TECHNIUM

*Technium Vol. 4, Issue 4 pp.1-10 (2022)*
*ISSN: 2668-778X*
*www.techniumscience.com*

# Usage one of Reverse Engineering Application (Node.js Web) to Construct Software Engineering Documents

**Naktal Moaid Edan[1], Basim Mahmood[2]**

[1]Department of Software, College of Computer Science and Mathematics, University of Mosul, Mosul. Iraq.

[2]Department of Computer Science, College of Computer Science and Mathematics, Mosul, Mosul. Iraq.

[1]naktal.edan@uomosul.edu.iq, [2]bmahmood@uomosul.edu.iq

**Abstract**. In software engineering, documentation is considered the best way to distinguish the main functions of the software. The lack of software documentation may lead to consuming time and effort during the software design phase, which finally affects the quality of software. For instance, without proper documentation, the software maintainability index would never get a reasonable mark. The research has been progressive originally as a resilient concept, once the concept is proved to be developable in the Software Development Life Cycle (SDLC), the finishing product will be started to improve with satisfactory documentation. Also, software responsibility has caused complications, such as high costs of software repairs and inefficient process of knowledge transmission between developers. As a Software Reverse Engineering (SRE) method, this work uses a combination of an automatic class diagram creation tool with some manual tweaks, as well as source code analysis and comfortable interviews, to recover the product's design principles and requirement use cases. As a result, two documents will be produced: a Software Requirements Specification document and a Software Design Document.

**Keywords**: Software Reverse Engineering; Software Documentation; Software Maintainability.

## 1. Introduction

In software engineering, the documentation process is crucial insofar as it contributes to sustaining the software. The documentation process should be performed from the first start point of software development, then, the definite product will be settled following a series of reviews based on the collected documents. The field of software engineering introduces what is called Software Reverse Engineering (SRE). Figure (1), demonstrates the key difference between Software Reverse Engineering (SRE) and Software Engineering (SE). SRE tools, which are created and developed for specific contexts, support extracting the information required for software engineering documents. However, it is not always possible to recover or access all the information needed for software documentation using SRE tools.

This work is built and deployed aiming at increasing software maintainability (e.g., the documentation is reversed). For example, the descriptions of functions and classes must be made by

hand, it is necessary to study and analyze the current source code to comprehend the software and its features and elaborate on every single component. Nevertheless, emerging technologies, such as Node.js demand a series of reverse engineering tools, which will take time to be created, implemented and enhanced.

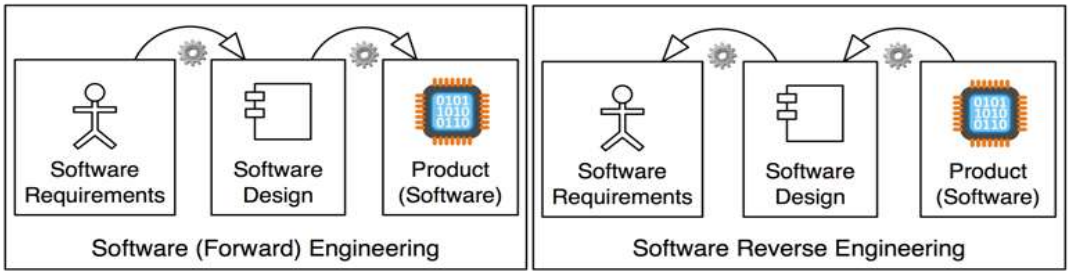



**Figure1: Software Engineering Vs Software Reverse Engineering**

The main purpose of this research is to create and develop Software Engineering documentation for the software server APIs. Moreover, one of the sub-disciplines of SRE, known as re-documentation has been considered to recover information from the software source code and other resources [1]. The software under consideration is a basic solution consisting of several modules and integrated components. Backend APIs as the chosen subsystem will be studied in four stages, as follows: arrangement, analysis, SRE performance, and development. Accordingly, in this work a collection of the necessary documents to achieve improvements and repair of the existing system is possible. This can be accomplished by employing SE principles and SRE techniques in the above processes, getting all potentially usable information from source code and other resources and organizing it in predefined document structures. To ensure that the study output is valid and useful for maintaining or upgrading the case study software. Thus, the documents will be created by existing documentation standards.

This paper is organized as follows: in section II, related works will be presented. Also, results and discussion will be explained in section III, and in section IV the conclusion has illustrated.

## 2. Related Works

This section reviews the literature on SRE, as well as the systematic approach employed in this subject area, and also the challenges that can be found when applying it. In addition, the background of software documentation in different forms will be extensively reviewed.

### *2.1. Software Reverse Engineering (SRE)*

SRE can be mentioned as an additional process, which guides other software engineering documents, for instance a high-level construction or a database illustration. Besides, SRE states the process of generating software engineering documents from the present software source code that is called re-documentation. In [1], described re-documentation as a review of a semantically corresponding illustration the longest with a similar relative thought level. Accordingly, the results of the demonstration are frequently considered by other opinions, such as data flow, data structure, and control flow for a human spectator. Moreover, they have specified the core determination of SRE for investigating the current system, instead of varying the substance system. Occasionally, the engineering exertions are concentrated on straight progress like the current software system and sort the software more reliable, maintainable, or flexible. Additionally, the SRE process goal is to enhance the software, and then it has been considered a reengineering effort and not a reverse engineering action.

The Reverse Engineering (RE) etymology is related to the military organizations obtaining

an understanding of their contestant's resources. Usage of RE leads to the major design and flow of invention process for tools which will be exposed and used in manufacture. A very common purpose of RE in the world of hardware is to duplicate the original object, while SRE is looking to better consider the design and maintenance, and improve the new software [2].

In [3], defined three dissimilar sets of methods for systematizing RE as the following: (a) Textual, (b) lexical, and (c) syntactic analysis to use UNIX's Lex and lexical metrics, Graphing methods for graphing the control flow of the program and so on. In contrast, another study demonstrated that 50-90 percent of works in the development stage are keen to program comprehension [4]. Besides, the authors in [5] deliberated the methods of RE as follows: Official transformations, meaning-preserving rearrangement, pattern recognition, purpose concept, and individualist identification, and reuse-oriented approaches. On the other hand, SRE has some difficulties, such as if the process going to be accomplished over a perfect system; in this situation, the whole software requires programming by the automated process. Hence, embedded software systems will be written in different languages. In [6], demonstrated a comparison of four RE tools, thus it was confirmed that missing some of the functions or variables in generated reports can be done as long as they used C and Assembler programming. Another issue it was encountered was that the case study was built for various perspectives that the RE tool was intended to emulate. Likewise, [3] provided a thorough definition of RE, so Re-documentation according to this definition, is the method of forming documentation for a software project (if it did not exist). The author goes on to list five instances where RE entails mapping in between to give a better understanding of the difficulties in SRE: application field and programming language, machines and programs, High-Level design, unique coherent, and hierarchical programs and knowledgeable suggestions.

Based on the above mentioned five areas, it can be concluded that SRE is a task that deeply depends on human collaboration and navigation. In [7] stated that the output documentation is either too generic or too complex to provide immediate responses to maintenance questions in either case.

### *2.2. Software Reverse Engineering Techniques*

The two frequently used SRE techniques are: (a) simple graph visualizations and (b) simple metrics surveys. Algorithms like the Sugiyama method [8], hyperbolic geometry [9], Libraries with a variety of algorithms [10], and Ternary diagrams support visual program representation. And, as measures that have been researched for a long time like [11]. There are two basic types of SRE for web apps: Static SRE only evaluates the present state of the items and aspects in the hierarchy of the Document Object Model (DOM) despite the events occurring on the web page and the modifications those occasions will implement on elements. dynamic SRE interacts with actions that affect a web page (DOM).

### *2.3. Software Reverse Engineering for Web Applications*

The authors in [12] provided an SRE approach and tool framework for obtaining Unified Modeling Language (UML) diagrams by both static and dynamic web applications. As a result, they have suggested extractor, a three-part tool that parses web pages using Hypertext Markup Language (HTML) on both the client and server sides and creates an intermediate representation of any usable data. The Repository is the second part, and it keeps the data that was retrieved during the first portion. The last component is the Abstractor, which performs a sequence of operations mostly on data information to improve the UML diagrams. Similarly, [13] planned a RE tool for web applications named Web App Viewer (WAVI) which uses static analysis and a filter-based mechanism to recover and document the construction of a web presence. Consequently, the suggested tool has a superior knowledge of JavaScript calls and detecting interrelationship links, as well as understandable and scalable visual modules. After testing the newest version of the tool, the results reveal that many more modifications are needed to increase the tool's accuracy. The findings are limited when it comes to

detecting variables and data structures. The diagrams produced are detailed, but if the topic system is large, the node relationships will be unclear and difficult to understand. [14], proposed WARE, a tool for Reverse Engineering Web Applications from their source code, in a study. The program is designed to RE dynamic websites, not just static HTML web pages, which makes their research noteworthy. WARE was created to semi-automatically recover information and documentation from web application source code. An extractor is the part of the product that analyzes the source code and is written in the C++ programming language. It parses HTML version 4.0 and supports the client-side scripting languages JavaScript and VBScript, as well as the server-side programming languages ASP and PHP. The parsed material will subsequently be represented as tag-described elements, which is known as the Intermediate Representation Form (IRF). The Abstractor will generate UML diagrams from the stored data.

### *2.4. Software Documentation*

The software development team should design and produce software artefacts systematically to have software based on a decent architecture, which is the key aspect of good software. Software architecture has become a highly crucial notion when it comes to the building of a successful complicated and massive system; according to [15] software architecture is maturing as a discipline in engineering. They continue to emphasise the necessity of effective software design documentation by emphasizing that if the software architecture is misunderstood, the generated product will never achieve its objectives. Another research conducted research in [16], focused on the automatic production of basic natural language and descriptions of complicated code artefacts; it was highlighting the need for external and internal documenting in most SE jobs. The documentation is frequently missing or obsolete, and Moreno proposes categorizing software components as a solution to this problem. Software documentation emerged as a significant component of the software engineering discipline. Software Development Plan (SDP), Interface Requirements Specification (IRS), Software Requirements Specification (SRS), Software Design Document (SDD), Software Test Plan (STP), and Software Test Design (STD) are just some types of software documentation that can be created to help people understand the software [17]. Not all projects require all possible documentation for software to be written; for example, if the product has no set of users to be accessible to certain other software or systems, the IRS will indeed be unnecessary. In contrast, [18] measured software documentation and found that artefacts such as requirements, design, and legal test documents are critical to fully comprehending the software. They underline that the documentation not just defines the software, but also covers the details of its construction and modifications and that it may be a valuable resource to consult during the knowledge transfer process.

## 3. Result and Discussion

### *3.1. Documentation Standards*

In the progression of producing the deliverables of this project SRS and SDD, the following list of documents has been referred to as standards and references: DOD-STD-2167A, DOD-STD-2168, Standard UML-2, Defense System Software Development, Defense System Software Quality Program, MIL-STD-1521B Technical Reviews, Thomson Software Development Reference System, The Unified Software Development, and Process.

### *3.2. The Process*

After initial research into the project and its available resources, which included technical documents and inline documents in source codes, it was determined that the product documentations are all non-scientific and in ad hoc format, and could not be used in the study's final deliverables. The following data was taken from the interview answers and the source code analysis stage and used in the production of the SRS and SDD documents. The SRE tool was also run over the source code to get an overview of the various classes and functions within the CSCI Backend API. Also, a flowchart of the project's stages and steps is described below. The tool chosen to work on the Backend API

subsystem's source code is a standalone application that can be installed and run via a terminal or command-line interface (CLI). The following command will install the tool as a standalone application after installing Node Package Manager (NPM) and Node.js on the test machine: (Install) $ npm–g.
Level 1: Planning ⟶ Initial research and Comfortable Meetings
Level 2: Analysis ⟶ Basis Code (Learning & Analysis) and Classify Controlled Situation
Level 3: Implement SRE ⟶ Implementation of SRE Tool
Level 4: Outcomes ⟶ Set up SDD and SRE physically

Once the tool has been installed on the local machine using NPM, run the following commands $ wavi app/example result/example.svg, while the program should be launched in the project root directory to traverse through all sub-directories. Some of the discovered entities in the diagram do not represent actual system classes, and the majority of the relationships are internal class constants or functions.

In the userController.js file, illustrates all of the existing functions. the shown functions have been collapsed using the IDE's collapse functionality to allow the reader to see all of the functions in a single file at once. Also, the following class diagram for the userController class has been created using the structure and functionalities of the files and directories, including the generated class diagram using the SRE tool. Figure (2), shows system outline.

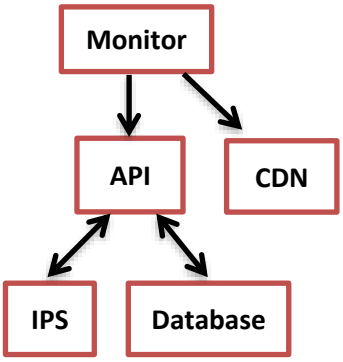


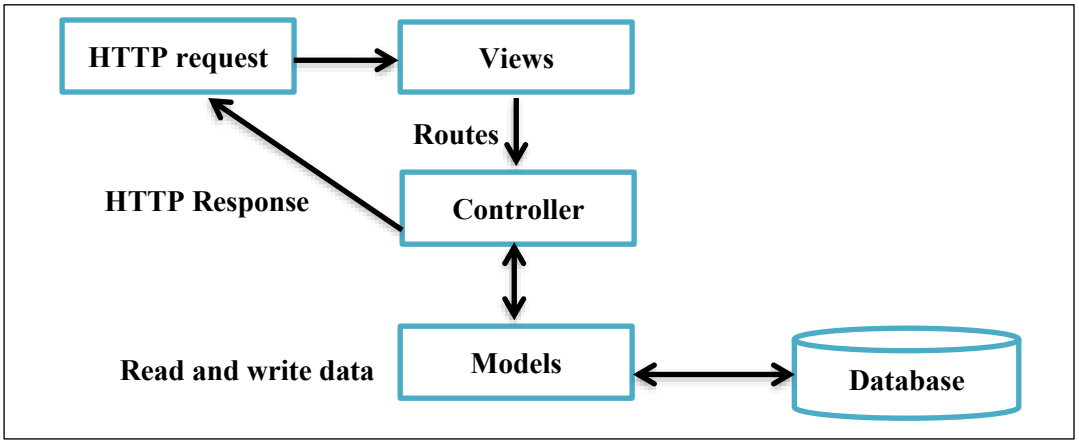

**Figure2. System outline**

### *3.3. System Overview*

The software under investigation in this project was created to give users with an innovative and user-friendly application for managing various digital signs (s). The existing software system is made up of a few subsystems that work together as a unified system to achieve the main goal. The system's major goal is to send planned and classified media to selected displays in the format of tailored content (stands). One can insert different media into predefined templates' defined sections, change different sections in templates, and then choose a certain date and time for the developed material to be published as a campaign using predefined templates. There are five main functional subsystems identified: Dashboard Subsystem, Backend API Subsystem, CDN Subsystem, IPS Subsystem, and Watchdog and Media Player Subsystem.

Dashboard Subsystem: the dashboard subsystem has a limited set of features. Campaign management including the scheduling of campaigns and the selection of information to be displayed, and the publication or launch of campaigns. Uploading or listing down media is part of media management (s). Management of the players' Management of playlists and templates. Dashboard functionality will be used for any driver or system admin contact with the system. The dashboard can be thought of as a system end-point or edge that communicates with the operator or end-user. Dashboard functions will also be used to log in or log out of the system. For example, a user may enter his credentials into a dashboard-provided form, after which the account will be sent to a

particular backend API, which will check the user account's validity against a third-party service and provide the outcome to the dashboard. The CSCI Backend API Subsystem is a computer software built and developed to serve incoming requests from various portions of the system. The following are the goals of the Backend API: (a) manage content, (b) manage campaigns, (c) manage schedules, (d) manage templates, and (e) manage users (login/logout). The Content Delivery Network (CDN) subsystem is the responsibility of the system's file formats. The CDN subsystem's stated responsibilities include photo resizing, video and image format swapping, and thumbnail generation from media. The backend API will use the dashboard to communicate with the CDN to control media files that will be downloaded to the player device and displayed by the player subsystem. The Watchdog Subsystem is a component of the system that is installed on the player's PC (stand). The player's interaction with the outside world will be handled by this subsystem. The initial step in the running watchdog application will be to get the ad file from the backend API. Once the watchdog has been installed and executed on the player machine, the backend API will be used to retrieve the campaign file for that specific player (stand). This promotional file is a plain text file that contains a JSON object. The Integrated Portal System (IPS) is a subsystem that keeps track of users and everything they have, including licenses, community groups, companies, and other user attributes. The Backend APIs can control user requests using the IPS interface. When a user logs in from the dashboard, for example, the Backend API verifies the user's identity by forwarding the user's account to IPS and checking the presence of the credentials. The IPS response would include the user's information as well as any other data available, such as authorization or responsibilities as shown in figure (3).




.

**Figure3: The Overview of the System**

### *3.4. Software Design Document*

All of the information about the subsystem's concept design is contained in the Software Design Document. The characteristics and functions of each object in the CSCI Backend API, and the duties of each component interface that links to others, are described below in figure (4):

- Micro-services are architecture in which several random and modular (standalone) services are grouped under one single application to provide the customer with needed data or action. Each business goal or use case in the system is handled by a single point of access, often known as an Application Program Interface (API). Because not all of the APIs in the software we're looking at in this project is individually deployable (standalone), the micro-service architecture as a well-known design isn't fully observed and implemented.
- The CSCI Backend API's architecture is quite similar to that of micro-service architecture.
- Backend API's data structures and controllers are split between two packages.

- The controller classes have no connection or relationship with one another. Data model classes are derived from a parent model class that comes from the Sails.js Waterline Node.js package.

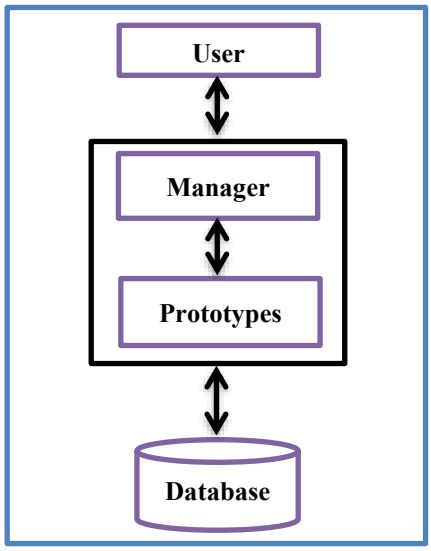



**Figure4. Design of API Subsystem**

### *3.5. Software Requirement Specification*

The SRS document will be created after the SDD has been completed. The use case diagrams and system specifics about the interaction between actors and the component are included in the Software Requirement Specification (SRS). Aside from the use cases, the SRS outlines the requirements for the interfaces' interaction. In the Backend API subsystem, five use situations have been discovered.

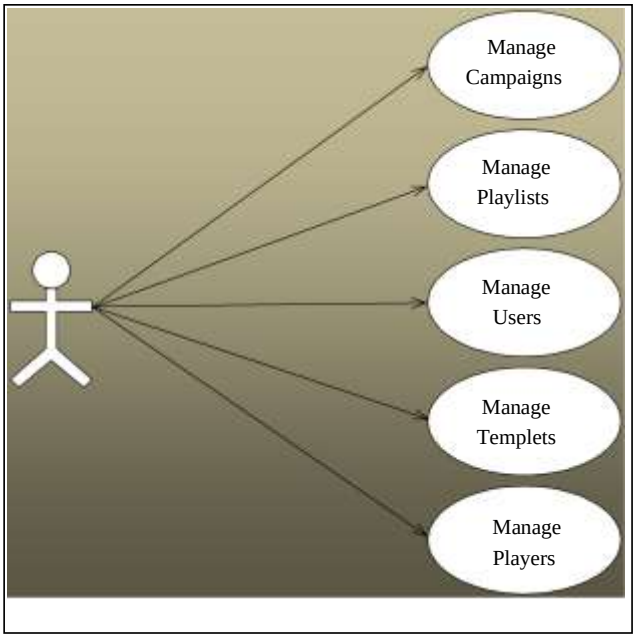



Figure5: Use Case Diagram

Figure (5), shows an activity diagram from the system developed for this investigation. It demonstrates

a simple action (create) in the context of the managing templates use case. Some procedures might be added to the system to increase the system's reliability or security, as shown in this figure. For example, once this end-point receives a request, there is no validation phase to ensure that the request has the necessary format or is coming from an approved system actor, which might result in significant system failure. This demonstrates a possible system flaw that was revealed by the use of created diagrams.

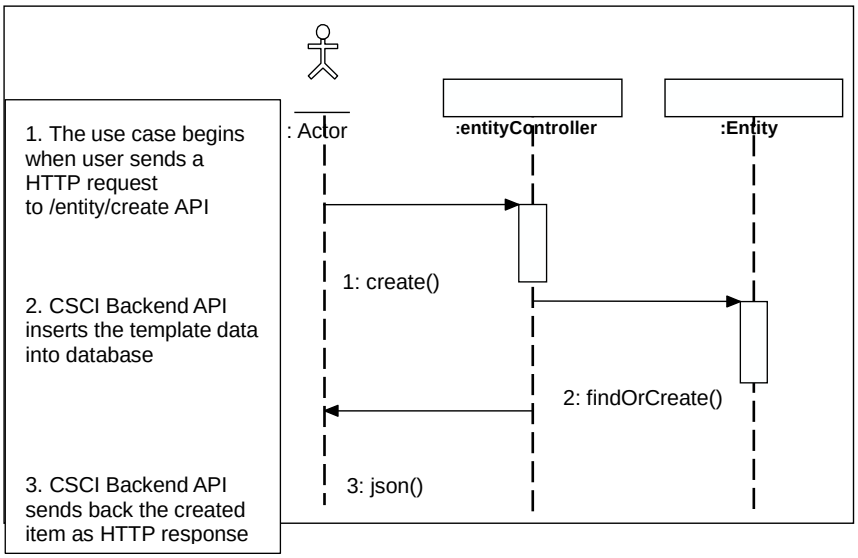



Figure6: A Sequence Diagram

The above-mentioned relationship across documents and classes in the system can be seen by comparing Figure (6), the userController class is devoid of any attributes.

An activity diagram from the commentators suggested for this research is shown in figure 6. It demonstrates a simple action (create) in the context of the managing templates use case. Some procedures might be included in the system to improve the system's reliability or security, as shown in this figure. For example, once this end-point receives a request, there is no authentication phase to ensure that the request has the necessary format or is coming from an allowed system actor, which might result in significant system failure. This demonstrates a possible system flaw that was revealed by the use of created diagrams.

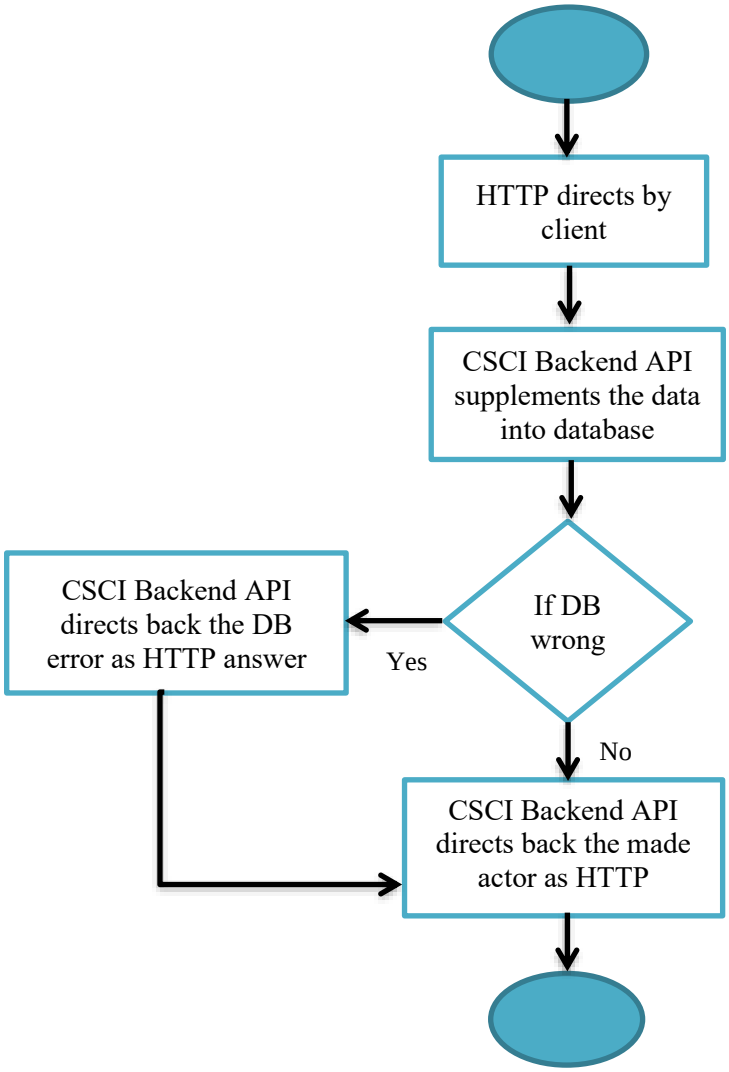



Figure7: An Activity Diagram

## 4. Conclusion

Although RE is not a new field that has been researched for decades in military and defence companies, SRE, particularly for online applications, is still in its infancy. Although the results of SRE and, in particular, re-documentation may not be as accurate as those obtained by documenting a software project first and before developing it, having documentation for a software project is required, and the use of SRE techniques aids in the process of documenting a developed software. Once the process of SRE and re-documentation is completed, many bottlenecks, flaws, and security risks will become apparent. The practice of rectification of recognized problems will be aided by documentation. This work introduced a framework for producing SE documents that included numerous stages or steps, which were then defined and pursued in detail to meet the study's objectives. This research has helped to improve the quality of developed software in a variety of ways and has produced useful deliverables that can be utilized in the reengineering process to improve the software takes an index and other essential aspects of good software.

**References**

[1] Elliot J. Chikofsky and James H Cross (1990). Reverse engineering and design recovery: A taxonomy. IEEE Software 7, 1, 13–17.
[2] C. Yung, K. Tai, S. Chong.(2018). Quality Evaluation of Structural Design in Software Reverse Engineering: A Focus On Cohesion, IEEE Access Journal, v (9), 5, 1-15.
[3] West, R., & Horvitz, E. (2019). Reverse-engineering satire, or "paper on computational humor accepted despite making serious advances". In Proceedings of the aaai conference on artificial intelligence Vol. 33, No. 01, pp. 7265-7272.
[4] Votipka, Daniel, et al. (2021). "An Investigation of Online Reverse Engineering Community Discussions in the Context of Ghidra." 2021 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE.
[5] Ted J Biggersta, Bharat G Mitbander, and Dallas Webster. (1993). The concept assignment problem in program understanding. In Software Engineering. Proceedings., 15th International Conference on. IEEE, 482–498.
[6] Berndt Bellay and Harald Gall. (1997). A comparison of four reverse engineering tools. In Reverse Engineering. Proceedings of the Fourth Working Conference on IEEE, 2–1.
[7] Gerardo Canfora, Aniello Cimitile, and Ugo De Carlini. (1992). A logic- based approach to reverse engineering tools production. IEEE Transactions on Software Engineering 18, 12, 1053–1064.
[8] Wessels, Michiel G., and Arthi Jayaraman. (2021) "Machine learning enhanced computational reverse engineering analysis for scattering experiments (crease) to determine structures in amphiphilic polymer solutions." ACS Polymers Au 1.3, 153-164.
[9] Zhou, F., Zhao, Y., Chen, W., Tan, Y., Xu, Y., Chen, Y & Zhao, Y. (2021) Reverse-engineering bar charts using neural networks. Journal of Visualization, 24(2), 419-435, 2021.
[10] Nyre-Yu, Megan, Karin Butler, and Cheryl Bolstad. (2022) "A Task Analysis of Static Binary Reverse Engineering for Security." In Proceedings of the 55th Hawaii International Conference on System Sciences.
[11] Nyre-Yu, Megan, Karin Butler, and Cheryl Bolstad. (2022) "A Task Analysis of Static Binary Reverse Engineering for Security." Proceedings of the 55th Hawaii International Conference on System Sciences.
[12] Giuseppe A Di Lucca, Massimiliano Di Penta, Giuliano Antoniol, and Gerardo Casazz. (2011). An approach for reverse engineering of web- based applications. In Reverse Engineering. Proceedings. Eighth Working Conference on. IEEE, 231–240.
[13] Jonathan Cloutier, Sègla Kpodjedo, and Ghizlane El Boussaidi. WAVI. (2016): A reverse engineering tool for web applications. In Program Comprehension (ICPC), IEEE 24th International Conference on. IEEE, 1–3.
[14] Zeeshan, Manaqib Ahmad, and Salman Sagheer Waris. (2021). "Reverse Engineering Application Instruments and Code Reliability: A Comparative Study of Tools." In Advances in Manufacturing Technology, pp. 53-63. IOS Press.
[15] dos Santos Júnior, Paulo Sérgio, Monalessa Perini Barcellos, and Fabiano Borges Ruy.(2021). "Tell me: Am I going to Heaven? A Diagnosis Instrument of Continuous Software Engineering Practices Adoption." In Evaluation and Assessment in Software Engineering, pp. 30-39. 2021.
[16] Marmolejos, Licelot, et al. "On the use of textual feature extraction techniques to support the automated detection of refactoring documentation. (2021). " Innovations in Systems and Software Engineering, 1-17.
[17] Ni, Xinghe. "Nuclear engineering software quality assurance. (2021). " In Nuclear Power Plant Design and Analysis Codes, pp. 55-74. Woodhead Publishing.
[18] Hutchinson, Ben, et al. (2021). "Towards accountability for machine learning datasets: Practices from software engineering and infrastructure." Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency.