Chinwe Reverse Software Engineering Model: A Case Study of Electronic Bookshop System

Ndigwe C.F.
Computer Science Dept. Chukwuemeka Odumegwu Ojukwu, University
Uli Campus, Anambra State

Okeke O.C.
Computer Science Dept. Chukwumemeka Odumegwu
Ojukwu University.Uli Campus, Anambra State

ABSTRACT
Chinwe Reverse Software Engineering (CRSE) model for Legacy Software was developed to enable software engineers to build application by reverse engineering legacy application using the best features of the classical system with contemporary advanced technology. This will enable the reverse engineered system to adapt to modern environment while preserving the good features of the legacy application. In this paper a use case is presented for use to testing the CRSE model and presenting the detail design of the system in illustration of how the model can be used in real life software engineering development. This will serve as a guide to academics and developers on the deployment of the model in other use cases and projects. The Use case used is an Electronic Bookshop System with a legacy Java application that work offline but is reverse engineered to work online. The result of the design show a successful decompilation and refactoring of the system illustrating the veracity of the CRSE process model in reverse engineering of legacy software.

General Terms
Chinwe Reverse Software Engineering Model, Use Case, Decompilation, refactoring

Keywords
CSRE Model, Code Refactoring, legacy software, Use Case Desig, Class UML Design

1. INTRODUCTION
Reverse engineering often refers to moving from running binary into source code and then to design and back to code again. In Chinwe Reverse Software Engineering (CRSE) Model (Ndigwe and Okeke, 2025) is a tailored reverse software engineering model specifically designed to address the intricacies of legacy systems. It incorporates new processes that enable legacy software without source code to be decompiled and the recovered code properly refactored. The refactored component provides a working design that can be used in building a new application that will be compatible to contemporary computing environment and new need.

In this paper a use case of an Electronic Bookshop legacy system is presented and the CRSE model deployed to reverse engineer the legacy system. This offer an experimental prove of the usability of the CRSE model in real life cases of reverse software engineering and also of the workability of the model when tested using real life scenario. It will provide insight into the complexities of legacy systems, helping organizations to better understand and manage their existing software assets. It is expected to improve Reverse Engineering Practices by developing a use case scenario, that will enhance current reverse engineering practices, making them more effective and applicable to diverse legacy systems use cases. The use case test will also offer clear strategies for refactoring technics offered by the model. The way the refactoring process captures the main legacy features desired in the system and how it integrates them with modern features desired in the final product of the reverse engineering process is also clearly highlighted in this work.

2. CASE STUDY
A case study is a deep, detailed examination of a specific real-world subject to gain in-depth understanding of its complexities, dynamics, and context, using real life scenarios to explore "how" a system work and "why" something happens, offering insights that may not generalize but provide rich, contextual knowledge. Key Characteristics of the case studies include:

In-depth Analysis: This goes beyond surface facts to understand the intricacies of a specific case.

Real-World Context: Studies subjects in their natural settings, not controlled experiments.

Versatile Subjects: This Can focus on individuals, groups, organizations, events, or phenomena.

Case studies can be applied in Business in analyzing a company's marketing strategy or a project's success. In Medicine for studying a patient's unique illness or treatment response.

In psychology/social work in understanding a person's behavior or a community's challenges. In education for evaluating a new teaching method's impact on a classroom.

Use cases provides rich, contextual understanding of complex issues and it is Useful for exploring new topics or models and uncover patterns and cause-and-effect relationships in real life. In this work it is used in reverse software engineering for testing and verifying the CRSE model.

In a reverse software engineering case study, analyzing existing software (often legacy systems) to understand its design, functionality, and architecture for maintenance, modernization, or reuse, with examples showing processes like use case realization. These aim to recover lost documentation or improve quality, often through techniques to extract system behavior, components, use cases and architectural patterns. These studies typically detail phases of the reverse software engineering model studied in our case the CRSE model.

In the selected case study the target include:

Legacy System Modernization: Recovering design from an old system to rebuild or integrate it.

Architecture Recovery: Analyzing the existing project to

understand their conceptual architecture for reuse or maintenance.

Analyze System Behavior: Observe how the system functions, often via use cases

Design Recovery via Refactoring : Reconstruct high-level models (like UML diagrams) to understand the original design intent.

Reconstruction: Use recovered artifacts to create improved versions or new compliant systems.

In this paper a case study of an online bookshop, owned by Csoft Nigeria Limited as the software development company and Ubookshop Limited as the system owner; (these are hypothetical companies). The book buying public is the system user. The existing legacy Software is an offline system developed using Java but need to be reverse engineered to an online system in a new environment. We have chosen PHP, an open source object-oriented programming language as the Language of implementation and MySQL as the database engine for storing the information and data required for the system.

The case study is used to test a reverse Software Engineering Model for Legacy Software code -Chinwe Reverse Software engineering model (CRSE) (Ndigwe and Okeke, 2025). These case study will present each process of the model and show how it is tailored to specific activities from the process of acquiring the legacy application through the decompilation stage to source code and context parsing. The process of component analysis, domain knowledge extraction, will be examined and then to code refactoring and design recovery. The final stage of the CRSE process is design reconstruction which involve the coding of the refactored system using tools that are compatible with contemporary computing environment.

This process when examined in detail using a case study will be important in enhancing understanding of Legacy Systems. It will provide insight into the complexities of legacy systems, helping organizations to better understand and manage their existing software assets. It will also provide a step by step guide that can guide software engineers in reverse engineering legacy application and providing wide real life experience that can be replicated when they handle the reverse engineering of legacy system. It also improve Reverse Engineering Practices by developing a tailored process based on the CRSE model, that will enhance current reverse engineering practices, making them more effective and applicable to diverse legacy systems. The case study will also offer clear experimental strategies for integrating legacy systems with modern technologies, aiding organizations in their digital transformation efforts.

Scenario Description: Classical Application without Code

The case study has a scenario which the legacy application falls into, which often include availability of source code and unavailability of source code. Software engineering from CRSE perspective is more than concepts, tools, techniques, methods and processes. In the case study presented in this work the unavailability of source code is assumed.

In the scenario, Chinwe Ndigwe, who is the Reverse software engineer and director Csoft Nigeria Limited (Csoft) receives a letter from Jonah the Software developer and project secretary Csoft. Chukwudi Nda is the System analyst/programmer at Csoft is seated. Musa Umar Programmer I at CSoft is busy on the keyboard, while Gbenga Olukoya the marketing manager at CSoft just works into his desk with a client request demanding that a classic Bookshop software be made to adapt to their new work environment with Apache, PHP and MySqL as their new tools. It was further discovered that the application he has brought for Reverse engineering does not have a source code but was a Java offline application.

In other to use this scenario in testing the program, the process model of CRSE will first be discussed to serve as a guide to the process this work intend to experiment on.

3. CRSE PROCESS MODEL

The Chinwe Reverse Software Engineering (CRSE) process model (Ndigwe and Okeke, 2025) is a process model developed for use in reverse engineering legacy systems. It has different components and major parts which when combined in certain steps form the process model as illustrated in figure 1.

The components include:

3.1 Decompilation

Decompilation is the process of converting compiled machine code or bytecode back into a highlevel programming language. This process is typically used to analyze how the software works when the source code is not accessible (Xu et al.,2023) Decompilation (Tan, et al.,2024; Wong et al.,2023; Feng et al,2024) is the reverse process of converting compiled binary code back into a high-level programming language, with the goal of recovering source code that is functionally equivalent to the original executable (Feng et al.2025). Decompilation has alluring application value in performing various reverse engineering tasks, such as vulnerability identification, malware analysis, and legacy software migration (Yunlong, et al., 2025). Decompilation reverses the compilation and transforms the binary code back to high-level code (Wei-Cheng, et al., 2024). Since compilation is not a fully reversible process, it is difficult to construct a decompiler that can decompile an arbitrary executable back into the high-level code it was compiled from (Jerome and Laurie. 2002). Reverse engineering can be used to find bugs, vulnerabilities, and malware, and perform verification, comparison, and multi-platform portability (EelcoVisser. 2023).

In this work the legacy application is only available in its binary form as legacy application so it is first decompiled to generate the source code. If the legacy source code is readily available there will not be the need to decompile anything effectively using the second entry point as illustrated in figure1. However when the application is in the binary form the model uses decompilation to generate the source code before the source code is subjected to Context parsing. which is included for classical application that have no source code.

3.2 Source code

Source code is the human-readable text of instructions written by programmers in a specific language (like Python, Java, C++) that tells a computer how to perform tasks, acting as the blueprint for any software, website, or app (Krysa and Sedek, 2008)). Computers don't understand this code directly, so it's either translated by a compiler into machine code (0s and 1s) or executed on-the-fly by an interpreter, making it the essential building block of all digital tools.

3.3 Context parsing

Context parsing is the process of analyzing source code to understand its syntactic, structure and the broader circumstances/information (context) that influence its interpretation and function (Eric, 2023). This process is a foundational step in compilers, code analysis tools, and modern

AI systems for code understanding. It is equally used in decompilation and reverse engineering. The process that source code pass through to be converted from high level language to machine language involve this process. It is also used in reverse engineering to extract desired context for reuse during refactoring.

The general process of parsing source code involves several stages: The raw stream of characters in the source code is broken down into a sequence of meaningful units called tokens (e.g., keywords, identifiers, operators, numbers). The stream of tokens is checked against the language's formal grammar rules to determine its grammatical structure. A parse tree or, more commonly, an Abstract Syntax Tree (AST) is generated, which is a hierarchical representation of the code's structure. This phase ensures that the code makes sense logically, checking for things like type compatibility, correct variable usage, and function calls. This functions (method) and its calls are used to establish classes and their relations in the new target modern design during refactoring. Once the design is created it can be used by software engineers in building new systems.

The immediate hierarchical relationship of a piece of code within the overall structure, as captured by the AST. For example, understanding that a variable is declared inside a specific function or class. The specific circumstances under which the code is run or analyzed, such as the available libraries, the operating system, or the state of variables at runtime.

For large language models (LLMs) used in code analysis, "context engineering" involves providing additional information, such as related files, documentation, or function descriptions, to help the model understand and generate correct code. This is often used in techniques like Retrieval Augmented Generation (RAG).

3.4 Component analysis

Before starting to reengineer a software product it is necessary to analyze the source code components in order to understand deeply the existing implementation (Radoslav and Gennady 2010). Component analysis is the process of identifying and evaluating third-party software and hardware components within a system to manage associated risks. When focused solely on software, it is commonly called Software Composition Analysis (SCA).

The **t**ools used typically scan the codebase and dependency files (like `pom.xml` or `package.json`) to create a comprehensive inventory of all external components and their versions. The source-code analysis is an expensive task and every useful tool automating (even partially) this process could decrease the time and price of the product and improve the overall productivity(Radoslav and Gennady 2010)..

The purpose of component analysis is to assess risks associated with using external components, specifically regarding security vulnerabilities (e.g., Log4j), license compliance, and component obsolescence. Once component analysis is done a detailed Software Bill of Materials (SBOM) is produced and reports on known vulnerabilities (often linked to public databases like CVEs), as well as licensing obligations (e.g., copyright, copyleft). Modern software heavily relies on open-source components, making it crucial to manage the risks of code not written in-house

3.5 Domain knowledge extraction

Program understanding involves mapping domain concepts to the code elements that implement them. Such mapping is often implicit and undocumented. However, identifier names contain relevant clues to rediscover the mapping and make it available to programmers (Surafel and Paolo, 2015). Source code domain knowledge extraction is the automated process of uncovering industry-specific concepts and relationships (domain knowledge) embedded in software by analyzing its structure, identifiers (names), and patterns, often using NLP and Machine Learning to build domain ontologies for better understanding, maintenance, and reverse engineering, filtering out implementation noise for clearer insights (Konys, 2018). Building domain ontologies requires the access to domain knowledge owned by domain experts or contained in knowledge sources (Gómez-Pérez et al.,2004; Suárez-Figueroa et al., 2015). However, domain experts (especially those that build the legacy application to be reverse engineered) are not always available for interviews. And in case they are available, the knowledge provided is often incomplete and subjective. In addition, as the domain evolves, the knowledge provided by the experts is likely to be out of date. (Azanzi et al.,2019). In handling this situation code structure (classes, methods, calls) are exploited to find domain concepts and relationships, often creating ontologies. Natural Language Processing can be applied to program identifiers (variable/function names) to extract domain terms and concepts.
In some instances hybrid methods are used by combining structural and linguistic data for richer extractions, sometimes using information retrieval (IR) or topic modeling to filter out low-level implementation details. Machine learning such as Hidden Markov Models (HMMs) for generalizable extraction or LLMs (Hamed et al.,2025) with prompt engineering for complex, open-domain tasks are used. Computationally efficient statistical methods or more precise, complex symbolic reasoning can also be used.

The challenges include (Noise Filtering**)** distinguishing important domain concepts from specific implementation jargon (e.g., `tempVariable`, `iterator1`). Handling different ways developers implement the same real-world idea and scalability in processing large codebases effectively. While a large number of facts can be extracted automatically, there are still ambiguous and sensitive cases that need to be verified by human. Human curation is seamlessly integrated with ODKE framework (Kun et al., 2023).

3.6 Code refactoring

Refactoring is the process of restructuring code, while not changing its original functionality. The goal of refactoring is to improve internal code by making many small changes without altering the code's external behavior. Software reverse engineers refactor code to improve the design, structure and implementation of software. Refactoring improves code readability and reduces complexities. Refactoring can be done by binding data with functions for classes and objects so that the benefit of object modelling can be derived after the new model is generated during design reconstruction, Design antipatterns are major symptoms of poor choices at the software architecture level. These bad programming practices typically violate object-oriented design principles, such as Single Responsibility and Law of Demeter. Two popular examples of design antipatterns are God Class and Feature Envy. The first characterizes classes that are abnormally large and monopolize most of the system’s behavior by controlling a significant number of other coupled classes. Decomposing this class is trivial to sustain the modular design of the system (Eman et al., 2023). The second is related to methods that heavily rely on methods and attributes that are outside of its class more than

those inside it. This is a symptom of a misplaced method that needs to be moved to a class to make it more cohesive. To cope with these antipatterns, refactoring has emerged as a de-facto practice to improve software quality through the removal of antipatterns (Eman et al., 2021). Refactoring is the art of improving source code internal design, without altering its external behavior (Fowler et al.,1999). Legacy system often have this antipatterns or they may not even be object-oriented risking security and other pitfalls and the design can be changed by refactoring them.

3.7 Design recovery

Design recovery intend to develop structures that will help the software engineer understand a program or system. Understanding is critical to many activities - maintenance, enhancement, reuse, the design of a similar new system, and training and reverse engineering (Ted,1989). Link et al. (2019) define Software Architecture Recovery (SAR) as "the process of recovering a system's architecture from its implementation artifacts, such as its source code. Design recovery is the process of reverse-engineering high-level design information (like architecture, patterns, and structures) from existing source code, especially for legacy systems lacking documentation, using techniques like static/semantic analysis to bridge the gap between code and design models, often involving parsing code, identifying patterns (structural, behavioral), and generating diagrams for better maintenance and reuse. Key challenges involve dealing with "noise" from utilities, while methods focus on identifying relationships, generating hierarchical structures, and validating patterns through fine-grained analysis. Reconstruct design abstractions (UML, patterns, architecture) from code to aid maintenance, understanding, and component reuse. Shahla et al., (2024) used Java IDEs with advanced code navigation and code refactoring capabilities integrated with blockchain extractor to recover design pattern which result to the extracted design patterns that are stored in the nodes of the blockchain.
Examines code structure, class hierarchies, dependencies. Studies runtime behavior to find patterns.Uses meaning and context (attributes, naming) to infer design intent. Processing files to build a hierarchical database of components and relationships. Using parsing and validation to find design patterns (e.g., Strategy, Observer). Linking code elements to design artifacts using property similarity. Removing utility components (like standard libraries) that obscure the core design. The result is a visual diagrams (UML), textual descriptions, or a browsable database of the recovered design.
Reconstructing code from compiled binaries (DLLs) for systems like Power Platform plugins, though with potential differences.

When design documents are missing or obsolete in legacy systems. To understand complex codebases for maintenance or modernization. To harvest reusable components.

3.8 Design reconstruction

Source code design reconstruction" refers to the process of reverse engineering a system's high-level design, architecture, or models from its existing source code, often using tools and techniques to aid comprehension and potential refactoring. Advancements in Large Language Models (LLMs) have become a prominent focus in research to automate and enhance this process. Code generation stands as a powerful technique in modern software development, improving development efficiency, reducing errors, and fostering standardization and consistency(Youjia, 2024) Software Architecture (SA) is central to managing complexity during software development by establishing the system's core structure, key components, and their interactions. As a foundational artifact, it guides long-term system evolution, supports informed technical decisions, and ensures alignment between system requirements and implementation. To effectively fulfill this role, the architecture must be explicitly defined and systematically documented(Ahmad et al., 2025).

3.9 CRSE Model Architecture

In the CRSE model architecture in figure 1 these components and the processes they go through is clearly illustrated. In the selected case study the decompilation process is used. This is the process where the application in binary is first decompiled to generate the source code. However when the application is in the binary form the model uses decompilation to generate the source code before the source code is subjected to Context parsing and refactoring.

CRSE is an Agile Software Reverse Engineering model and is believed to be a dynamic process of building reverse software and dynamically specified since reverse software is expected to result to a new improved system with design features of the legacy application (Ndigwe and Okeke, 2025). In figure 1 the proposed system considers the two main form of legacy applications, the application that have source code and the one that the source code is not available. In this case study the application source code is not available and the application itself need to be decompiled. The application is in binary format or in bytecode (for applications using virtual machine or run time to execute such as Java or dot Net). These application file in that format is decompiled to extract the source code. The source code is then sent for context parsing to extract domain knowledge. The source code was also refactored using Agile software method refactoring technique for design recovery. On the other hand the parsed code is represented in intermediate format, a detail component analysis is carried out on the system before it could be refactored, which lead to design recovery. In design recovery the new system information architecture and the new system requirement are integrated to the recovered design from the legacy system. During design reconstruction expert knowledge is injected into the system for the creation of a new system (Ndigwe and Okeke, 2025).. The new system comes out as an improved system of the legacy application that have being reverse engineered as the base design of the system. This have the benefit of inheriting all the merits of the legacy system with new injection of more modern design artifacts that enable the system adapt to new execution environment.

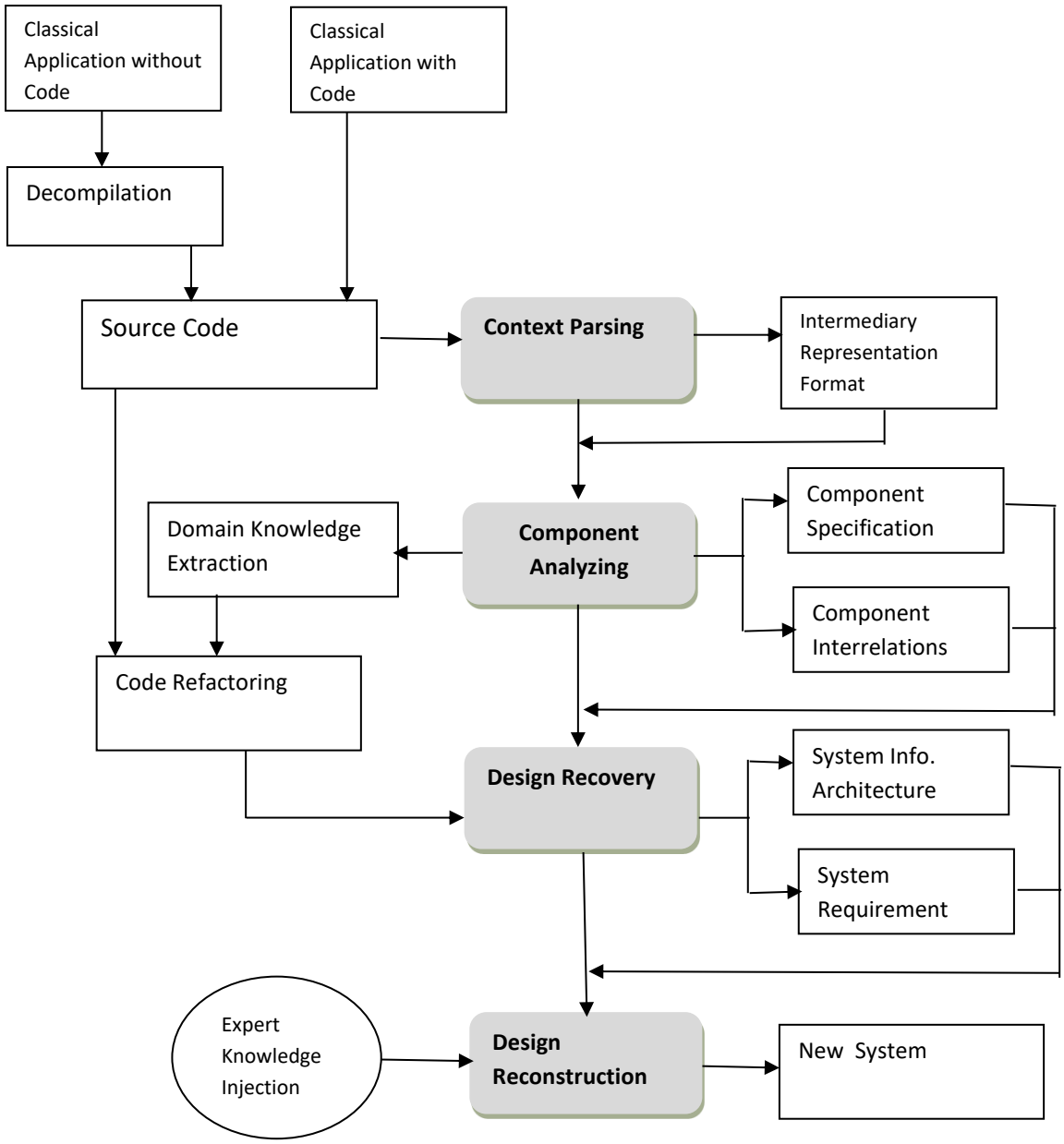

Fig 1: The CRSE Process Model (Source: Ndigwe and Okeke, 2025)

The target of these process activities is working Software in the contemporary computing environment-new operating system, more modern databases, new platform and even new users. In the case study in this paper the new platform is online rather than the offline environment which is the platorm of the legacy system This is a major justification identified for the CRSE model. The Operating system of host environment is Linux instead of Windows 10 of legacy system which is no longer supported by Microsoft. It was realize that these issues are also important in getting software in another domain to be working in another domain once reverse engineered. CRSE model rules were followed in the reverse engineering of the legacy system in the case study presented in this work, showing the use case and the design detail of the system which can also be followed in the reverse engineering of other legacy system similar to the one reverse engineered in this case study.

4. USE CASE- MODELLING

Refactoring produces a design from where the legacy system artifacts are extracted and integrated into the new system. The refactoring produces a new design, and the use case of the new system is extracted as presented in this work. The particular components that is required for the case study based on the use for the new system is presented. The Use case is divided into units for the purpose of detail description.

4.1 Project Initiation Use Case

Scenario: Worlu worked into the open office and announced that Ubookshop has indicated interest in building an online store to boost its sales to a wider market after a proposal is submitted to them. Mean while there is a legacy Bookshop app that have been working offline but the source code is no longer available. A project team is immediately selected to meet Ubookshop for more clarification. A meeting holds between CSoft and Ubookshop and an agreement is reached to build an online store for Ubookshop by reverse engineering their legacy system. Figure 2 illustrates the UML use case of the project initiation state. In the figure there are three actors, the Csoft Management, the Csoft Reverse Software Engineering Team and the UBookshop Operator acting as owner of project. The Actions they take to initiate the project include Announcing UBookshop intent, send info to Csoft management, hold a

reverse software Engineering meeting, select Csoft Project Team and Schedule Decompilation. These actions are performed by different actors/

Project Initiation
The agreement is sealed with the payment of a consultancy fee and a documentation of the contract agreement signed. A meeting was scheduled to hold in ten days to fine tune details of the project.

CRSE Plan
The team meets and outlines the following New functionality to be added to their classic legacy Software that needed to be Reverse Engineered to work online:

i) A database for the books Ubookshop wants to sell online
ii) Online catalog of all the books by category
iii) Shopping cart to track the books a user wants to buy
iv) Checkout section that processes payment and delivery details
v) An Upgraded administrative interface used for adding books, removing books, fixing and updating prices and other general administrative activity.

From the New requirement gotten from the client (the UBookshop Team), users are supposed to:

i) browse books based on category for instance by field of study, publisher or author.

ii) Users should also be able to select item from the catalog and the system should be able to track which items are selected.

iii) The system should be able to total up users order, take their delivery details, and process their payment when they finish shopping.

iv) An updated administrator interface to Ubookshop site should also be built so that the administrator can add and edit books and categories on the site.

Figure 2 clearly illustrates the actors of the case study which are majorly the administrative stakeholders of the project. They include CSoft management, UBookshop and CSoft Reverse Engineering Team. These stakeholders are very important in domain knowledge extraction in getting the deferent requirement needed for injection into the reverse engineered project. The actions include Announce E-Bookshop project, Hold Reverse Engineering Meeting, Select Csoft Project Team, Send info to Csoft management. The other set of requirement are derived from the refactoring of the system most of which are generated from the legacy code. Once the stakeholders actions are completely articulated the new requirement for the new system will be made up of both the legacy system refactored design and the knowledge domain generated from the stakeholders concerning the new software working environment.

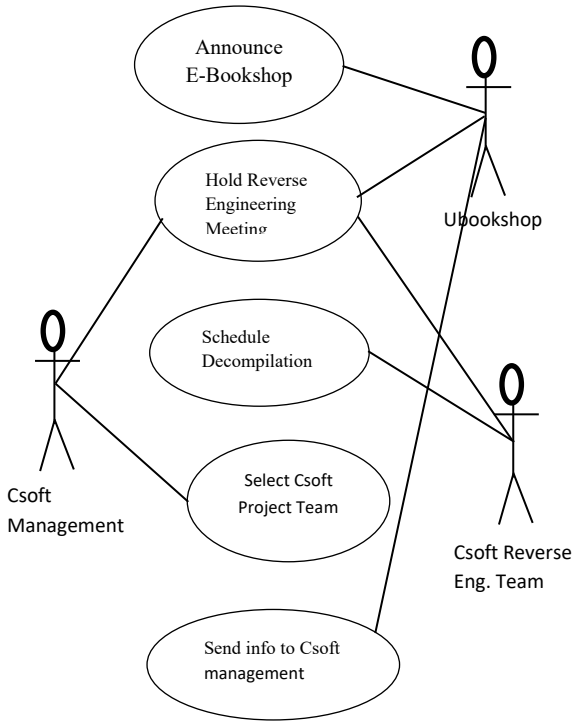

Fig. 2: UML Use Case of Project initiation

4.2 E-Bookshop Use Case

In figure 3 the UML E-Bookshop Use case indicates two main updated functionalities, admin and general function performed by admin and users (Ubookshop) respectively. In the legacy system which is offline the bookshop software that is being reverse engineered and made to function online will need new or updated system requirements that is presented during the meeting. These new requirement will be added in the new system that is being built and properly re-integrated with the old requirements which the old software that is being reverse engineered have. This not only help the system to be effective but very functional in the new online environment where it is being deployed. The Chinwe Reverse Software Engineering (CRSE) model use require proper planning and analysis using Agile methodology guideline.

Ubookshop is based in Nigeria and will also like to sell books that are written in English and local languages like Igbo, Ijaw, Efik Yoruba and Hausa. That calls for possible integration of the languages for users' pleasure.
From the users perspective these requirements are simple but we know that from software engineering perspective they are up-hill tasks that require serious implementation efforts.

The database backend need to be very effective in handling request when more users make request from the system and when there is expansion on the user base of the system. These will need more detail consideration of the condition of storage and the database ability to respond to the need. The host location though a post deployment requirement also need to be considered at this design stage. Considering a cloud host for scalability, a shared host for the system or a dedicated host all border to the need of the stakeholders.

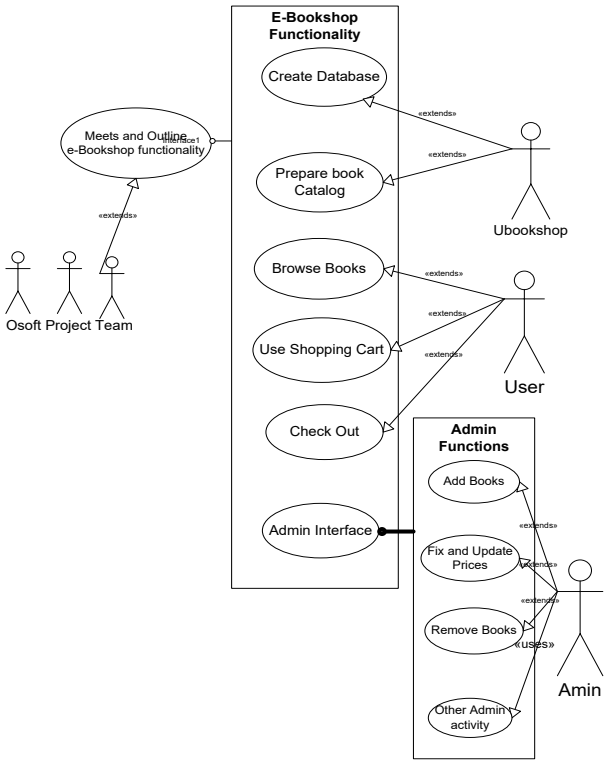

Fig. 3: UML E-bookshop Use Case

4.2.1 CRSE Analysis of the E-Bookshop Use Case

-Building the Database backend

We need to build online catalog hence we need to develop a database, using MySQL, that will handle all the data and the categories of books, shipping addresses, payment details and so on.

-Building the tracking of user purchases

We can build the tracking section, which tracks a user's purchase when the shopping is going on, either by putting book selections into our database or by using a session variable. In CRSE we consider a balance of usability and efficiency so we select session control. The system therefore will need session variables to store a user's selection to save the overhead of constant querying of the database for the information. When the user finishes shopping and pays for the items the information can then be put in the database as a record of the transaction. The stored information can also be used to give sales report.

-Build payment processing

Payment involves lots of issues which in an open source system call for outright reuse of payment systems of a reputable online merchant. In the system we will only build an interface that will connect, instantiate and link to the payment gateway. In the payment system one needs to organize a merchant account with a bank for the cards to be accepted. When done we can remove the session variables.

-Building administrative interface

This interface will allow Ubookshop to add, delete, and edit books and categories from the database. This will allow Ubookshop staff to change book prices as the publishing cost changes or to remove a book that has been out of stock for a very long time.

4.2.2 CRSE Analysis of the Legacy System Requirements

The legacy software have some beautiful requirements that need to be maintained but improved upon hence. The new requirement may have been submitted by the UBookshop Team but CRSE performs lots of outside requirement abstractions to unearth requirements from a virtual dimension. This abstraction represents unbounded group of possible requirements and behaviors. For instance Ubookshop team may not know that online robot may come to buy products. Such possibilities need to be considered so that such robots may not come in and place some wonderful order that will distract system and management attention. Security of payment processing may not be remembered by Ubookshop team but it is a very serious requirement for the success of the system. There is a need to make provision for security in all information on transit across the client and server of the E-Bookshop system. In other to achieve this the system may have to make member variables to be private and avoid global variables in client server-communication. The CSRE in addition considers the ***security unknown*** also and take into account the fact that the system may change during its life cycle. The stability requirement is also unknown if the online bookshop will last above the first version. The database -MySQL version used in developing the system may change, the programming language-PHP version also used in the development of the system may also change. Similarly the server (Apache) version where the system runs may change hence we must also have ***stability unknown*** as built-in component of the system.

5. CRSE E-BOOKSHOP DESIGN

Before looking into the detailed design CRSE encourages minimal documentation or reuse of documentation done in the classical software that is being reverse engineered. In CRSE, Software entities (classes, modules, functions, etc) should be open for extension, but closed for modification, a principle adopted from agile methodology. This means that it is undesirable for a single change to a program to result in a cascade of changes to dependent modules. The open-closed principle resolves this by recommending the design of modules that never change. If requirement changes, the behavior of such modules should be extended by adding new codes, not by changing old codes that already work as recommended by open-closed. This will help in the recoding and reuse of the essential modules in the classical software being reverse engineered.

CRSE on the other hand believes that if requirement changes a module can be refactored to accommodate the change without affecting the external modules. Even if the design has virtual or abstract class implementation, conditions may warrant the addition of a virtual member to the class. Extending the abstract class does not make a good design sense. It is better to refactor the abstract class to reflect the new changes which will not affect other classes or their object implementations.

5.1 System View Design

In the preliminary design, the system flow design is provided to guide the process. This will help the CSRE team on the components that are required. It will also guide the team in selecting components that can be reused and those that must be coded and the once that can be refactored and used in the project. The views of the system from the requirement includes the user view at the front-end and the administrator view at the back-end. Figure 4.5 illustrates the user view system design of the Ubookshop system which allows the users browse books by category, view book details, add books to their cart, and purchase them.

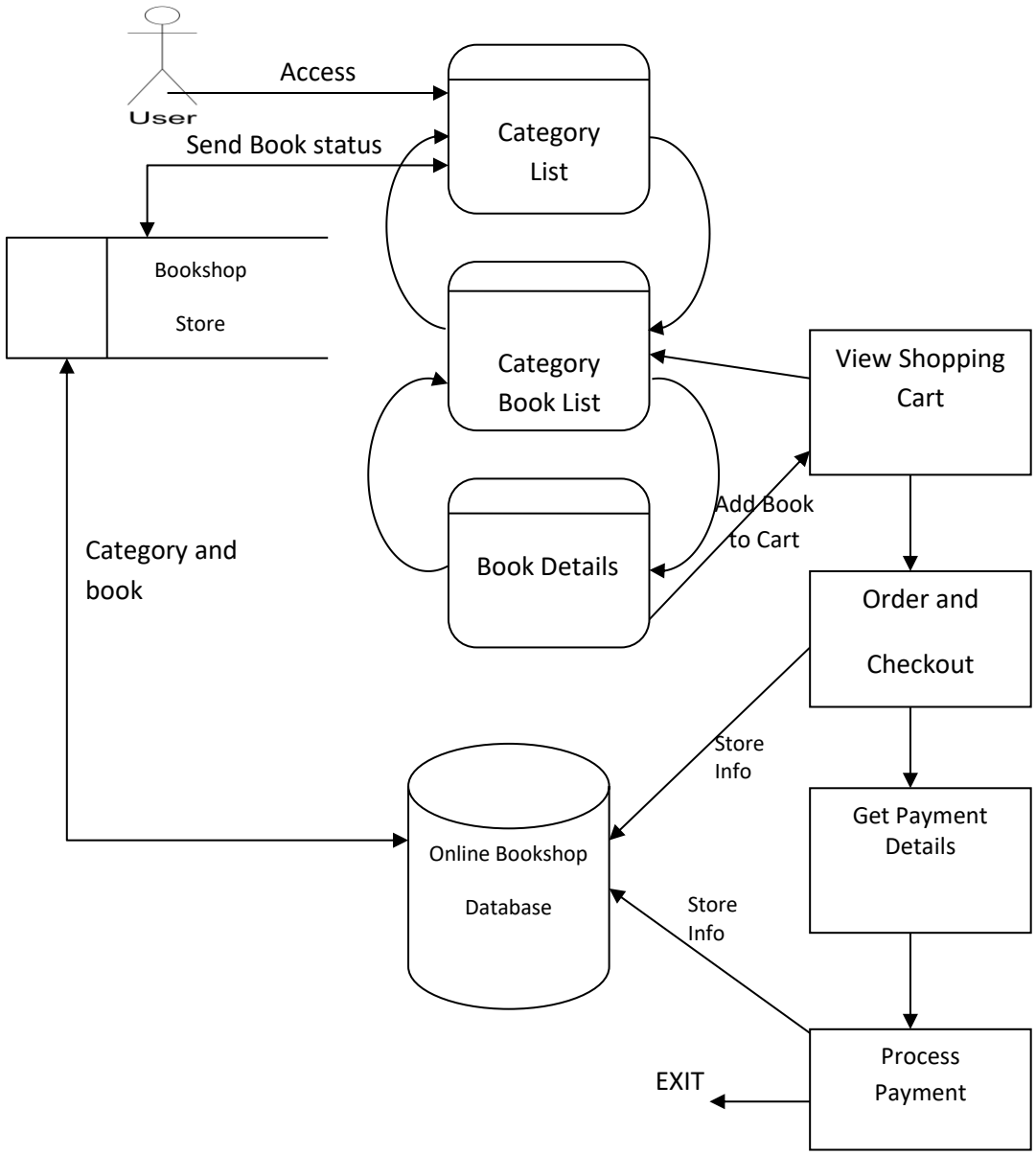

Fig. 4: User View System Design

The User view design presents the software perspective of how the user interaction with the system should flow within the refactored software system. It directs the development of user's interaction from the front-end of the application to the backend including databases. It also shows how the components are supposed to connect to one another guiding the development process. The main links between the scripts in the user part of the online site as shown in figure 4.5. A customer (User) gets to the main page, which lists all the categories of books in the site. The user can proceed to a particular category of books, and then gets the individual book details. The user is also given a link to add a particular book to the cart from where he can order and check out of the online bookshop. The information collected from the user after placing order and making payment is recorded in the database.

6. CONCLUSION

In conclusion a case study of a E-bookshop legacy system have been presented for reverse software engineering using the CRSE process model is use case was clearly presented and the states that the system will pass through described. The design of the system is also presented indicating extracted requirement both from the legacy system and from the stakeholders of the new system. The paper presented the need to blend this requirement into one for the purpose of development of the new system that will meet the online use conditions expected from the offline legacy system that is reverse engineered.

7. RECOMMENDATION

In this paper, we recommend that the case study design presented in this paper should be used for reconstruction using a programming language like PHP and a databases model like MySQL. The design reconstruction offers the case study last point of reverse engineering which is expected to produce the new online E-Bookshop system. This process can also be used in reverse engineer other legacy software which have similar or different nature. Developers can also use CRSE model in guiding its reverse engineering process.

8. ACKNOWLEDGMENTS

Our thanks to Eke Bartholomew (Phd) the experts who reviewed the work and pointed corrections during the development of this paper.

9. REFERENCES

[1] Ndigwe C. F. and Okeke O. C (2025) Dual-Step Phase Reverse Software Engineering Model for Legacy Software System, International Journal of Computer Applications (0975 – 8887) 187 (61).

[2] EelcoVisser. (2023). Why decompilation. http://www.program-transformation. org/Transform/WhyDecompilation

[3] Wei-Cheng W., Yutian Y., Hallgrimur D. E., David P., Steven C., Christophe H. and Weihang W. (2024) Is This the Same Code? A Comprehensive Study of Decompilation Techniques for WebAssembly Binaries, SecureComm 2024, Dubai, United Arab Emirates

[4] Xu X., Zhuo Z., Shiwei F., Yapeng Y., Zian S., Nan J., Siyuan C., Lin T., and Xiangyu Z..(2023). Lmpa: Improving decompilation by synergy of large language model and program analysis. CoRR, abs/2306.02546.

[5] Feng, Y.; Li, B.; Shi, X.; Zhu, Q.; Che, W. (2025) Interactive End-to-End Decompilation via Large Language Models. Electronics 2025, 14, 4442. https://doi.org/10.3390/ electronics14224442

[6] Tan, H.; Luo, Q.; Li, J.; Zhang, Y. (2024) LLM4 Decompile: Decompiling Binary Code with Large Language Models. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Miami, FL, USA, 12–16 November 2024; pp. 3473–3487.

[7] Wong, W.K.; Wang, H.; Li, Z.; Liu, Z.; Wang, S.; Tang, Q.; Nie, S.; Wu, S. (2023) Refining Decompiled C Code with Large Language Models. arXiv 2023, arXiv:2310.06530.

[8] Feng, Y.; Teng, D.; Xu, Y.; Mu, H.; Xu, X.; Qin, L.; Zhu, Q.; Che, W. (2024) Self-Constructed Context Decompilation with Fined-grained Alignment Enhancement. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, FL, USA, 12–16 November 2024; pp. 6603–6614.

[9] Krysa J. and Sedek G (2008) Source Code, doi:10.7551/mitpress/9780262062749.003.0034

[10] Eric V. W. (2023) Context in Parsing: Techniques and Applications, Commemorative Symposium (EVCS 2023). Article No. 30; pp. 30:1–30:10 OpenAccess Series in Informatics, Germany

[11] Radoslav K. and Gennady A. (2010) Source Code Analysis – An Overview, Bulgarian Academy of Sciences Journal of Cybernetics And Information Technologies 10(2) 60-77, Bulgaria.

[12] Azanzi J., Gaoussou C. and Maurice T. (2019) Extracting ontological knowledge from Java source code using Hidden Markov Models Open Computer Science 9(1), 181-199, De Gruyte. DOI: 10.1515/comp-2019-0013

[13] Suárez-Figueroa M. C., Gómez-Pérez A. and Fernández-López M. (2015) The NeOn Methodology framework: A scenario-based methodology for ontology development, Applied Ontology, 2015, 10(2),

[14] Kun Q., Anton B., Fei W., Samira K., Azadeh N., Rahul K., Yisi S., Katherine L., Xianqi C., Eric C., Yash G., Chloe S., Yiwen S., Ahmed F., Theo R., Ihab I., Xiaoguang Q. and Yunyao L. (2023) Open Domain Knowledge Extraction for Knowledge Graphs,

[15] Surafel L. A. and Paolo T. (2015) Extraction of domain concepts from the source code, Science of Computer Programming, 98(4) 680-706, Elsevie

[16] Eman A. A., Mohamed W. M. and Ali O. (2023) Automating Source Code Refactoring in the Classroom, In Proceedings of ACM Conference. ACM, New York, NY, USA. https: //doi.org/10.1145/nnnnnnn.nnnnnnn

[17] Fowler M., Kent B., Brant J, William O., and Roberts D. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. http://dl.acm.org/citation.cfm?id=311424

[18] Eman A. A., Mohamed W. M., Christian N. and Ali O. (2021). On preserving the behavior in software refactoring: A systematic mapping study. Information and Software Technology (2021), 106675

[19] Satrio A. R., Lina O. and Michel R.V. C. (2024) Deductive Software Architecture Recovery via Chain-of-thought Prompting, IEEE/ACM 46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSENIER)

[20] Link D., Pooyan B., Ramin M., and Boehm B. (2019). The Value of Software Architecture Recovery for Maintenance. In Proceedings of the 12th Innovations on Software Engineering Conference (Formerly Known as India Software Engineering Conference). Association for Computing Machinery, New York, Article 17, 10 pages. https://doi.org/10.1145/3299771.3299787

[21] Hamed J., Mohammad M. and Roozbeh R. (2025) Large Language Models (LLMs) for Source Code Analysis: applications, models and datasets, arXiv:2503.17502v1 [cs.SE] 21 Mar 2025., Woodstock, NY

[22] Shahla R., Mansour E., Abdolreza H., and Sepideh A. (2024) A Blockchain Approach to Extract Design Patterns From Source Codes, Wiley Modelling and Simulation in Engineering Volume 2024, Article ID 4992169, 15 pages https://doi.org/10.1155/2024/4992169

[23] Franzoni, V.; Tagliente, S.; Milani, A. (2024) Generative Models for Source Code: Fine-Tuning Techniques for Structured Pattern Learning. Technologies 2024, 12, 219.

[24] Youjia L. , Jianjun S., and Zheng Z. (2024) An Approach for Rapid Source Code Development Based on ChatGPT and Prompt Engineering, IEEE Access 12 53074 – 53087.

[25] Ahmad H., Christoph K., and Andreas R. (2025) Generating Software Architecture Description from Source Code using Reverse Engineering and Large Language Model, https://arxiv.org/abs/2511.05165v1