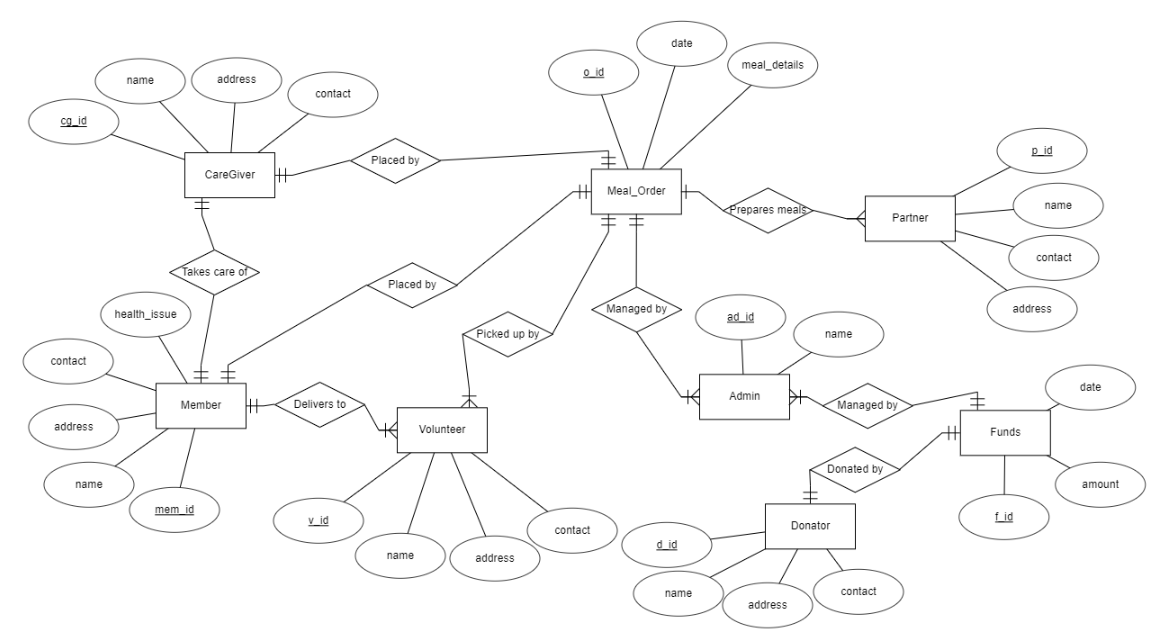
* Spring Tool Suite
* IntelliJ IDEA Ultimate
* Visual Studio Code

1. **Tools:**

* Draw.io
* ERDPlus
* Microsoft Word
* Microsoft Project
* Microsoft PowerPoint
* Figma

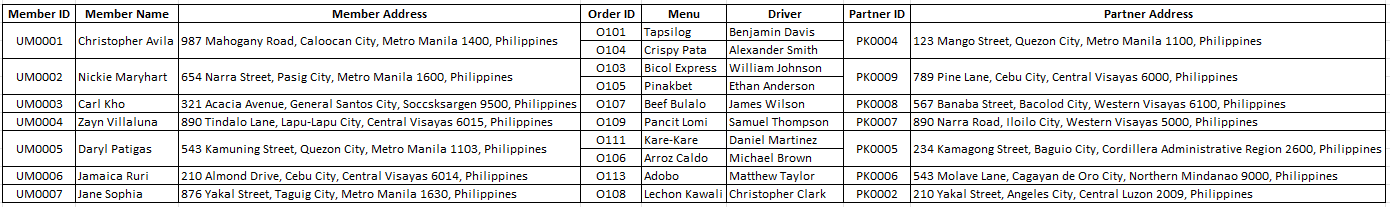
**1.4. Database Design: ERD Diagram and EERD (Physical Diagram)**

**EERD Diagram:**

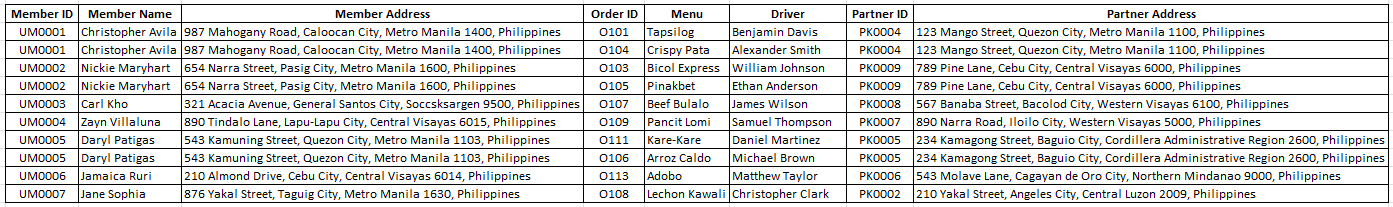
****

**Normalization:**

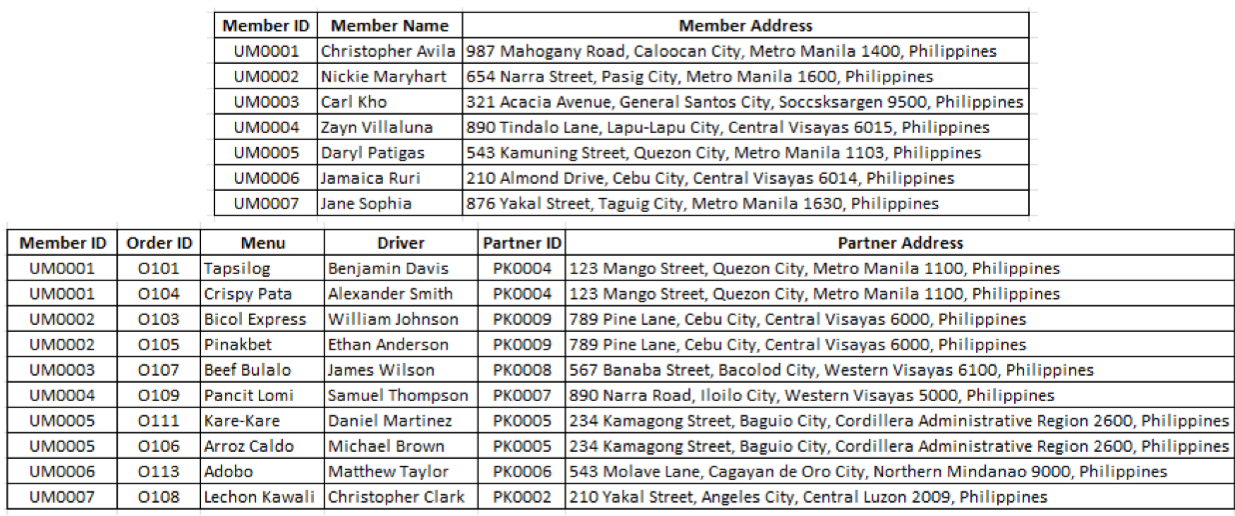
**UNF**

****

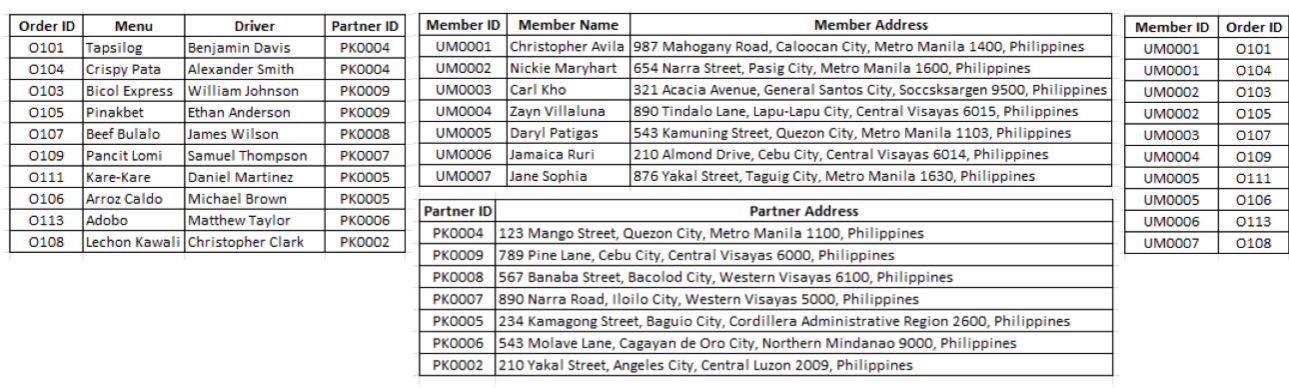
**1NF**

****

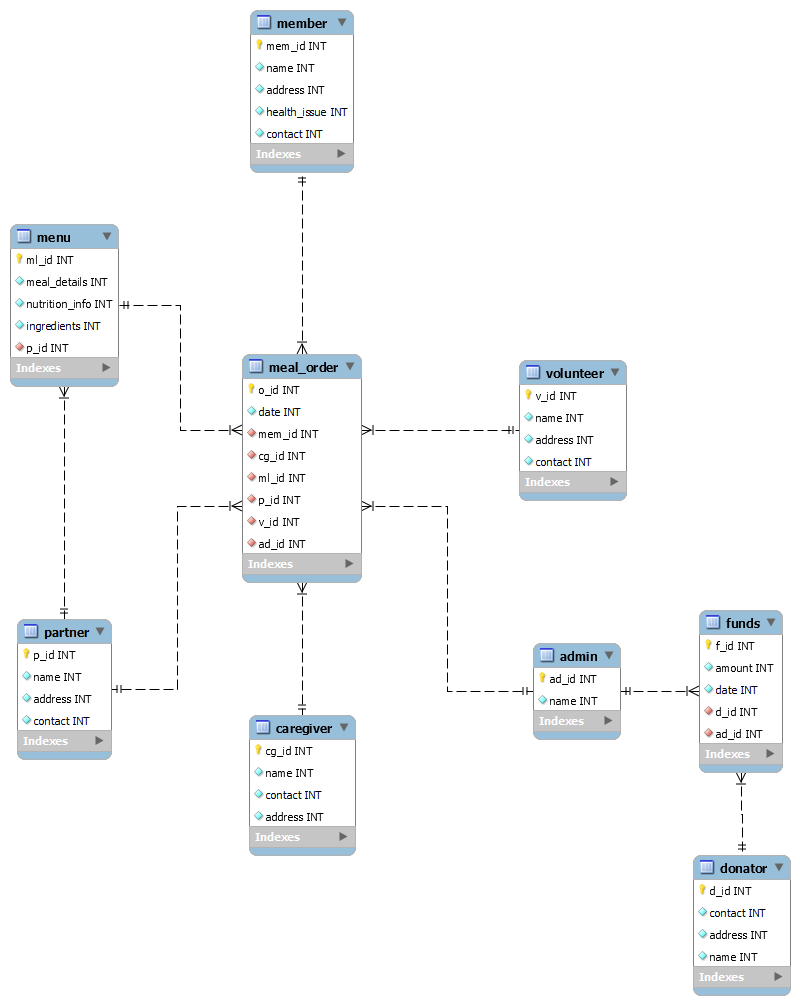
**2NF**

****

**3NF**

****

**Physical Diagram:**

****