**IS PROJECT 2**

**RESEARCH DESIGN ASSIGNMENT ON BANK MANAGEMENT SYSTEM**

**BBIT GROUP 4.1 A&C**

**GROUP MEMBERS**

136446 – Mirav Bhojani

136788 Nyaga Derrick

136709 Jane Mugo

136895 Wesley Wanyama

135399 Sheila Kavinya

**JULY 2023**

# **Abstract**

The case study on the bank management system describes the system's development using a prototyping process. Customers, tellers, and administrators are the system's main actors. The system requirements are divided into functional and non-functional criteria, with an emphasis on security, performance, usability, and availability. Because it works with prototypes and has the capacity to handle complicated systems, Object-Oriented Analysis and Design is the optimal system paradigm. The case study also looks into leveraging service-oriented architecture as a different system design. The differences between loose coupling and tight coupling are examined, with a focus on the benefits of loose coupling in terms of flexibility, modularity, and simplicity of maintenance. In general, the case study offers insights into system design approaches for bank management systems.

**Table of Contents**

[**Abstract** 1](#_Toc140067353)

[**List of Figures** 3](#_Toc140067354)

[**Most Appropriate System Design Methodology** 4](#_Toc140067355)

[**Ideal System Paradigm/ Approach** 5](#_Toc140067356)

[**Primary Actors** 6](#_Toc140067357)

[**Other System Architecture** 7](#_Toc140067358)

[**Sequence Diagram** 8](#_Toc140067359)

[**System Sequence Diagram** 9](#_Toc140067360)

[**Loose-Coupling and Tight-Coupling** 10](#_Toc140067361)

[**References** 12](#_Toc140067362)

# **List of Figures**

[Figure 1: Prototyping Methodology 4](#_Toc140064351)

[Figure 2: Example of SOA with various services 7](#_Toc140064352)

# **Most Appropriate System Design Methodology**

Prototyping methodology will be used in development of the system. This methodology concurrently executes the analysis, design, and implementation phases to swiftly construct a simplified version of the proposed system and get reviews and feedback from users. The prototype of the system is a rapid version of the system that offers limited features. As shown in figure below (Martin, M, 2020) , based on feedback received the developers subsequently engage in reanalysis, redesign, and reimplement a second prototype, rectifying any flaws and incorporating additional features. The iterative cycle continuous until the users reach a consensus that the prototype delivers sufficient functionality to be implemented (Pressman, 2020). The methodology will be utilized due to its inherent flexibility in design, enabling easy modification and adjustments according to the users or developers’ preferences and needs. It also allows for the convenient identification of missing functionalities and early detection of errors, resulting in significant savings in effort and cost.

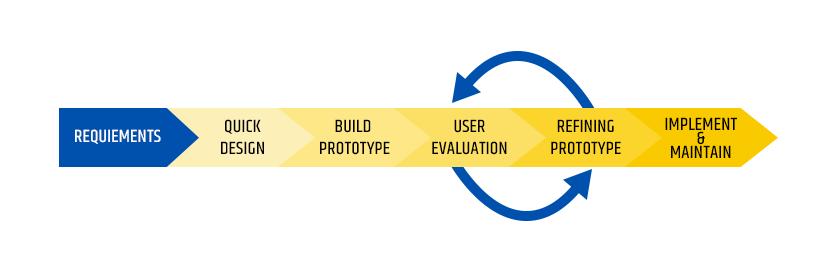


Figure 1: Prototyping Methodology

# **Ideal System Paradigm/ Approach**

The paradigm approach chosen is Object Oriented Analysis and Design (OOAD) this is because OOAD turns out to be more suitable with prototyping methodology as mentioned above since prototyping is an iterative methodology. Moreover, this approach divides the system into use cases thus understanding the interests of specific users of the system well especially for the bank application that has users with specific roles thus understanding each user requirements well, this is easier as this approach is also data oriented. Moreover, if there are any changes or upgrades required for the system it was easy to implement (Alam, 2017). Also, OOAD uses bottom-up approach which allows developing complex systems in a simpler many easing up the development process (Hammad, 2020).

# **Primary Actors**

The bank management system has three major primary actors that is the customer, the teller and the admin.

The customers usually use the system to manage their money and funds and to view their account information. For the Teller, he uses the system to withdraw, deposit or transfer the funds for the customer basically to manage the customers account upon request. Finally, for the admin his main role is to manage workers and branch information.

# **Other System Architecture**

Service-oriented Architecture (SOA)

The service-oriented architecture’s main concept is to distribute the system logic among several separate services. This is different from the traditional monolithic architecture which keeps it in a large, unified model (Vasileva, 2022).

The different segmented services are therefore able to communicate with each other either through passing data or through two or more services coordinating an activity (Kanade, 2023). SOA promotes loose coupling between services meaning they should be independent of the internal workings of other services. The services have well-defined contracts that specify the operations they provide and the data formats they accept and return. SOA also allows for the composition of services to create composite applications. One of the goals of SOA is to promote service reusability so that they can be leveraged across multiple applications and processes.

Diagram of a diagram of a service

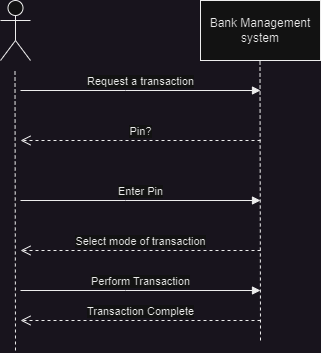
Description automatically generated

Figure 2: Example of SOA with various services

This architecture improves collaboration between IT and business, hence very suitable in an environment where the business analysts can work effectively with developers to uncover important insights. This also improves how IT organizations can make strategic decisions.

# **Sequence Diagram**

# **System Sequence Diagram**



# **Loose-Coupling and Tight-Coupling**

We can analyze loose coupling and tight coupling and their implications for system development in the context of a bank management system with actors such as customers, tellers, and administrators.

Loose Coupling in a Bank Management System: In a loosely coupled bank management system, the components that represent clients, tellers, and administrators function separately and with minimum reliance on one another. For example, the customer component can complete transactions and interact with the system without needing to know how the teller or administrative components are developed.

For communication between components, a loosely linked bank management system would have well-defined interfaces and standardized protocols. Because each component can be built, tested, and maintained individually, this enables flexibility and modularity. Changes to one component, such as the addition of a new feature or the modification of existing functionality, can be implemented without affecting the other components (Pandey, P., 2020).

In this situation, loose coupling enhances flexibility in addressing client requirements and facilitates system scaling. For example, new services or channels for clients can be added without interfering with the existing functionality of the teller or admin components. Furthermore, the system's loosely connected architecture allows for easier maintenance and evolution, as components can be upgraded or changed without requiring major changes to the entire system.

Tight Coupling in a Bank Management System: A closely connected bank management system, on the other hand, would include substantial dependencies and interactions between the customer, teller, and administrative components. For example, the customer component may have direct access to the teller or admin components' internal methods or data, resulting in tight integration between the many actors.

Changes to one component in a strongly connected system might have cascade impacts on other components (Pandey, P., 2020). For example, if the teller component is changed, the customer and admin components may need to be altered as well, thereby increasing complexity and maintenance efforts.

Tight coupling can reduce the flexibility and modularity of the bank management system. For example, adding a new service may necessitate major changes to several components, making the system more rigid and difficult to adapt (Locke, R. P., 2017). Furthermore, the lack of clear boundaries between components might impede independent development and testing, making it more difficult to detect and diagnose system flaws. (Locke, R. P., 2017).

Using a Bank Management System to Compare Loose and Tight Coupling:

The following points emphasize the distinctions between loose coupling and tight coupling in the context of bank management systems:

Dependency: Loose connectivity reduces interdependence between components, but tight coupling increases interdependence.

Loose coupling improves system flexibility by allowing individual components to be updated or extended without affecting others. Tight coupling limits flexibility and may result in system rigidity.

Loose connection encourages modular design by allowing components to be developed and tested independently. Tight connection can lead to a monolithic design, which makes it difficult to remove and maintain individual components.

Change Impact: In a loosely linked system, changes to one component have little effect on others. alterations to one component in a densely connected system may necessitate substantial alterations to numerous components.

Maintenance: Because components can be upgraded or replaced without disturbing the entire system, loose coupling promotes maintainability. Tight coupling might make maintenance difficult.

Developers can design a flexible, modular, and maintainable system that supports changing requirements and modifications in individual components by implementing loose coupling in the bank system.

# **References**

Alam, R. (2017). Object Oriented Analysis and Design (OOAD). *Cybarlab*, 1.

Hammad, M. (2020). Difference between Structured and Object-Oriented Analysis. *GeeksforGeeks*.

Kanade, V. (2023, April 28). *What Is Service-Oriented Architecture? Working, Principles, and Benefits*. Retrieved from spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-service-oriented-architecture/

Locke, R. P. (2017). Retrieved from https://dzone.com/articles/the-risks-of-tight-coupling

Martin, M. (2020). *Prrototyping in Software Engineering: Methodolgogy Process*. Retrieved from Ace Infoway: https://www.aceinfoway.com/blog/project-scope-and-prototype-model

Pandey, P. (2020). *Understanding Loose Coupling in Software Development*. Retrieved from https://dzone.com/articles/understanding-loose-coupling-in-software-development

Pressman, R. S. (2020). *Software Engineering: A Practitioner's Approach.* New York : McGraw-Hill Education.

Vasileva, A. (2022, August 16). *Software Architecture Styles and the Projects They Suit Best*. Retrieved from DREAMIX: https://dreamix.eu/blog/software-architecture/software-architecture-styles-and-the-projects-they-suit-best