TOOL DEVOLPMENT SOFTWARE ENGINEERING - UG3 9

https://github.com/cs22b035raghu/ToolSE

Requirement-Code Traceability with Chat – Final Project Report

ABSTRACT

This project presents a comprehensive web-based Module Analyzer integrated with an AI-driven Chat interface, designed to streamline the process of understanding and maintaining complex software systems. The tool focuses on analyzing GitHub repositories, providing intelligent insights into the structure and functionality of the codebase, while offering real-time, interactive support to developers and stakeholders.

One of the core capabilities of the tool is its ability to compare implementation code against requirement documents, enabling automated traceability analysis. This ensures that the developed modules align closely with specified requirements, helping identify missing features, inconsistencies, or deviations early in the development lifecycle.

The integrated chat feature acts as a smart assistant, allowing users to ask questions about the codebase in natural language. Backed by advanced summarization and suggestion models, the system can generate concise module overviews, explain code functionality, and offer recommendations for improvements or refactoring. This AI-driven support significantly enhances software comprehension, especially for new developers or during code reviews.

Key benefits of the Module Analyzer include:

Enhanced traceability between requirements and implementation.

Accelerated onboarding for developers through code summarization.

Efficient code exploration using natural language queries.

Improved collaboration among development teams via shared insights.

Increased development productivity by reducing manual documentation and analysis efforts.

By combining intelligent code analysis with conversational AI, this tool serves as a powerful assistant for software engineers, quality assurance teams, and project managers, ultimately improving software quality, maintainability, and development efficiency.

KEYWORDS

Code analysis, requirement traceability, GitHub, chatbot, summarization, Flask

ACM Reference Format:

TOOL DEVOLPMENT SOFTWARE ENGINEERING - UG3 9. 2025. Requirement-Code Traceability with Chat – Final Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software traceability is essential for ensuring that each requirement is accurately implemented within the codebase. Our tool addresses this challenge by automatically analyzing source code from GitHub repositories and mapping it to structured requirement documents in PDF format. By establishing clear links between requirements and implementation, it facilitates early detection of gaps, inconsistencies, or missing features.

To enhance usability, the tool features an integrated AI-powered chatbot that allows users to interact with the codebase through natural language queries. Complemented by intelligent summarization capabilities, the chatbot provides concise explanations of code modules, improving accessibility and accelerating onboarding.

This combined approach supports effective validation, documentation, and comprehension of complex software systems—making it an invaluable asset for developers, project managers, and quality assurance teams.

2 CORE FEATURES

1. GitHub Repository Analyzer

- Accepts a GitHub repository URL with an optional module name for focused analysis.
- Automatically clones the repository or pulls the latest changes if it already exists locally.
- Scans and parses source files (`.py`, `.js`, `.java`, `.cpp`, `.ts`) to extract functions, methods, and associated comments.
- Applies regular expressions to filter and isolate code segments matching the specified module name.

2. PDF Requirement Comparison

- **Extracts and parses requirement subheadings** from PDF documents: The tool efficiently identifies and extracts the key subheadings or sections of requirements from a provided PDF, enabling easy comparison and analysis.
- **Compares requirements against code using string similarity and content analysis**: The system employs advanced string matching techniques and semantic content checks to determine whether the requirements in the PDF are reflected in the code. This ensures that all requirements are thoroughly assessed for coverage.
- **Identifies and highlights covered and uncovered requirements**: Once the comparison is complete, the tool provides a clear visualization by marking which requirements are fully implemented in the code and which ones are missing or not addressed. This helps in quickly identifying gaps in compliance or functionality.

3. AI-Powered Chatbot

- **Seamlessly integrates with OpenRouter (Mistral model)**: The chatbot leverages the OpenRouter platform, utilizing the Mistral model for advanced natural language processing, enabling it to deliver context-aware and accurate responses.

- **Provides real-time answers to developer queries on functionality and design**: The AI chatbot is equipped to address a wide range of questions related to the system's functionality, architecture, and design choices, offering developers instant, insightful feedback.
- **Explains code structure and facilitates navigation**: Beyond answering questions, the chatbot also assists developers in understanding complex code structures, guiding them through different sections of the codebase, and helping them efficiently navigate the project.

4. Code Summarizer

- **Leverages `facebook/bart-large-cnn` model through HuggingFace**: The Code Summarizer uses the powerful model, accessed via the HuggingFace platform, to generate high-quality summaries based on deep learning techniques.
- **Generates concise and informative summaries for modules and large code files**: The tool efficiently condenses extensive code files or individual modules into clear, brief summaries, making it easier for developers to understand code logic, structure, and key functionalities at a glance.

5. Suggestion Generator

- **Provides recommendations for code refactoring and improving readability**: The Suggestion Generator analyzes code and offers actionable suggestions for refactoring, making it cleaner, more modular, and easier to read and maintain.
- **Recommends Python best practices and optimization techniques**: It also identifies opportunities for improving the code's efficiency and performance, suggesting best practices for Python development, as well as optimization strategies to ensure faster execution and better resource utilization.

3 NEW FEATURES

1. Code Issue Detection

- **Detects common code smells**: The tool thoroughly scans the code to identify common "code smells," which are indicative of suboptimal coding practices. This includes detecting deep nesting, which can make the code harder to follow and increase the likelihood of bugs. It also flags poor variable naming, which can confuse developers and reduce the code's readability, and unused variables that clutter the code and waste memory. By identifying these issues early, the tool helps ensure the code remains clean, maintainable, and easier to debug.
- **Identifies missing docstrings and large functions**: The tool checks for missing docstrings, which are essential for documenting the purpose and functionality of functions, classes, and modules. Without adequate docstrings, understanding and maintaining the code becomes significantly harder, especially for new developers joining the project. Additionally, it identifies functions that are too large, which often violate the principle of single responsibility and can be a sign of poor design. Large functions tend to be harder to understand, test, and modify. By pinpointing these issues, the tool encourages writing smaller, more focused functions and ensuring that the code is well-documented.
- **Helps improve code maintainability and readability**: By detecting these issues, the tool plays a critical role in maintaining code quality over time. Clean code with proper naming conventions, reduced complexity, and adequate documentation is easier for teams to maintain and extend. The tool serves as a preventative measure, allowing developers to address potential problems before they evolve into more serious and costly issues.
- **Enhances collaboration and onboarding**: Missing docstrings and unclear code are significant barriers to effective collaboration, particularly in larger teams. The tool helps ensure that code is easily understandable, even for new developers or external contributors. Well-documented and clean code speeds up the onboarding process and fosters better collaboration across the development team.
- **Promotes adherence to coding standards**: By detecting code smells and missing documentation, the tool encourages adherence to established coding standards and best practices. This not only improves code quality but also ensures consistency across the project, making it easier to manage and scale as the project grows.

2. Requirement Coverage Checker

- **Extracts subheadings from PDF requirements**: The tool automatically extracts subheadings from the PDF document containing project requirements. This process involves parsing the document and identifying the key requirements and specifications listed, allowing for efficient extraction and analysis without manual effort.
- **Compares extracted subheadings to code implementation using fuzzy matching**: Once the requirements are extracted, the tool compares them to the codebase to check if each requirement has been implemented. The comparison uses fuzzy matching techniques, which ensure that even slight variations in phrasing or terminology between the requirement document and the code don't result in false negatives. This enables a more accurate, flexible matching process, accounting for different ways a requirement might be expressed in the code.
- **Generates a traceability matrix with ✓ and ×**: After the comparison, the tool generates a traceability matrix that clearly indicates whether each requirement is covered by the code. The matrix visually represents the relationship between the requirements and the code, with ✓(check) symbols showing that a requirement is successfully implemented and ×(cross) symbols indicating uncovered or missing requirements. This matrix serves as a quick, visual reference for stakeholders to understand the level of coverage and identify any gaps in the code's functionality.
- **Enhances requirement validation and project tracking**: The traceability matrix aids in validating whether all the requirements have been met, ensuring that no critical functionality has been overlooked. It also provides a valuable tool for project tracking, enabling teams to monitor progress

and ensure that the development aligns with the project's defined objectives.

- **Improves compliance and reporting**: By providing a clear mapping between requirements and code implementation, the tool helps ensure compliance with project specifications and facilitates reporting to clients or stakeholders. The traceability matrix offers an easy-to-read overview of how well the code meets the defined requirements, streamlining the review and audit process.
- **Facilitates gap analysis and future improvements**: If gaps are identified (i.e., requirements that are not covered by the code), the tool helps teams quickly pinpoint areas for improvement or further development. This promotes a proactive approach to ensuring full requirement coverage and the delivery of a complete product.

4 HOW TO RUN THE APPLICATION

Folder Structure

```
project/
 app.py
 templates/
  index.html
 static/
  style.css.
 requirements.txt
```

- **app.py**: This is the main entry point of the project. It typically contains the core application logic, such as routing, handling user input, and interacting with databases or APIs. In a Flask or similar web framework, this would be where the routes and views are defined.
- **templates/**: This directory holds the HTML templates for the project. These templates are used by the application to render dynamic content. For example, index.html might be used to generate the home page of the application, where content is displayed to the user.
- **static/**: This folder contains static files like stylesheets, JavaScript files, and images. These files are served directly by the web server and are not processed by the application. For instance, style.css is used to style the HTML content, making the website visually appealing.
- **requirements.txt**: This file lists all the Python packages and dependencies required to run the project. It is typically generated by running pip freeze and is used to set up the project environment, ensuring that the correct versions of libraries are installed when the project is deployed or set up by another developer.

Execution Steps

(1) **Clone the repository**:
- Open your terminal or command prompt and run the following command to clone the repository from your version control platform (e.g., GitHub, GitLab):
- `git clone <your-repo-url>`
- Replace `<your-repo-url>` with the actual URL of the repository you wish to clone.

(2) **Install dependencies**:
- Navigate to the project directory where the repository has been cloned:
- `cd <project-directory>`
- Install all required Python dependencies listed in `requirements.txt` by running the following command:
- `pip install -r requirements.txt`
- This will ensure that all the necessary libraries and packages are installed, including frameworks and other dependencies needed to run the application.

(3) **Add your OpenRouter API key to `app.py`**:
- Open the `app.py` file in your preferred code editor.
- Locate the section where the OpenRouter API key is required (usually marked with a comment or placeholder).
- Insert your valid API key in the appropriate place. If you don't have an API key, follow the provider's instructions to obtain one.

(4) **Start the application**:
- In your terminal, from the project directory, start the application by running the following command:
- `python app.py`
- This will start the Flask or similar web framework application and run it locally on your machine.

(5) **Access the app in your browser**:
- Open your web browser and navigate to the following URL to access the running application:
- `http://127.0.0.1:5000`
- You should now see the application running on your local machine. If you encounter any issues, check your terminal for error messages.

5 DEPENDENCIES

The following packages are listed in the `requirements.txt` file, which defines the necessary dependencies to run the project:

```
Flask
transformers
pdfminer.six
requests
openai
huggingface_hub
torch
PyPDF2
```

- **Flask**: Flask is a lightweight web framework for Python, ideal for building web applications and APIs. It provides the essential tools for routing requests, handling HTTP methods, and rendering templates, making it a popular choice for small to medium-scale web projects.
- **transformers**: The transformers library by Hugging Face provides access to state-of-the-art pre-trained models for natural language processing (NLP) tasks such as text generation, summarization, translation, and more. This package is essential for leveraging models like BERT, GPT, and T5 in your project.
- **pdfminer.six**: pdfminer.six is a Python package for extracting text, images, and metadata from PDF files. It is used in this project to parse and extract requirement subheadings and content from PDFs, allowing you to perform comparison and analysis with the codebase.

- **requests**: The requests library is a simple and powerful HTTP library for Python, used to make HTTP requests such as GET, POST, PUT, DELETE, and more. It is often used for interacting with REST APIs, web scraping, and sending data between applications.
- **openai**: This package allows you to interact with the OpenAI API. It provides the necessary functionality to integrate OpenAI's powerful language models (like GPT) into the application, enabling features such as chatbot responses, text generation, and more.
- **torch**: PyTorch is a deep learning framework that provides powerful tools for building and training machine learning models. It supports both CPU and GPU acceleration and is widely used for implementing neural networks, including those used in NLP tasks with the transformers library.
- **PyPDF2**: PyPDF2 is another library for working with PDF files in Python. It allows you to read, extract, merge, and manipulate PDF content, complementing pdfminer.six in tasks related to handling PDFs, such as extracting specific data or performing simple manipulations.

6 TEAM CONTRIBUTIONS

- **Raghuram:** Developed the Flask backend, implemented Git operations, and designed suggestion/issue modules.
- **Nayak:** Focused on static analysis and code evaluation.
- **Raghuveer:** Created summarization logic using Python AST.
- **Sampath:** Designed the front-end and visual traceability mapping.
- **Aseem:** Conducted validation and tested requirement coverage.

7 CONCLUSION

This project showcases an innovative integration of AI-powered tools for code analysis and requirement traceability. By leveraging advanced AI techniques, the project enhances the understanding of software code and its alignment with specified requirements. Key features include:

Summarization: It automatically generates concise summaries of large code files and modules, allowing developers to quickly understand the structure and functionality of the code without diving deep into every line.

Requirement Matching: Using AI, the project compares requirements extracted from PDF documents with the implemented code, providing traceability and highlighting covered and uncovered requirements. This ensures that no functional requirements are missed during development.

Chatbot Integration: The inclusion of an AI-powered chatbot enables developers to ask questions about the code's design, functionality, and structure, helping them navigate and understand the code more efficiently.

Together, these features contribute to improving software comprehension, reducing the chances of overlooked requirements, and ultimately improving code quality.

Future Directions:

Looking ahead, the project aims to evolve in several key areas:

Semantic Traceability: Moving beyond simple string matching, future versions will incorporate semantic analysis to better understand the intent behind the requirements and code. This would allow for more accurate and context-aware requirement matching.

Improved Visualization: The traceability matrix and requirement matching features will be enhanced with more sophisticated visualizations, providing users with a clearer, more intuitive way to understand which requirements are covered by the code and where gaps exist.

Broader Language Support: Currently, the project is tailored to specific programming languages and document structures. Future versions will expand language support, making the tool accessible to a wider range of development environments and coding languages.

8 FUTURE IMPROVEMENTS

1. Enhanced Requirement Extraction Current Limitation: The current version extracts subheadings and compares them to the code. However, the level of granularity for requirement extraction may be limited.

Future Improvement: Incorporate Natural Language Processing (NLP) techniques for semantic parsing of the entire document, which would allow for extracting more detailed requirements from complex documents. This could include more sophisticated identification of functional and non-functional requirements, use cases, and constraints.

2. Deep Semantic Code Understanding Current Limitation: The requirement matching is based on basic string similarity and some content checks.

Future Improvement: Leverage deep learning models to understand the semantic meaning of the code. This would go beyond syntactic matching to accurately understand the logic of the code and how it fulfills specific requirements. For instance, integrating Graph Neural Networks (GNNs) to represent code as graphs could enhance the understanding of dependencies and connections within the code.

3. Integration of Test Case Generation Current Limitation: The tool currently focuses mainly on requirement traceability and code summarization.

Future Improvement: Integrate automated test case generation from requirements. Based on the traceability matrix, the system could suggest relevant test cases, ensuring that the code implementation correctly fulfills the requirements. This would streamline testing efforts and improve the overall development lifecycle.

4. Better Visualization Tools Current Limitation: Visualizations are relatively basic, potentially limiting developers' ability to easily identify gaps or areas of concern.

Future Improvement: Develop more interactive and dynamic visualizations for the traceability matrix, requirement coverage, and code structures. This could include heatmaps for requirement coverage, flowcharts for understanding code dependencies, or interactive graphs for exploring code and requirement relationships.

5. Real-time Code Analysis and Feedback Current Limitation: Code analysis is likely performed in batch processes.

Future Improvement: Implement real-time feedback and suggestions for developers while they are writing or modifying the code. This could include live requirement checking, code quality analysis,

and suggestion generation as the developer types, helping catch issues earlier in the development process.

6. Multi-Language and Multi-Framework Support Current Limitation: The tool is currently tailored to specific programming languages and frameworks.

Future Improvement: Expand support for multiple programming languages and frameworks to make the tool more universally applicable. This would involve adding language-specific parsing and analysis modules for languages like Java, C++, Ruby, or JavaScript, as well as supporting different web and application frameworks.

7. Enhanced Chatbot Features Current Limitation: The chatbot primarily serves to answer basic questions about the code.

Future Improvement: Enhance the chatbot's ability to understand complex queries related to the design and architecture of the code. The bot could help with tasks such as code navigation, explaining design decisions, or providing reasoning for certain implementations based on the requirements. Using advanced contextual AI models could make the bot more interactive and intelligent.

8. Collaborative and Shared Traceability Current Limitation: The tool is currently focused on individual developers and their requirements traceability.

Future Improvement: Enable collaborative features where multiple developers can interact with the traceability matrix and requirement matching tool in real-time. This would allow teams to collaboratively review and update the code's alignment with requirements, improving team coordination and communication.

9. Integration with Version Control Systems Current Limitation: The project does not integrate directly with version control systems like Git.

Future Improvement: Integrate the tool with version control systems (e.g., Git, GitHub, GitLab) to automatically track changes in code and analyze how new commits impact requirement coverage and code quality. This would allow for continuous integration and more seamless code analysis during the development process.

10. Support for Non-Functional Requirements Current Limitation: The focus is mainly on functional requirements.

Future Improvement: Expand the system's ability to track non-functional requirements such as performance, security, and usability. By incorporating checks for these types of requirements, the system could offer a more comprehensive view of how well the code meets all aspects of the project's specifications.

9 REFERENCES

Below are the references for the libraries and frameworks used in this project:

(1) **Flask**: Flask is a lightweight Python web framework for building web applications. https://flask.palletsprojects.com/
(2) **transformers**: Hugging Face's transformers library provides pre-trained models for a wide range of NLP tasks. https://huggingface.co/transformers/
(3) **pdfminer.six**: A Python library for extracting text, images, and metadata from PDF files. https://pdfminersix.readthedocs.io/
(4) **requests**: A simple HTTP library for Python to send HTTP requests. https://docs.python-requests.org/en/master/
(5) **openai**: The official Python package to interact with OpenAI's API for tasks such as text generation. https://openai.com/
(6) **huggingface_hub**: A Python library for interacting with Hugging Face's Model Hub to access pre-trained models. https://huggingface.co/docs/huggingface_hub
(7) **torch**: PyTorch is an open-source deep learning framework for building neural networks. https://pytorch.org/
(8) **PyPDF2**: A Python library for reading and manipulating PDF files. https://pythonhosted.org/PyPDF2/
(9) **TensorFlow**: An open-source machine learning framework for building and deploying machine learning models. https://www.tensorflow.org/
(10) **spaCy**: A library for advanced natural language processing in Python, designed specifically for production use. https://spacy.io/
(11) **NumPy**: A fundamental package for scientific computing with Python, supporting large, multi-dimensional arrays and matrices. https://numpy.org/
(12) **pandas**: A powerful, flexible, and easy-to-use open-source data analysis and manipulation library for Python. https://pandas.pydata.org/
(13) **scikit-learn**: A machine learning library for Python that provides simple and efficient tools for data analysis and modeling. https://scikit-learn.org/
(14) **Plotly**: A graphing library for creating interactive, publication-quality graphs online. https://plotly.com/
(15) **Matplotlib**: A Python library for creating static, animated, and interactive visualizations. https://matplotlib.org/
(16) **SQLAlchemy**: A SQL toolkit and Object-Relational Mapping (ORM) library for Python. https://www.sqlalchemy.org/
(17) **BeautifulSoup**: A Python library for parsing HTML and XML documents and extracting data from them. https://www.crummy.com/software/BeautifulSoup/
(18) **Celery**: An asynchronous task queue/job queue based on distributed message passing. https://docs.celeryproject.org/
(19) **Gunicorn**: A Python WSGI HTTP server for UNIX, often used to serve Flask applications. https://gunicorn.org/
(20) **Docker**: A platform used to develop, ship, and run applications inside lightweight containers. https://www.docker.com/
(21) **Jupyter**: A web-based interactive computing environment for creating notebooks that combine code, text, and data visualizations. https://jupyter.org/
(22) **FastAPI**: A modern, fast (high-performance), web framework for building APIs with Python. https://fastapi.tiangolo.com/