**Chapter 3 Project Application Analysis Phase**

In previous chapters, the Secondary and Primary research was handled by discussing the project topic. This revealed the interest in the project and the importance of creating a related mobile application that gathers artisans and the general public on the same platform. To develop such an application, implement proper methodology before the development starts.   
The current chapter discusses the methodologies that are being used in the industry, revealing their cons and pros. Besides this suitable methodology will be chosen based on the given analysis.

**3.1 Object-Oriented Environment**

Objects are the main element of Object-Oriented Programming (OOP). An Object is an element that contains properties of procedures and data (attributes) that can have computations and save the local state.   
In OOP all of the actions are sending and receiving messages between objects and to process such a message uses their procedures called methods. (Wagner, 1990)

Class is used as a description for one or multiple alike objects. The example of a class in Kotlin language is represented in Figure 3.1, where Class Person have 3 attributes firstName, lastName and age in different types of data.

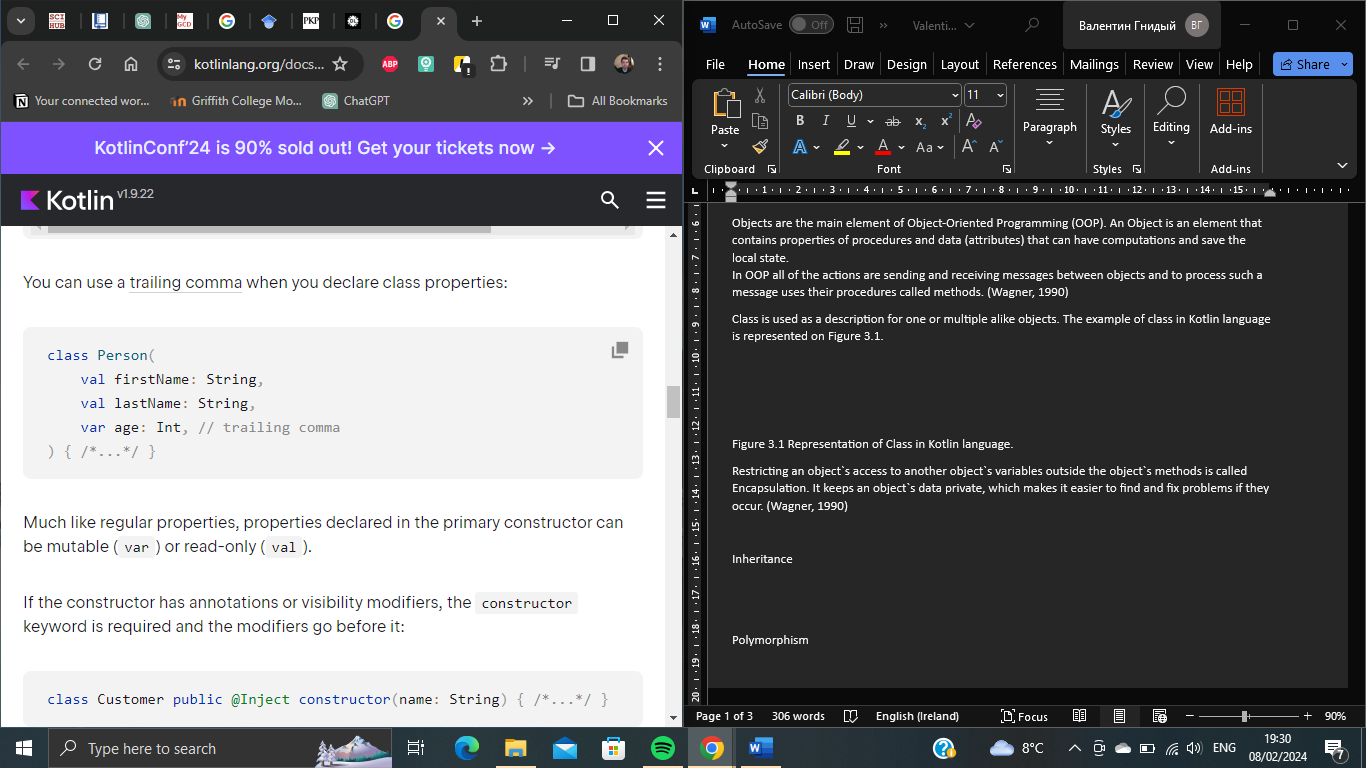


Figure 3.1 Representation of Class in Kotlin language. (kotlinlang.org)

The most recognisable concepts of OOP are:

**Encapsulation**: Restricting an object`s access to another object`s variables outside the object`s methods. It keeps an object`s data private, which makes it easier to find and fix problems if they occur. (Wagner, 1990)

**Inheritance** – the principle that allows the class to be created from another class or a hierarchy of classes that have common sets of attributes and methods. Also, it can be represented as a relationship between super classes and subclasses – elements which inherit data and behaviour from the superclass. Proper representation of inheritance is represented in Figure 3.2, where the Animal is a superclass with its attributes and methods, Lion and Dog are subclasses and inherit from superclass Animal attributes and methods, however have their such elements as well.

Advantages: Code reusability, avoidance of code duplication, increase of flexibility and extensibility of code, better code structure and management. (Gautam)

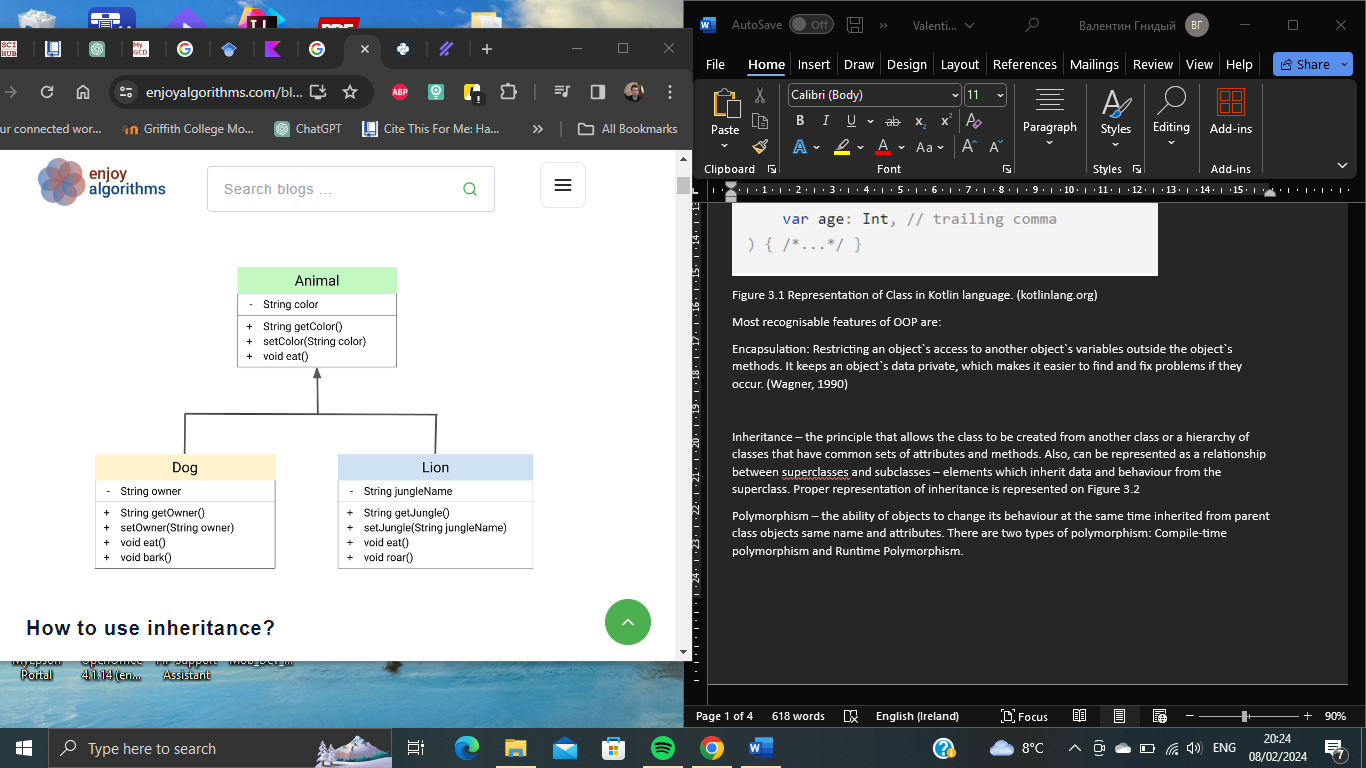


Figure 3.2 – Representation of Inheritance in OOP (Gautam)

**Polymorphism** – the ability of objects to change their behaviour at the same time inherited from parent class objects same name and attributes. There are two types of polymorphism: Compile-time polymorphism and Runtime Polymorphism.

**Compile-time polymorphism** – overloading of functions, where functions have the same names but are different in a way of accepting the argument. This makes methods of the same name perform differently.   
**Runtime polymorphism** when a program decides which version of a method to use while the program is running. This happens by changing the behaviour of the method in a subclass of a class. To make this work subclass must have a method with the same name as the one in the parent class.  
Advantages: code reusability. (Taylor, 2023)

**Data Abstraction** - display only essential information and hide the background details or implementation. There can be 2 types of abstraction: data abstraction – which displays only essential data and hides the unnecessary ones; and control abstraction – which shows only required information about implementation and hides unnecessary ones.   
Advantages: prevents users from writing low-level code; less code duplication and more reusability; makes application and program more secure; reduces complexity and redundancy of code, helps with readability. (Geekforgeeks.com)

**Advantages of OOP:**

* Modularity - chinking complex systems into smaller portions, makes it easier to maintain, create and comprehend.
* Flexibility and Scalability – OOP makes code easier to add and modify without making an impact on the entire database.
* Code Organisation – having a structured approach, increases collaboration and code readability.
* Code Maintenance – debugging and updating can be proceeded with only selected classes.
* Code Reusability – OOP is made to have code elements reused, which saves time and improves system reliability.
* Better Problem Solving – model real-world systems, developers can mimic real-world conditions and create solutions.

Disadvantages of OOP:

* Require knowledge – OOP is a complex conceptual model and requires specific knowledge for proper usage.
* More complex – having a big and difficult project can create difficulty in understanding and maintaining it properly.
* Performance overhead – despite procedure languages OOP ones are less focused on performance cost, due to additional abstraction and encapsulation.
* Management of dependencies – reuse of code and inheritance makes classes connect to, and depend on each other, which can create difficulties to implement changes.
* Inheritance overdue – not proper usage of inheritance can lead to over-complex hierarchy, maintenance and modification of such a code leads to additional difficulties. (Gaurav, 2023)

Due to inherent ability, the OOP approach can model the way artisans and creators communicate within the community. Organising code in Classes and Objects, modular and scalable structures that can be created that reflect the complexity of the handmade products marketplaces. Such an approach gives more in-code organisation, helps with maintenance and updates, and makes developed mobile applications more flexible.

**3.2 Development Methodologies**

Development methodologies play a huge role in software projects by supporting developers with planning, executing and managing the development process. Such methodologies whether traditional (Waterfall) or agile (Scrum), help with navigation through the complexities of the development process by defining objectives, recourse allocation, and creating proper communication. Using and choosing the methodology accordingly development teams can minimize risks, improve productivity, and deploy quality products that meet user's expectations and needs.

**3.2.1 Waterfall Model**

The waterfall model is one of the popular and widely used project management methods with a direct linear approach. Each stage where needs to be completed before stepping to the next one. Such an approach is highly suitable for projects where the objectives are strongly clarified from the start.

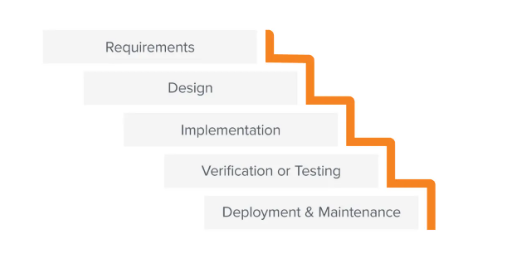


Figure 3.3 – Graphical Representation of Waterfall development methodologies. (business.adobe.com)

Waterfall Methodology was represented in 1970 by Winston W. Royce. It contains five stages of management: Requirements, Design, Implementation, Verification and Maintenance. Each of these stages needs to be deliverable before proceeding with followed one. The graphic representation of the current methodology is represented in Figure 3.3. Let's discuss further such methodology stages.

* **Requirements**. In this phase - clarify and verify the project`s requirements.
* **Design**. After making stakeholders' requirements clear developer team can start to design a proper solution.
* **Implementation**. This phase focuses on selecting one of the proposed designs and technologies to start implementing them.
* **Verification.** The created implementation needed to be checked and tested if it validated the requirements.
* **Maintenance.** Even after passing through the validation and verification, the system needs to be maintained and updated.

**Advantages and Disadvantages of Waterfall Methodology**

One of the main advantages of the Waterfall Methodology is that budget and time constraints are already specified before the start of creating the system itself. After verifying the goals of the project development team does not need frequent feedback or coordination from the client, except for reports from established milestones and deliverables.

The disadvantage of such a discussed methodology is the difficulty in addressing unexpected problems, challenges or unexpected changes in a way that validation and verification can proceed with the phase plan. One of the ways to minimise such issues is to gather as much information as possible to not go back on a phase.

Waterfall and Agile Methodologies

Agile and Waterfall project methods are suitable for different types of projects. The Agile methodology was created after Waterfall (more fixed and solid), as an alternative way to approach projects that need to be faster deployed and be more flexible to changes. (Hoory L., 2023)

**3.2.2. Increment/ Iterative Model**

Agile methodologies were represented in 1990s and quickly gained popularity with project managers. The Agile methodology is more focused on an iterative approach to reach and make decisions based on updated information faster.   
   
**Iterative development** – releasing the product in phases (iterations), which are shorter development and testing cycles related to one aspect (version) of the product. After the release and taking feedback from customers and stakeholders, the new phase starts with counting to count gathered information. Cycles are continuing until a working and optimised version of the product is launched.   
  
**Incremented development** – breaks the project into smaller segments called increments, where each of such pieces can be handled independently, based on the previous one. This way helps to implement improvements one by one.

The Incremented and Iterative models are strongly associated with each other in a way that they often work side by side in product development cycles.

What makes incremental and development processes popular for developers is an option to deliver value to customers efficiently and quickly. Integrating the Software Development Live Cycle in the same steps – plan, design, develop, test, release, and evaluate – but about each development iteration cycle, which is shorter and faster, gives the option to focus on the incrementation of the product.   
Also having feedback from customers and stakeholders gives advantage opportunity to improve the final product. The representation of iterative and incremental models is represented in Figure 3.4.

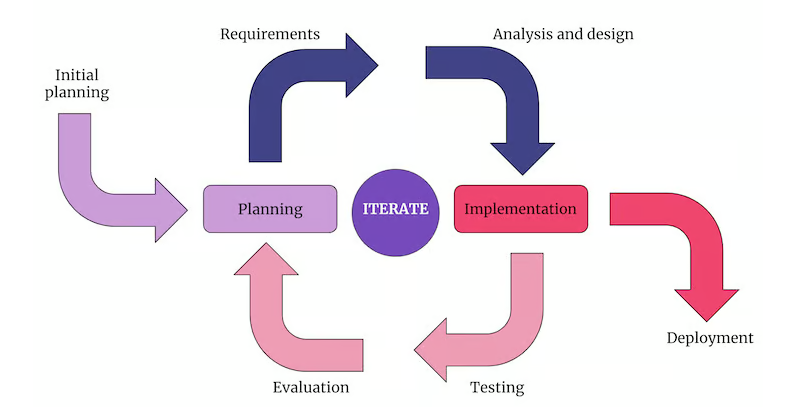


Figure 3.4 Graphical representation of Iterative/incremental development methodology and its cycles. (Godam S, 2023)

**Advantages and Disadvantages of iterative/incremental development methodology**

The **advantages** of using the represented development method are:

* Speed – having faster feedback and shorter development cycles, helps to identify user`s and stakeholder’s problems faster and have a better understanding of their needs.
* Flexibility – more adaptive to changes and implement new strategies related to feedback.
* Cost-efficiency – identifying development challenges in the process and better planning related to gathered data from past cycles, help with the use of recourses.
* Collaboration - gathering feedback after each cycle gives additional communication flow.
* Risk Reduction – having validation on every cycle helps to identify issues in the early stages.

The **disadvantages** can be considered as:

* Scope creep – customer requests for adding more features with a new development cycle can shift the initial scope of the project and also can put pressure related to having resources.
* Change management – changing of project score requires additional processes to manage, minimise, and measure challenges and move them into the product requirements can be difficult.
* Technical debt – incrementing new features and updates of functionality can lead to requests for new technologies in use.
* Minimal documentation – changing requirements and features leads to a lack of documentation.
* Unrealistic expectations – a developed product can be not fully ideal by not covering all user's and stakeholder`s desires and wishes related to the project. Without proper communication can create frustration and misunderstanding.

The discussed methodology helps to develop a high-quality product with proper recourse management. However, it is more related to the project and development team itself to identify what is better to conditions and constraints. (Gadam S., 2023)

**3.2.3. Boehm`s Spiral Model**

The Spiral model for software development represents that each cycle has the same steps as the Software Development Live Cycle. A representation of such a concept is in Figure 3.5

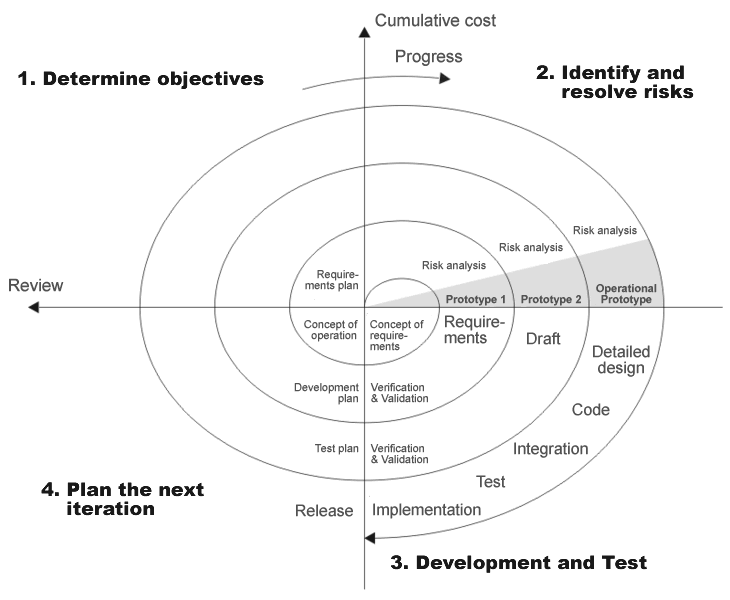


Figure 3.5 Boehm`s spiral life-cycle model. ([www.researchgate.com](http://www.researchgate.com))

Each such spiral begins with objectives related to the current project portion, choosing the way of implementation and clarifying the constraints. The next step is alternative objectives and risk evaluation. Followed step – stating the strategy related to the risks by using one of many techniques (prototyping, benchmarking, simulation, user questionnaire etc.) or a combination of such techniques.   
The next step is to determine other risks that were not being taken into consideration at the time start creating a plan related to the next level of prototyping and develop a more detailed prototype that can be used to resolve risk issues. If the represented prototype proves operationally viable and robust it will be considered for future evolution that will be represented in sequent steps of evolutionary prototypes, leading forward to the deployed product.

The Risk-driven approach allows for flexibility in adopting various software development strategies, chosen based on the magnitude of risks and the effectiveness of techniques. Additionally, the model ensures review and commitment at the end of each cycle involving stakeholders to ensure alignment with the approach for the sequential phases. (Boehm B. W.)

Advantages and Disadvantages of Spiral Development Model

The **advantages** of the spiral model are:

* Flexibility – additional functionality can be added even in later stages of development
* Easy cost estimation – having an established prototype gives an understanding of cost estimation.
* Risk management – minimising risks, by having evaluation in every stage of deployment.
* Faster than Waterfall – development is fast and features can be added systematically.
* Feedback – available to be taken at every stage.

The **disadvantages** are:

* Budget and time increasing – the risk of not meeting the deadline and budget.
* Constant risk evaluation – requires risk assessment expert.
* Complex documentation – intermediate phases increasing quantity of documentation.
* Not suitable for small projects – with small and limited timing. (Martin, M. (a) 2023)

The current development project has limitations in budget and timing which is why the Spiral Development methodology is not suited in a way of its complexity, level of planning and limitations of resources as well.

**3.2.4. Rapid Application Development (RAD)**

The Rapid Application Model is a software development approach centred around prototyping with minimum upfront planning. In this model, less effort is related to planning and more priority is given to the actual development of the tasks, aiming to create software in a short time frame.   
RAD uses followed phases that are represented in Figure 3.6 :

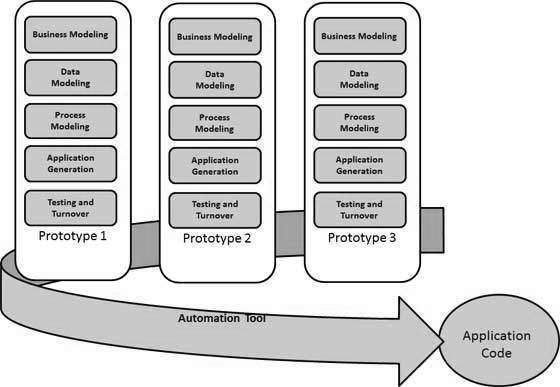


Figure 3.6 Agile Methodology development phases ([www.tutoarialpoint.com](http://www.tutoarialpoint.com))

* Business Modelling – designing of products, based on information flow
* Data Modelling – creating data objects that are essential for the business related to business modelling
* Process Modelling – achieving information flow that needed to be implemented to have business functioning.
* Application Generation – using software development tools to convert processes and data models into a prototype.
* Testing and Turnover – consider that developers are testing the prototype during every iteration the time for such phase is reduced.

The RAD methodology is mostly used in projects where:   
- having very short time constraints  
- requirements are well-known  
- users are highly involved in the development process  
- low technical risks  
- budget is enough to have professionals in development

Having RAD gives **advantages**:

* Flexible – easy to make implementations and changes;
* Risk Reduction – issues and constraints can be evaluated at any time;
* High-level abstractions – having an easy understanding process gives an advantage in coding and representation of products to stakeholders and users;
* Communication – functionalities are given to the client for feedback;
* Increase productivity – having a small group of highly qualified professionals with close and ease of communication highly increases the working flow.
* Time reduction – faster than the Spiral methodology.

Despite all such a methodology will not fit every project in a way of its **disadvantages**:

* Suitable only for short-term projects
* Not possible to use in large teams
* Can be used only in modular systems
* Teamwork and collaboration are essential
* Complex to manage
* The needs of users have to be met throughout its life cycle. (Martin M., (b) 2023)

The RAD technology become more widely used. Such methodology with prototyping is suited for developing a proposed mobile application as an artisan`s marketplace because it allows developers to create markups or prototypes of the application`s interface and functionality early in the development process. The iterative approach accelerates the timing and reduces the risks. Also, it helps to ensure that the final product meets the needs of artisans and customers in the local marketplace.

**3.2.5 Agile and Lean Software Development**

The Lean Software development concept focuses on optimizing efficiency and minimising losses during software development. Such an approach related to the Lean manufacturing movement in the 1980s, but nowadays considered as a part of Agile Software development Methodology.

The main concept of Lean development is represented as efficiency can be applied and waste can be handled at any level of the development process. These actions are taken at the individual level, inside departments, across interdepartmental operations, and through the entire organisation and in iterations between the organisation and its customers and its suppliers. The principles of Lean methodology that were represented in James Womack book in 1996 are graphically represented in Figure 3.7.

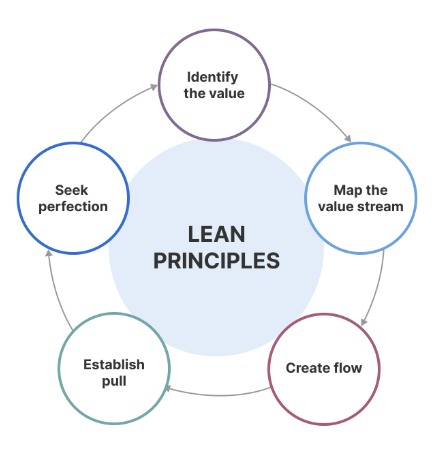


Figure 3.7 Principles of Lean methodology ([www.aha.io](http://www.aha.io) 2023)

In 2001 the Poppendiecks represented seven principles related to the Lean Software Development process:

* Eliminate Waste – cut from workflow anything (meetings, tasks, processes etc.) that doesn`t add value.
* Build quality – create quality checks in each stage by having frequent testing, incremental development, constant feedback, and automation.
* Amplify learning – knowledge must be shared between the development team via code reviews and learning through the development of itself and question assumption.
* Delay commitment as long as possible – having a goal of experimentation and learning in the process as much as possible before committing to the best decision. Software engineers deploy features and functions as late as possible with the intent to create solutions that do not need to be changed or updated soon.
* Deliver fast – Lunching product quickly to receive customer feedback fast and use it for improvement strategy. Fail fast – learn from the result.
* Respect people – one of the bases for a productive and collaborative atmosphere. Proactive communication, healthy conflicts and constant feedback are always supported by the Lean approach.
* Optimise the whole – examine the process by the team through all stages - make the Leam value stream as efficient as possible.

About differences with Agile, the Leam management approach is more focused on building better processes, while Agile`s – is on building better products. However, there is a lot in common between them. Both methods work in a seamless manner that is hard to tell them apart. That`s why people referred to them as Lean-Agile. (Lutkevich B., Silverthorne V. 2021)

There are some **advantages** to using the Lean software development method:

* Eliminates Waste – improves the productivity of development, which accelerates the development time and resources spent.
* Delivers the Projects on Time – allowing to complete more tasks gives time for implementing more features, that help to impress the stakeholders and clients.
* Empowers the Project and Development Team – helps developers be better at making decisions, creates a positive atmosphere and keeps the team motivated.
* Increase Quality – efficiency between development phases is improved, and regular scheduling assures the quality of the delivered project.
* Amplifies Knowledge – working in pairs and share of knowledge related to code elevates the development team in creating proper solutions for users' needs.

**Disadvantages and challenges** also can be faced using Lean methodology:

* Comprehensive documentation - using comprehensive and reliable information or precise figures in each stage of development required to be well documented and recorded.
* Flexibility – the growth of specifications can lead to new challenges and risks.
* Extent of Scalability – relying on team cohesion and individual contributions it is less scalable for the time consumption than traditional management approaches.

Lean Software Development is closely tied to Agile methodology, emphasising experience, understanding and rapid delivery of the product. The entire development process is created to achieve the final product at the lowest cost and reasonable timeline. Despite some challenges, most companies using such a methodology, find that benefits cover the drawbacks.

(Raghavan R., 2022)

3.2.6 Dynamic System Development Methodology (DSDM)

DSDM is an agile software development methodology that was released in 1994 and most of the time was used as a part of RAD. Later, was reconsidered as a separate approach in project management.

DSDM involves a diverse set of activities that are conducted through a project`s lifecycle, providing a solid foundation that enables an effective project lifecycle. The chosen methodology adopts an iterative and incremental approach to systems development, allowing active engagement of users and customers in development processes, that are based on Agile principles. (Janse B. 2022)

The DSDM development methodology's eight principles are represented in Figure 3.8 and are based on the conception that most issues in the working process are related to human mistakes, that why it is vital for to people work together.

* Focus on business needs – the team must understand project needs.
* Deliver on time – every iteration is a timebox, despite of Scrum spin. The method can have an incremental approach as well as iterative to deliver software. Having deadlines helps to manage changes in requirements.
* Collaborate – the team need to be working as one unit and collaborate to encourage understanding and high performance.
* Never compromise quality – high standards are established from the start. Testing starts from the early stages.
* Build incrementally – with each increment the team verifies priorities and project visibility.
* Develop iteratively – changes are constant and an incremental approach helps to meet stakeholders’ satisfaction.
* Communicate continuously – meetings, visual communication (modelling), releases of prototypes and workshop sessions, representation of prototypes.
* Demonstrate control – ensure that the project is visible to everyone.

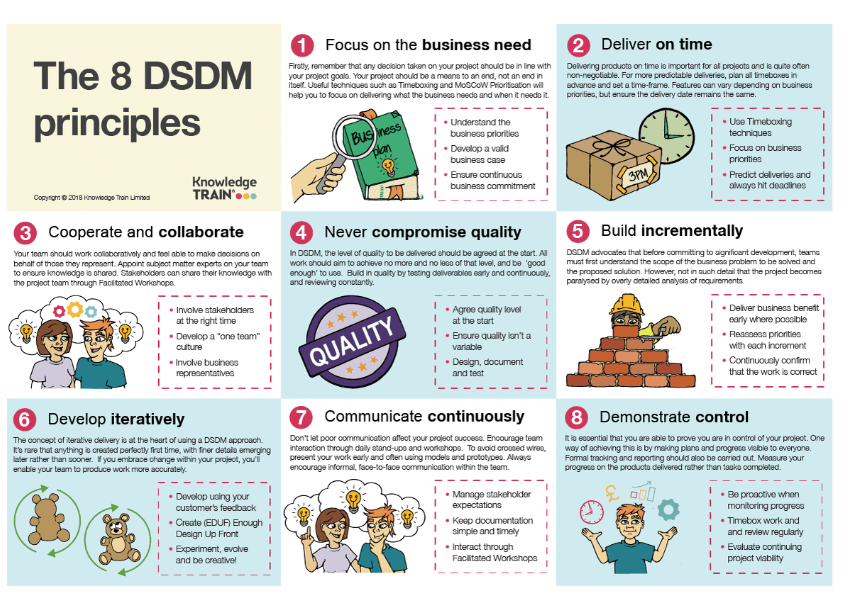


Figure 3.8: Principles of DSDM (Buehrig S, 2023)

DSDM principles represent the base states of the Agile manifest, such as a focus on iterative delivery, effective communication, collaboration and continuous delivery. Process model and specific team roles are some of own characteristics of the discussed methodology.

(Buehrig S, 2023)

3.3 Object-Oriented Analysis with UML Diagrams

Object – Oriented Analysis (OOA) is the process of clarifying software requirements and specifications in terms of the system object model, that represents iterating objects.

The main emphasis of the object-oriented approach, where requirements are based on object communications with data and functions. That is modelled like real-world objects that the system is associated with.

The main tasks in object-oriented analysis are:

* Identify objects.
* Creating an Object Model Diagram.
* Clarifying attributes and methods of the Objects.
* Identify object iterations.

Creating diagrams for object representations is used by the United Modelling Language (UML).

UML is a graphical language that gives a standard approach to creating graphical representations of a software system that helps visualise, specify, construct and document elements of an object-oriented system.

UML was developed in the 1990s as a result of combining several techniques, such as Object Orientated Agilisys design (OOAD) by Grady Booch, Object Modelling technique by James Rumbaugh, and Object Orientated Software Engineering by Ivar Jacobson. UML was trying to standardise the semantics of models, syntactic notations and diagrams of OOAD.

(Larman C, 2005)

3.3.1 Use Case Diagram

UML Behavioural Diagram helps to visualise and build the dynamic aspects of a system. Use case diagram is categorised as behavioural one. Such a diagram is used to specify the contest of the system, capture the requirements of a system, and validate system architecture.

Elements of the Use case Diagram are:

* Actor:

Something that interacts with the use cases and can trigger them. Playing as role of the user, however, can be as well another system outside of a current. All system input and output are related to the actor.

Figure 3.10 (a): Actor role representation in Use Case Diagram

* Use Case:

Have a role of System Function. An actor has to be linked to a Use case, however, not every use case needs to be linked to the actor.

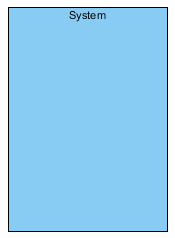
Figure 3.10(b): Use case representation in Use Case Diagram

* Communication Link:

Represents the connection between an actor and a use case, and also shows the connection between use cases



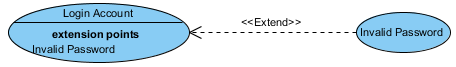
Figure 3.10(c): Communication representation in Use Case Diagram

* System Boundary:

Abstractive representation of the modelled system. For big and complex systems each module can be represented as a separate boundary. All information flow and use cases are proceeded inside the boundary. Actors are the only elements that are located outside the boundary.

Figure 3.10(d): System Boundary representation in Use Case Diagram.

Relationships in the Use Case Diagrams play a role in showing the dependencies between use cases, and based on them will be made important decisions in software development. There are two main relationships **include** and **extends**. <<Extend>> Is directed to a relationship that states as optional or possible behaviour that is not necessary to be involved in the iteration. <<Include>> - is used to represent what is included in the use case on a general basis and executes by itself without the involvement of the actor. (Amber S. W., 2010)

  
Figure 3.10 (e, f) Examples of Include and Extend communication of use cases.

Appendix A4 represents the Use Case Diagram of the Developed project. The chosen development solution is the Mid–Range, where have 3 Actors: Artisan, Customer and third-party Digital Wallet.

The **Customer** starts to iterate with the developed system by *Registration* and *Login* after being able to look for artisans in their products by *Viewing Google Mars with Artisans in Vicinity* and *Selecting Specific Artisan and Product(s)* be able to select the desired product and *Send Product Request Form* to purchase, receiving *Notification of Product Request Acceptance/Rejection* with providing *Invoice for Product Payment* if the Artisan approved of availability of purchase, that purchase can be made by *Providing Payment for Product* via *Created digital wallet*. The customer is also available to leave *Feedback & Rating for Product Obtained,* which will be sent to the craft maker.

The **Artisan** interacts with the system by *Logging in* and *Creating/Updating a Business Profile* with the representation of the profile on Google Maps. Within time *Receive Product Request Forms* from the customer and clarify whether to Accept or Reject They will be sent back to the Customer as a notification and after that will receive an *Invoice for Product Payment* to present the purchased product to the *Customer*.

The **Third-Party Wallet** is an Actor thatisinvolved in the functionalities of payment in a system. This is related in such: *Creating Digital Wallets*, *Funding Digital Wallets* and *Providing/Receive* *Payment* for *Products*.

The **Third-Party Chat** is an Actor that supports the designed system with the functionalities of *Chat* and *Forum* that help to create proper communication between Artisan and Customer.

**3.3.2 Sequence Diagram**

UML Sequence Diagram is an interaction diagram that shows how operations are executed in a system. This diagram represents the interaction between objects in the partnership settings. It is an important tool for realising how the application works and how it processes information. It displays communication between objects in running applications by showing messages that pass between them. An example of the discussed diagram is represented in Figure 3.11

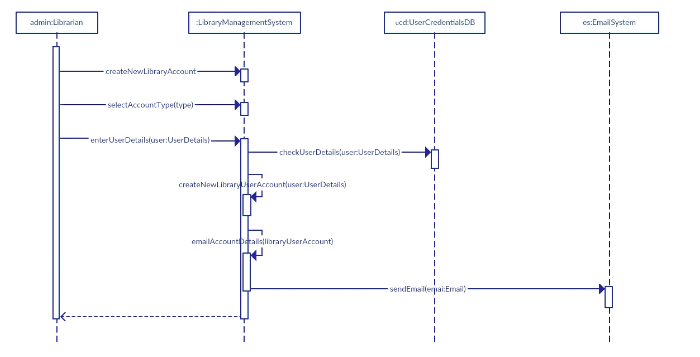


Figure 3.11: Example of Sequence Diagram (Athuraliya A., 2022)

Elements of Sequence Diagrams are:

* **Actors**

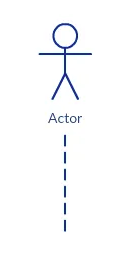
Familiar to Use Case Diagram. An actor can be a representation of a person or another system that interacts and sends messages to the modelled system. It is always outside of the system, although there should be no direct connections between actors on a diagram. The graphical representation of an actor is represented in Figure 3.12(a)

Figure 3.12 (a): Representation of an Actor in Sequence Diagram. (Athuraliya A., 2022)

* **Messages**

Messages are represented as horizontal lines with arrowheads and labels. They are placed from one object to another and go through the system.

Figure 3.12 (b): Representation of message in Sequence Diagram (Athuraliya A., 2022)

* Lifeline

Each Object obtains its dotted vertical line, which represents the lifeline related to iterations within objects. An example of a lifeline is shown in Figure 3.12(c)

Figure 3.12 (c): Representation of Object Lifetime in Sequence Diagram

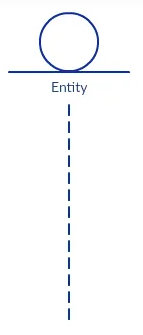
* Focus of Control or Execution Occurrence (Athuraliya A., 2022)

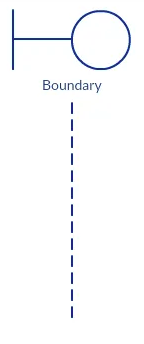
Thin bars on the lifeline of related objects – are a graphical representation of the Focus of Control at the Sequence Diagram. Shows a time when the message was received by the Object and when was sent.

Figure 3.12 (d): Focus of Control or Execution Occurrence Representation (Athuraliya A., 2022)

* Object

Objects are placed on the top of the diagram alongside with Actors and are responsible for sending and receiving messages. The most common types of objects are:

**Entity Class** – stores data and behaviour. The entities continue to exist within a system until the end of the system's life.

 **Boundary Class** – supports with connection between Actors and Use Cases, as being an interface of the system.

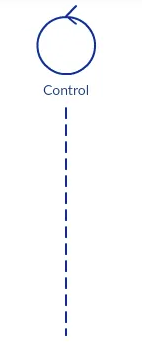
**Control Class** – logic control of the system. Contains the algorithm related to the Use Case. Messages that come to the Control Class can be processed and redirected, but will not be stored.

Figure 3.12 (e-g): Representation of a (e) Entity class, (f) Boundary Class, (g) Control Class in Sequence Diagram. (Athuraliya A., 2022)

The most typical scenario in Sequence Diagrams is where an Actor Class interacts with a Boundary Class. The message that was received by the interface was directed to a Control Class from where the processed information is stored in a related Entity Class and represented to an Actor Class by a Boundary Class. Such a system works by rules of direct flow, that an Actor Class can interact only with a Boundary Class. A Boundary contacts with an Actor Class and a Control Class. Control can be connected with another Control Class, Boundary Class and Entity Class. Entity Class – can communicate only with Control Class and can`t communicate with another Entity Class. (Athuraliya A., 2022)

The Sequence Diagram of the current project is represented in Appendix A5 and contains such elements as:

**Actors:**

* User
  + Messages:
    - Request Login
    - Request View Artisans in Vicinity
    - Select Artisan from the list
    - Place Request for Selected Artisan`s Item within Category
    - Request Create/ Fund Digital Wallet
    - Provide Feedback and Rating for Artisan's Product and Service Obtained
* Artisan
  + Messages:
    - Request Login
    - Request Create Profile
    - Send Notification for Customer`s Request
    - Request Create / Fund Digital Wallet
    - Request for Create Chat / Forum

**Entities**:

* Product Category
* Artisan
* Artsan`s Profile With Categories of Products
* Artisan`s Item Detailed Within a Category
* Customer
* Notification of Customer Request Acceptance/ Rejection
* Invoice for Payment of Accepted Customer`s Request
* Customer`s Payment for Invoice
* Notification of Customer`s Payment Provided
* Customer`s Feedback & Ratings for Artisan`s Item Obtained
* Third-Party Digital Wallet SW Boundary
* Third-Party Chat/Forum SW Boundary

3.3.3 Object Class Diagram

Object Class diagrams reveal the relationships between classes, iterations, instances and interfaces of the class. The class itself is represented as a rectangular element with 3 sections: a class`s current name on a top section, a middle section left for attributes (data of class), and behaviour (actions, message modification and functionality and is shown in a bottom layer. Key elements – Primary Keys and Foreign Keys need to identify the Object itself and help with understanding communications between them.

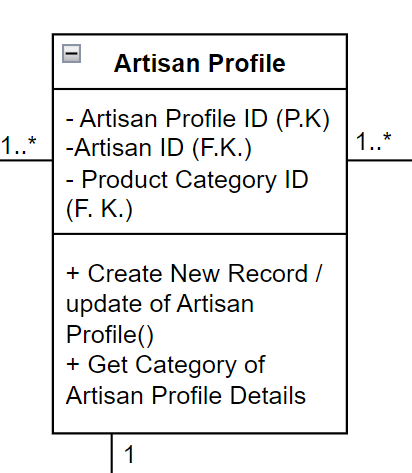


Figure 3.13: Graphical representation of an Object in Object Class Diagram. (Appendix A6)

Communication between classes or within itself is represented as solid lines also known as *relationships*. Symbols shown in Figure 3.13 placed on a line show the cordiality of a relationship. (Nisansala, C. 2021)

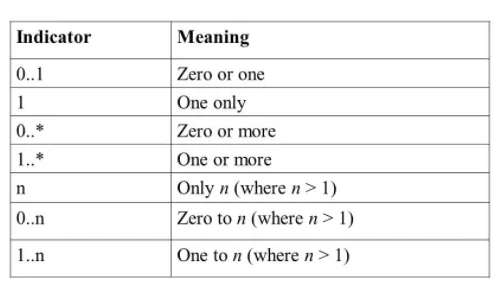


Figure 3.13: Cordiality in the Class Diagram

The Proposed Project`s Object Class Diagram is represented in Appendix A6, which contains classes of:

***Artisan***

**Attributes**:

* Artisan ID (P.K.)
* Artisan Name
* Artisan Mobile Number
* Business address
* Email Address
* Password

**Methods**:

* varifyLoginDetails()
* getArtisanDetails()

***Artisan Profile***

**Attributes:**

* Artisan Profile ID (P.K.)
* Artisan ID (F.K.)
* Product Category (F.K.)

**Methods:**

* createNewRecord() / updateArtisanProfile()
* getCategoryOfArtisanProfileDetails()

***Product Category:***

**Attributes:**

* Product Category ID (P.K.)
* Category Name

**Methods:**

* getProductCategory()
* createNewRecordOfProductCegory()

***Customer Digital Wallet***

**Attributes:**

* Wallet ID (P.K.)
* Customer ID (F.K.)
* Balance

**Methods:**

* createNewRecordOfCustomer`sDigitalWallet ()
* updateDigitalWalletBalance()

***Customer***

**Attributes:**

* Customer ID (P.K.)
* Customer  Name
* Customer Mobile Number
* Email Address
* Password

**Methods:**

* Create New Record of Customer ()
* Varify Login Details()
* Registration of Customer

***Notification of Acceptance / Rejection of Customer Request***

**Attributes:**

* Notification ID (P.K.)
* Customer Request  ID (F.K.)
* Customer ID (F.K.)
* Artisan ID (F.K.)
* Artisan Product (F.K.)
* Acceptance (Y/N)
* Notification Text
* Notification Date(System Date)
* Notification Time (System Time)

**Methods:**

* createNewRecordOfNotificationforAccptance/RejectionOfCustomer`sRequestForProduct()

***Artisan Product Item Within Category***

**Attributes:**

* Artisan Product Item ID (P.K.)
* Artisan ID (F.K.)
* Product Category ID (F. K.)
* Product Quantity on hand#
* Product Item Name
* Product Item Description
* Product Item Price
* Product Item Discounted Price
* Product Picture 1
* Product Picture 2

**Methods:**

* createNewRecord/UpdateOfArtisan`sProductItem()
* getDetailsOfArtisan`sProductItem()

***Customer Request for Artisan`s Product Item***

**Attributes:**

* Customer Request  ID (P.K.)
* Customer ID (F.K.)
* Artisan ID (F.K.)
* Artisan Product (F.K.)
* Quantity Requested
* Request Total

**Methods:**

* CreateNewRecordOfCustomer`sRequestForAproduct()

***Customer Payment for Invoice of Accepted Customer Request***

**Attributes:**

* Customer Payment ID (P.K.)
* Invoice ID (F.K.)
* Notification ID (F.K.)
* Customer Request ID (F.K.)
* Customer ID (F.K.)
* Artisan ID (F.K.)
* Artisan Product (F.K.)
* Payment amount Total
* Notification Text
* Payment Date (System Date)
* Payment Time (System Time)
* Artisan Bank Account Details

**Method:**

* CreateNewRecordOfCustomerPaymentForInvoiceForAcceptedCustomerRequestOfProduct()

***Invoice for Payment of Accepted Customer Request***

**Attributes:**

* Invoice ID (P.K.)
* Notification ID (F.K.)
* Customer Request ID (F.K.)
* Customer ID (F.K.)
* Artisan ID (F.K.)
* Artisan Product (F.K.)
* Invoice amount Total
* Invoice Date (System Date)
* Invoice Time (System Time)
* Artisan Bank Account Details

**Methods:**

* CreateNewRecordOfPaymentOfAcceptedCustomer`sRequestForProduct()

***Notification for Payment Provided of Accepted Customer Request***

**Attributes:**

* Notification for Payment ID (P.K.)
* Invoice ID (F.K.)
* Notification ID (F.K.)
* Customer Request ID (F.K.)
* Customer ID (F.K.)
* Artisan ID (F.K.)
* Artisan Product (F.K.)
* Payment amount Total
* Notification Text
* Notification Date (System Date)
* Notification Time (System Time)
* Artisan Bank Account Details

**Methods:**

* CreateNewRecordOfPaymentProvidedForInvoiceForAcceptedCustomerRequestOfProduct()

**3.4 Alternate Design Solutions**

Creating alternative solutions is a way to give a choice to the stakeholders in a way to clarify what is more realistic to be taken as a basis for the development. The quantity of solutions left to the project manager can be from only one (as no alternative) to as much as possible. However, the common practice is to give 3 alternatives:

* Low-End Alternative – low-cost solution with minimum functionality, that makes the system work.
* Mid-Range Alternative – a comprehensive solution between Low-End and High-End.
* High-End Alternative – a high-cost solution that includes extra features that users might desire. (Kendal K.E., Kendal J.E., 1999)

The alternative solutions for the Developed project are represented in Appendix A7:

**Low–end Alternative:**

The current system will not be possible without the ***functionalities*** of:

* Login
* Registration
* Created/Update Business Profile
* View Google Maps with Artisans in Vicinity
* Select Specific Artisan and Products

**Mid-Range Alternative**:

Includes necessary features to have the system work and includes the functionalities that users looking to have in an Artisan Marketplace. Such a case includes all functionalities of *Low-End Alternative*, with the addition of:

* Send/Receive Product Request Form
* Notification of Product request Acceptance/Rejection
* Invoice for Product Payment
* Create a Digital Wallet
* Fund Digital Wallet
* Provide/Receive Payment for Product

**High-End Alternative:**

The High-End Alternative includes functionalities of Low-End and Mid-End Alternatives and also has:

* Feedback & Rating for Product Obtained
* Chat
* Forum

Related to the timing and budget constraints, the Alternative Solution that is going to be a plan for the development of mobile applications is the Mid-Range one. It includes the functionalities that users are most interested in based on the Initial Survey and Structured Interview, within the availability of recourses of the developed team.

3.5 Chapter Summary

This chapter describes Object Oriented environment and its specifications, compares different Software Development Methodologies, represents Unified Modelling Language diagrams (Use Case, Sequence and Object Class diagrams) as a great tool for analysis of the developed project and shows Alternative Design Solutions.

Related to the described information in the chapter, the proposed application will be using the Object Orientated system, with Rapid Application Development with Prototyping methodology. Object Oriented Programming makes the system modular, flexible and easy to build and maintain. The RAD with Prototyping is useful to evaluate risks and helps to gather feedback in a short time.

Alternative design evaluation reveals that the most realistic way to build a prototype is the Mid-Range Solution which will give to user the option to have not only basic but essential functionality of a platform such as Local Artisan Marketplace.

The next chapter will describe the Interface and Database from the Coding and Design process of the Proposed Application by storyboards, wireframes and screenshots.

**REFERENCES:**

*Abstraction in C++* (2023) *GeeksforGeeks*. (Online) Available at: https://www.geeksforgeeks.org/abstraction-in-cpp/ (Accessed: 05 December 2023).

*Agile vs. lean: Is there a difference?: AHA! software* (2023) *Agile vs. Lean: Is There a Difference? | Aha! software*. (Online) Available at: https://www.aha.io/roadmapping/guide/agile/agile-vs-lean (Accessed: 13 January 2024).

Ambler, S.W. (2010) *The elements of UML 2.0 style*. Cambridge: Cambridge University Press.

Athuraliya, A. (2022) *Sequence diagram tutorial – complete guide with examples*, *Creately*. (Online) Available at: https://creately.com/guides/sequence-diagram-tutorial/ (Accessed: 16 January 2024).

Boehm, B.W. *A spiral model of software development and enhancement*. (Online) Available at: https://www.cse.msu.edu/~cse435/Homework/HW3/boehm.pdf (Accessed: 17 December 2023).

*Boehm’s spiral life-cycle. the spiral model attempts to address two ...* (Online) Available at: https://www.researchgate.net/figure/Boehms-spiral-life-cycle-The-spiral-model-attempts-to-address-two-main-difficulties-of\_fig4\_220631422 (Accessed: 17 December 2023).

Buehring, S. (2023) *DSDM principles: 8 principles of DSDM (Dynamic Systems Development Method): Infographic*, *Knowledge Train*. (Online) Available at: https://www.knowledgetrain.co.uk/agile/agile-project-management/agile-project-management-course/dsdm-principles (Accessed: 17 January 2024).

*Classes: Kotlin*, *Kotlin Help*. (n.d.) (Online) Available at: https://kotlinlang.org/docs/classes.html#constructors (Accessed: 02 December 2023).

Gaurav, S. (2023) *Advantages and disadvantages of OOP*, *Scaler Topics*. (Online) Available at: https://www.scaler.com/topics/oops-advantages/ (Accessed: 05 December 2023).

Janse, B. (2022) *Dynamic Systems Development Method (DSDM) explained*, *Toolshero*. (Online) Available at: https://www.toolshero.com/information-technology/dynamic-systems-development-method/ (Accessed: 16 January 2024).

Kendall, K.E. and Kendall, J.E. (1999) *Systems analysis and design*. Upper Saddle River, NJ: Prentice Hall.

Larman, C. (2005) *Applying UML and patterns: An introduction to object-oriented analysis and design and the unified process*. Upper Saddle River (New Jersey): Prentice-Hall.

Lutkevich, B. and Silverthorne, V. (2021) *What is Lean Software Development? definition and principles*, *Software Quality*. (Online) Available at: https://www.techtarget.com/searchsoftwarequality/definition/lean-programming (Accessed: 15 January 2024).

Martin, M. (a) (2023) *Spiral model: When to use? advantages and disadvantages*, *Guru99*. (Online) Available at: https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html (Accessed: 04 January 2023).

Martin, M. (b) (2023) *What is Rad Model? phases, advantages and disadvantages*, *Guru99*. (Online) Available at: https://www.guru99.com/what-is-rad-rapid-software-development-model-advantages-disadvantages.html (Accessed: 11 January 2024).

Nisansala, C. (2021) *UML class diagrams*, *Medium*. (Online) Available at: https://chathushinisansala96.medium.com/uml-class-diagrams-db86533de854 (Accessed: 19 January 2024).

Raghavan, R. (2022) *What is Lean Software Development ? - ACODEZ IT solutions*, *Acodez*. (Online) Available at: https://acodez.in/lean-software-development/ (Accessed: 13 February 2024).

*SDLC - Rad Model* (no date) *Tutorialspoint*. (Online) Available at: https://www.tutorialspoint.com/sdlc/sdlc\_rad\_model.htm (Accessed: 11 January 2024).

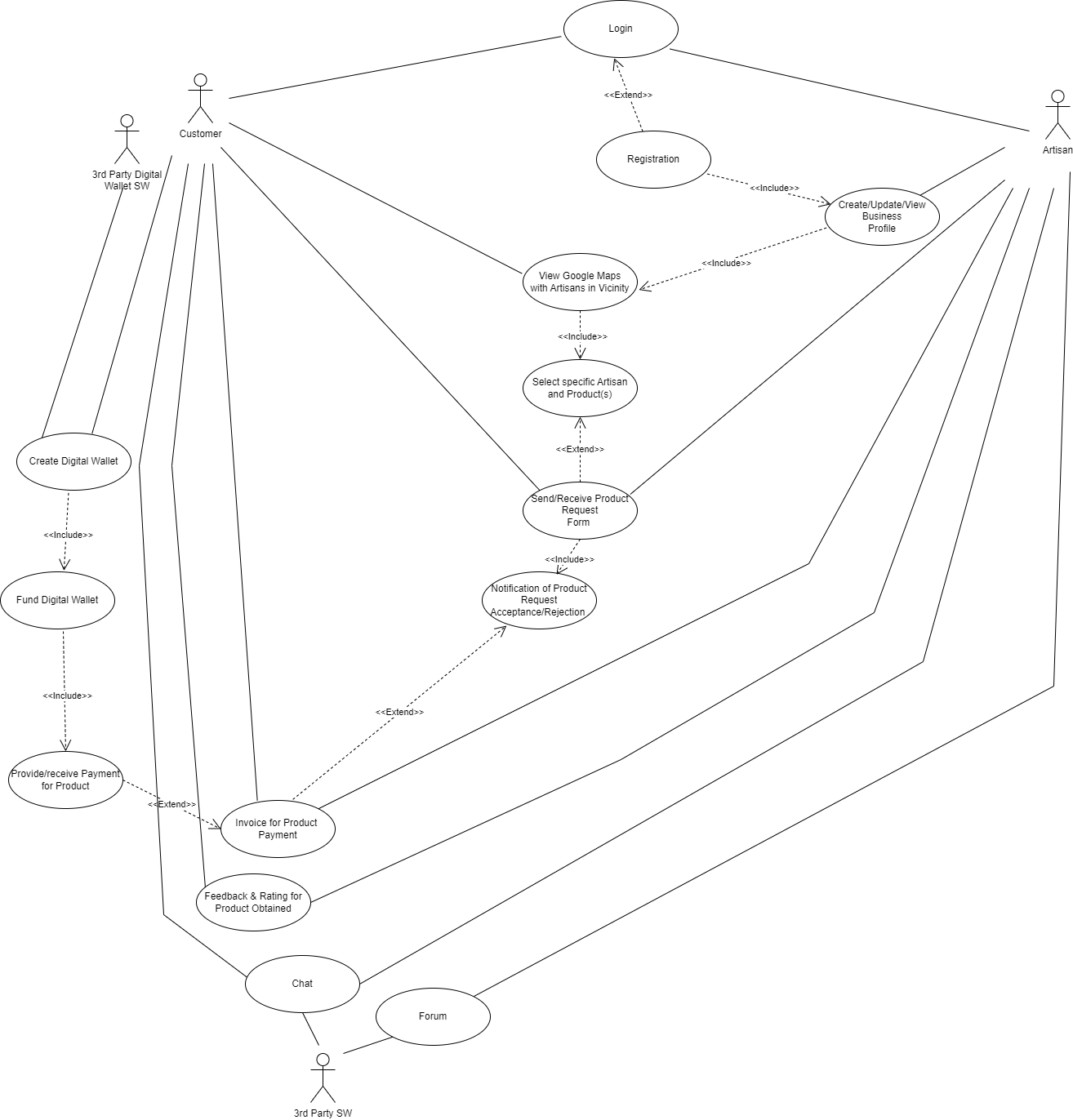
Taylor, T. (2023) *Understanding the role of polymorphism in OOP: TechTarget*, *App Architecture*. (Online) Available at: https://www.techtarget.com/searchapparchitecture/tip/Understanding-the-role-of-polymorphism-in-OOP (Accessed: 03 December 2023).

Hoory, L. (2023) *What is waterfall methodology? here’s how it can help your project management strategy*, *Forbes*. (Online) Available at: https://www.forbes.com/advisor/business/what-is-waterfall-methodology/ (Accessed: 06 January 2024).

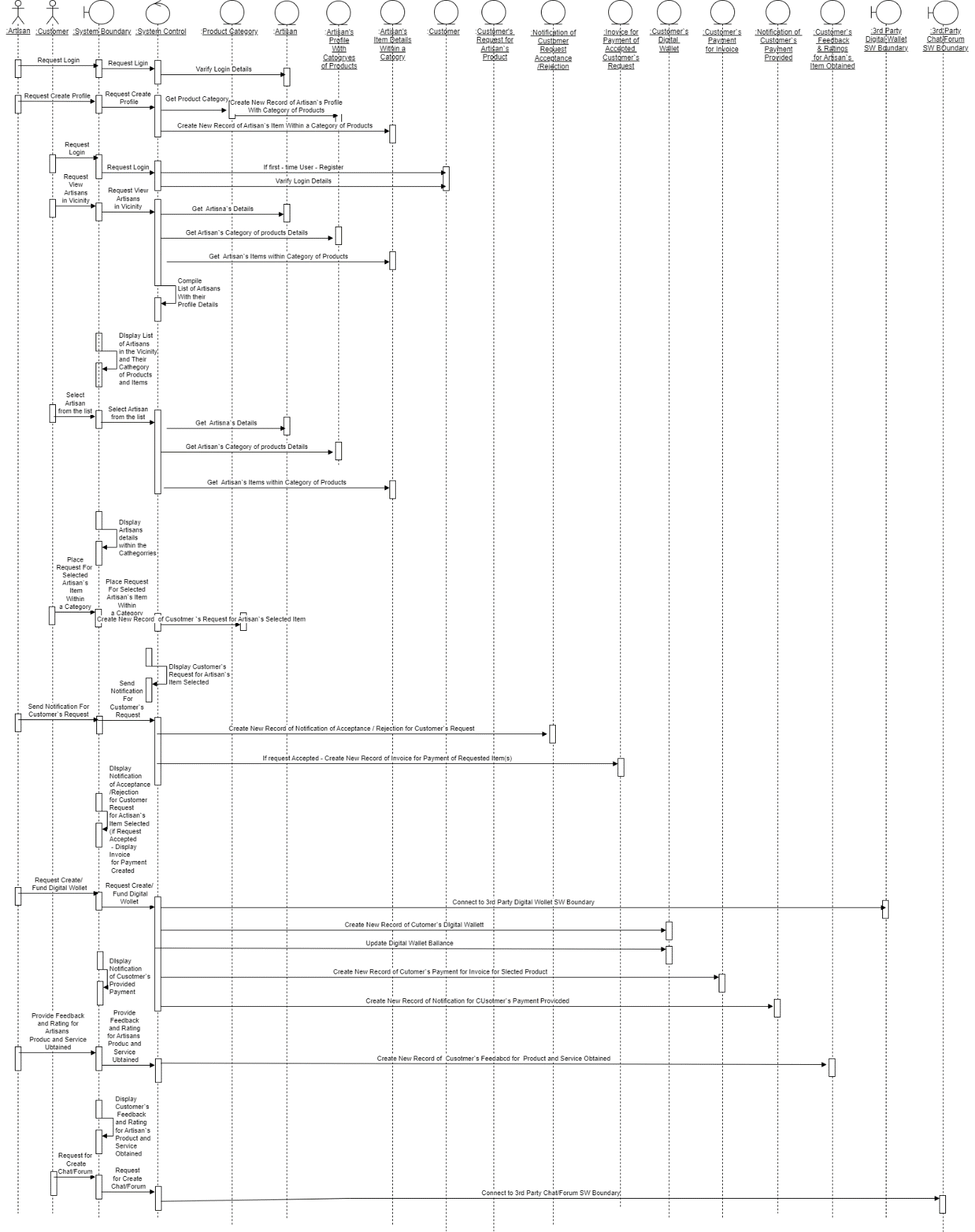
*Waterfall methodology: Project management | Adobe Workfront*. (Online) Available at: https://business.adobe.com/blog/basics/waterfall (Accessed: 05 January 2024).

Wegner, P. (1990) ‘Concepts and paradigms of object-oriented programming’, *ACM SIGPLAN OOPS Messenger*, 1(1), pp. 7–87. doi:10.1145/382192.383004.

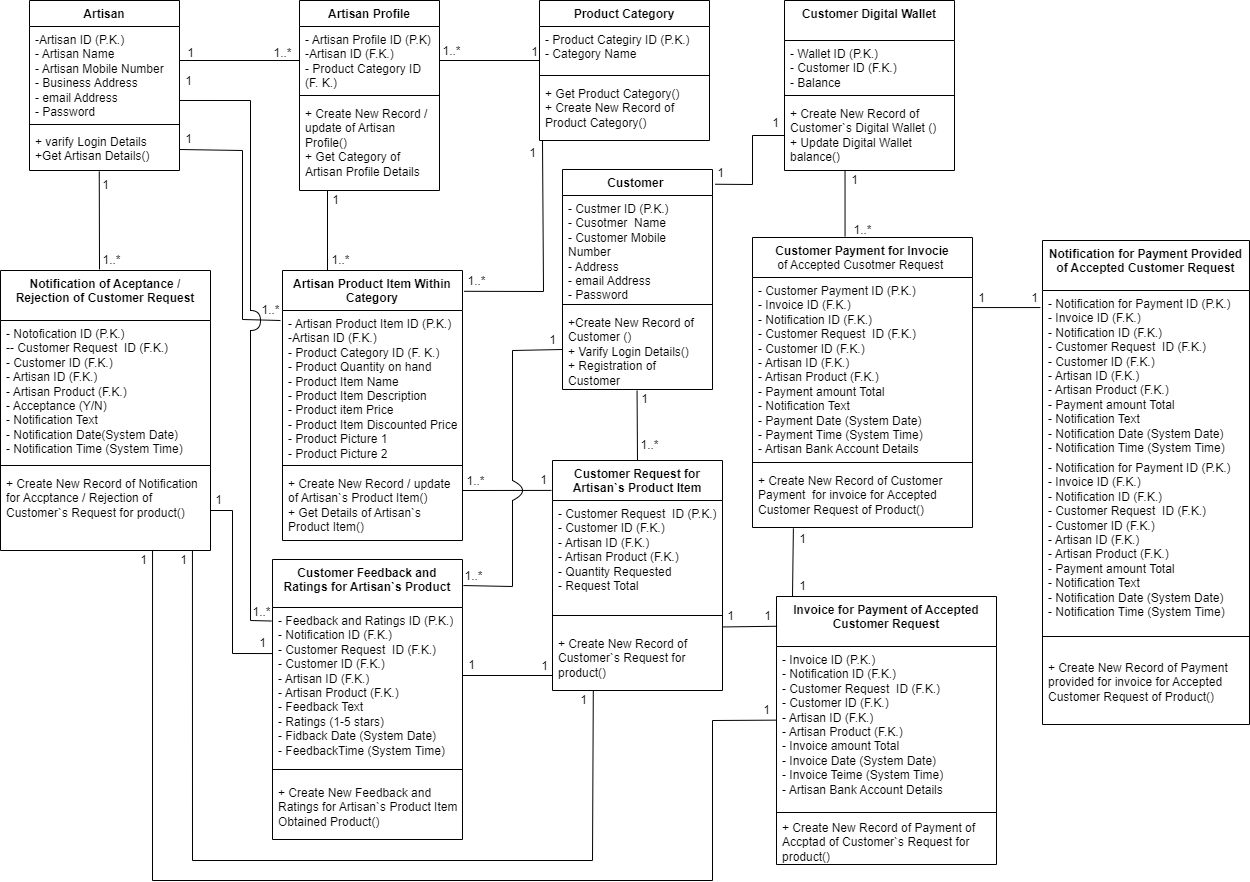
APPENDIX A4: Use Case Diagram for The Proposed Project:



APPENDIX A5: Sequence Diagram for the Proposed Project



APPENDIX A6: Object Class Diagram for the Proposed Project



APPENDIX A7: Project`s Tree Alternative Design Solutions

