COBOL RuleForge: Extracting Clarity from Complexity

Chiranjeevi B S - CS23M104
Department of Computer Science and Engineering
Indian Institute of Technology, Tirupati
Andhra Pradesh, India

April 22, 2024

Abstract

In software engineering, legacy system comprehension and modernization need a thorough grasp of and application of business rules that are embedded in legacy codebases. This work offers a comprehensive approach for extracting business rules out of legacy COBOL code, summarizing them, and comparing them with conventional code summaries. The proposed work utilizes Large Language Models (LLMs) to extract and summarize Business Rules from a COBOL code, and then use this summary to compare against the conventional code summary. Our experiment highlights how crucial it is to isolate and fully comprehend business rules within legacy systems by comparing these summaries. This methodology attempts to streamline modernization tasks, aims towards a better understanding of the system, and emphasizes the importance of extracting business rules for software development and maintenance.

1 Introduction

In the vast landscape of software engineering, the evolution of legacy systems presents both challenges and opportunities. Among the myriad tasks involved in modernizing and maintaining such systems, decoding and effectively utilizing business rules embedded within their codebases stands out as a critical one [AMC22]. Traditional codebases, particularly those written in COBOL, often harbor a wealth of business rules mixed with procedural instructions, making it difficult for developers to identify and manipulate these rules efficiently.

The need to extract business rules from COBOL code arises from several pressing imperatives. Firstly, as organizations strive to adapt to ever-evolving market demands and regulatory requirements, the ability to quickly comprehend and modify business rules becomes necessary. However, in the absence of clear documentation or structured representations, extracting these rules from the codebase proves to be a difficult task.

Moreover, conventional code summarization techniques fall short in capturing the essence of business rules. While code summaries may elucidate the structure and functionality of a system, they often fail to extract and clearly outline the underlying business logic that dictates its behavior. Business rules encapsulate the core principles and decision-making criteria driving system behavior, transcending mere procedural instructions. Thus, a comprehensive understanding of a system necessitates the explicit extraction and representation of its business rules.

In this context, the approach proposed in this project seeks to bridge this gap by leveraging Large Language Models (LLMs) to extract and summarize business rules from COBOL code. By harnessing the capabilities of LLMs, the project endeavors to unravel the intricacies of business logic embedded within COBOL systems. Subsequently, it compares these business rule summaries with conventional code summaries, sheddng light on the stark disparity between the two and emphasizing the critical importance of extracting business rules for system comprehension and modernization efforts.

In summary, this project addresses a pressing need in the realm of software engineering by providing a LLM-based approach to extract, summarize, and compare business rules from COBOL code. By elucidating the distinct nature of business rules and their significance in system evolution, it lays the groundwork for enhancing the maintainability and adaptability of legacy systems in today's dynamic business landscape.

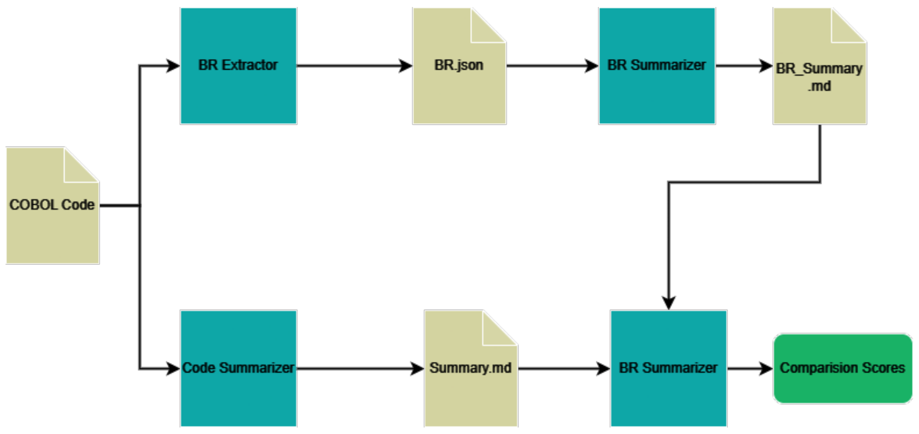


Figure 1: Proposed Workflow

2 Problem Statement

The problem at hand revolves around the need to develop an automated approach to extract, summarize, and compare business rules from COBOL code, distinct from traditional code summarization methods. This approach must harness Large Language Models (LLMs) to unravel the implicit business logic embedded within COBOL systems and generate concise summaries that highlight the essence of the business rules. Furthermore, it must facilitate the comparison of these business rule summaries with conventional code summaries, emphasizing the importance of explicitly extracting and understanding business rules for efficient system modernization and maintenance.

Addressing this problem requires a comprehensive understanding of both COBOL programming constructs and business domain concepts, coupled with proficiency in advanced Natural Language Processing (NLP) techniques. The solution must integrate seamlessly into existing software engineering workflows, providing developers with actionable insights into the business logic encapsulated within legacy COBOL systems.

3 Methodology

For our experiments, a total of 12 COBOL codes from the X-COBOL dataset [AMC23] were chosen as the test set. This is because our evaluation method involves a lot of manual observations, and analysing hundreds of COBOL codes manually is simply not possible.

3.1 Workflow

Our workflow has been summarized (no pun intended) in Fig. 1. For every COBOL file in our testing set, we first use LLM few-shot prompting to extract Business Rules from it and store it in json format. We also use the LLM to summarize the code and store the summary in markdown format (for human-friendly interpretation). Next, we use one-shot prompting to summarize the Business Rules extracted in the previous phase and store this summary in markdown format as well. Finally, we compare the corresponding markdown files of both the summaries and calculate BLEU ROUGE scores on them. Manual analysis is also performed on the generated BR summaries and the general code summaries to figure out the need to extract business rules.

3.2 Technology and Libraries used

The LLM used for extracting and summarizing business rules is Google's Gemini-Pro. We make use of google's `generativeai` library to make the API call to the invoke an instance of the model and infer from it. We have used few-shot and one-shot prompting techniques to generate content from the model. Before providing raw COBOL input to the model, we first preprocess all test files to exclude comment lines and trim spaces to reduce input token size. This has to be done since one-shot prompting is also input to our model. We cannot include a large COBOL file as part of the prompt. We make use of the `glob` library to essentially iterate through the list of files in a directory. Its a useful tool for automation of file content extraction. We have also used the `evaluation` library to evaluate generated content using the BLEU ROUGE scores.

The entire project is written in Python (version 3.10.8). We tested the validity of the BR extractor and the code summarizer in `Jupyter Notebook` before coding it in a `.py` file. This helped us in two aspects: unit testing and easy debugging.

3.3 Project Structure

In this subsection, we will provide a concise overview of the project structure and the role that each file in our project play. There are 4 main files, which deal with the processing part of the workflow described in 3.1. All the test files, along with their extracted BRs and summarized markdown files are in the `Test_COBOLs/InputFiles` folder. We shall first understand the role that the processing files play:

1. `BR_generator.ipynb`: This Notebook file is used to extract the BRs from input COBOL file and store their JSON outputs in `Test_COBOLs/BR_JSON`.
2. `CodeSummarizer.py`: This python file is used to summarize the content of a COBOL file directly, and store the markdown output in `Test_COBOLs/Output_Code_Summary`.
3. `BRSummarizer.py`: This python file is used to summarize the BRs extracted from the COBOL files, and store the markdown output in `Test_COBOLs/Output_BR_Summary`.
4. `CompareSummaries.ipynb`: This Notebook file is the evaluator that we use to compare the outputs of `CodeSummarizer.py` and `BRSummarizer.py`. Results can be viewed while running the Notebook file.

Files like `codeSummarizationMessage.txt` and `BRSummarizerMessage.txt` store few-shot prompting information that we feed as input to our model.

3.4 Running the Project

First, you will need to store your COBOL files in the folder `Test_COBOLs/InputFiles`. After making sure that all files are placed in the folder, follow the steps given below:

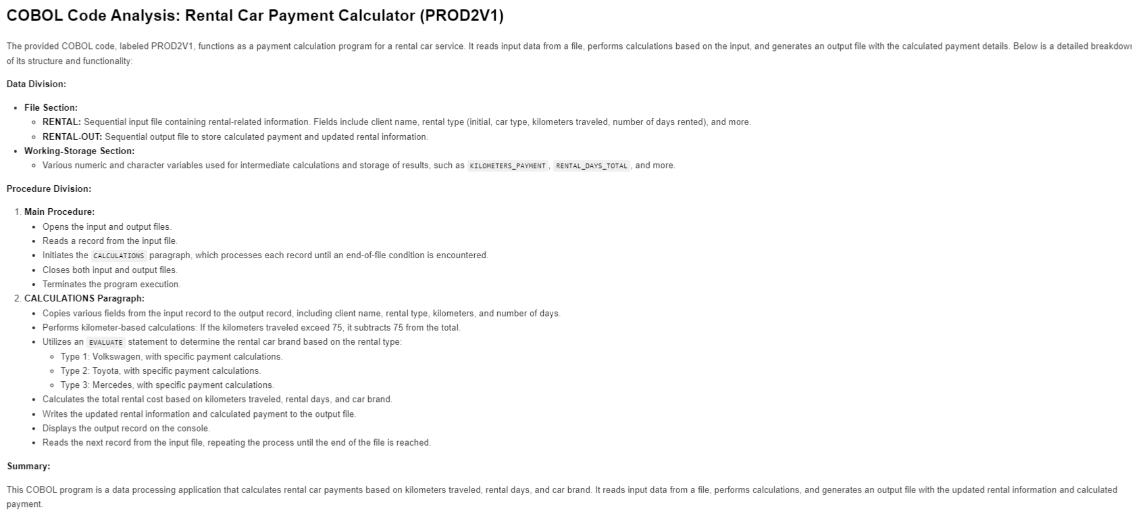
1. Execute the `preprocess.py` file by running the command `python preprocess.py`. This will remove comment lines and extra spaces in your COBOL files.
2. Run the `BR_generator.ipynb` file completely. Make sure you have the `codeSystemMessage.txt` file before you run the Notebook.
3. Run the `Summarizer.sh` script file by executing the command `./Summarizer.sh`, to produce code summaries and BR summaries. Be sure to grant execute access to the script file before you execute it.
4. Finally, to evaluate the results, run the `CompareSummaries.ipynb` file to get the BLEU ROUGE scores of the generated summaries.

4 Results

In this section, we will present the results of our experiments, both manual and automated. For the manual evaluation, we have inspected the markdown files produced by `CodeSummarizer.py` and `BRSummarizer.py`, and analysed the data based on availability, completeness and clarity of the Business Rules inside each summary. For the automated evaluation, we have used the BLEU ROGUE score, a popular metric for NLP evaluation, to compare the normal code summary to the generated BR summary.

4.1 Manual Evaluation

Conventional code summaries and BR summaries were manually inspected to assess the quality of Business Rules inside each of them. Here's what we found: Although conventional code summaries describe the overall structure and working of the code, they do not highlight the rules that are embedded inside each file. More often than not, it is these rules that define the purpose of the COBOL code and help in migration of the system. On the other hand, the BR summaries highlight



COBOL Code Analysis: Rental Car Payment Calculator (PROD2V1)

The provided COBOL code, labeled PROD2V1, functions as a payment calculation program for a rental car service. It reads input data from a file, performs calculations based on the input, and generates an output file with the calculated payment details. Below is a detailed breakdow of its structure and functionality:

Data Division:

- **File Section:**
  - **RENTAL:** Sequential input file containing rental-related information. Fields include client name, rental type (initial, car type, kilometers traveled, number of days rented), and more.
  - **RENTAL-OUT:** Sequential output file to store calculated payment and updated rental information.
- **Working-Storage Section:**
  - Various numeric and character variables used for intermediate calculations and storage of results, such as `KILOMETERS_PAYMENT`, `RENTAL_DAYS_TOTAL`, and more.

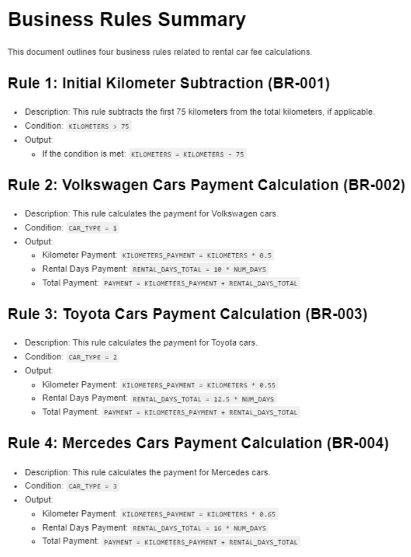
Procedure Division:

1. **Main Procedure:**
   - Opens the input and output files.
   - Reads a record from the input file.
   - Initiates the `CALCULATIONS` paragraph, which processes each record until an end-of-file condition is encountered.
   - Closes both input and output files.
   - Terminates the program execution.
2. **CALCULATIONS Paragraph:**
   - Copies various fields from the input record to the output record, including client name, rental type, kilometers, and number of days.
   - Performs kilometer-based calculations: If the kilometers traveled exceed 75, it subtracts 75 from the total.
   - Utilizes an `EVALUATE` statement to determine the rental car brand based on the rental type:
     - Type 1: Volkswagen, with specific payment calculations.
     - Type 2: Toyota, with specific payment calculations.
     - Type 3: Mercedes, with specific payment calculations.
   - Calculates the total rental cost based on kilometers traveled, rental days, and car brand.
   - Writes the updated rental information and calculated payment to the output file.
   - Displays the output record on the console.
   - Reads the next record from the input file, repeating the process until the end of the file is reached.

Summary:

This COBOL program is a data processing application that calculates rental car payments based on kilometers traveled, rental days, and car brand. It reads input data from a file, performs calculations, and generates an output file with the updated rental information and calculated payment.

Figure 2: An example of Code Summary generated for CarRental.cbl file



Business Rules Summary

This document outlines four business rules related to rental car fee calculations.

Rule 1: Initial Kilometer Subtraction (BR-001)

- Description: This rule subtracts the first 75 kilometers from the total kilometers, if applicable.
- Condition: `KILOMETERS > 75`
- Output:
  - If the condition is met: `KILOMETERS = KILOMETERS - 75`

Rule 2: Volkswagen Cars Payment Calculation (BR-002)

- Description: This rule calculates the payment for Volkswagen cars.
- Condition: `CAR_TYPE = 1`
- Output:
  - Kilometer Payment: `KILOMETERS_PAYMENT = KILOMETERS * 0.5`
  - Rental Days Payment: `RENTAL_DAYS_TOTAL = 10 * NUM_DAYS`
  - Total Payment: `PAYMENT = KILOMETERS_PAYMENT + RENTAL_DAYS_TOTAL`

Rule 3: Toyota Cars Payment Calculation (BR-003)

- Description: This rule calculates the payment for Toyota cars.
- Condition: `CAR_TYPE = 2`
- Output:
  - Kilometer Payment: `KILOMETERS_PAYMENT = KILOMETERS * 0.55`
  - Rental Days Payment: `RENTAL_DAYS_TOTAL = 12.5 * NUM_DAYS`
  - Total Payment: `PAYMENT = KILOMETERS_PAYMENT + RENTAL_DAYS_TOTAL`

Rule 4: Mercedes Cars Payment Calculation (BR-004)

- Description: This rule calculates the payment for Mercedes cars.
- Condition: `CAR_TYPE = 3`
- Output:
  - Kilometer Payment: `KILOMETERS_PAYMENT = KILOMETERS * 0.65`
  - Rental Days Payment: `RENTAL_DAYS_TOTAL = 16 * NUM_DAYS`
  - Total Payment: `PAYMENT = KILOMETERS_PAYMENT + RENTAL_DAYS_TOTAL`

Figure 3: An example of BR Summary generated for CarRental.cbl file

most (if not all) of the rules explicitly, making it easier for the reader to understand the business logic thoroughly.

Consider the example given in Fig. 2. This was the code summary generated for the COBOL file `CarRental.cbl`. Clearly, the summary gives the overall structure of the code and its working, but does not explicitly mention the rules that are embedded in the code. On the contrast, consider the example given in Fig. 3. Here, all rules of the COBOL files are explicitly described, especially the rule about different car types and their rental fee, which was missing in the code summary. This shows that a conventional code summary may not be enough for tasks that involve extracting and analyzing Business Rules. We will have to extract these Business Rules and then proceed with further steps.

4.2 Automated Evaluation

For the purposes of automated evaluation, we used BLEU ROUGE scores [Ble], a popular metric for NLP evaluation. We first extracted all file content and stored them in a dictionary: key corresponds to the file name and the value is a list of strings, each string corresponding to a line in the generated Markdown file. For each COBOL file in our test set, the conventional code summaries are compared against the generated BR summaries. Results are shown in Table. **??**. Ideally, both the values of BLEU and ROUGE should be close to 1. The results show that these

Filename	BLEU score	ROUGE1 score
Car_Rental	0.0	0.07570
ClassCondition	0.0	0.07549
cobol-calc	0.0	0.04719
ConditionNameCondition	0.09224	0.13442
CWBWDATE	0.0	0.06109
FIZZBUZZ	0.0	0.08202
GETLOAN	0.0	0.06681
player-turn	0.0	0.06768
PROGCOB07	0.0	0.04762
PROGCOB08	0.0	0.03081
shop	0.0	0.05364
SignCondition	0.0	0.04149
Avg:	0.00769	0.06557

Table 1: BLEU and ROUGE scores on generated summaries

two are entirely different, and therefore differ in the information that they provide too.

References

[AMC22] Mir Sameed Ali, Nikhil Manjunath, and Sridhar Chimalakonda. Cobrex: A tool for extracting business rules from cobol. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 464–468, 2022.

[AMC23] Mir Sameed Ali, Nikhil Manjunath, and Sridhar Chimalakonda. X-cobol: A dataset of cobol repositories, 2023.

[Ble] Understanding bleu and rouge score for nlp evaluation. https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb.