Automating VBA Macro Documentation and Transformation

Vasudha R B-21z267
Thiriksha S-21z263

PROBLEM STATEMENT:

Automating VBA Macro Documentation and Transformation: Leveraging Cutting Edge Technologies to Modernize and Document Legacy VBA Macros for Enhanced Operational Efficiency. Many organizations rely on legacy VBA (Visual Basic for Applications) macros for various automated tasks in Microsoft Office applications. However, documenting and transforming these macros can be time-consuming and error-prone. This project aims to automate the extraction, documentation, and visualization of VBA macros to improve operational efficiency and maintainability.

OVERVIEW:

Organizations often rely on legacy VBA (Visual Basic for Applications) macros for automating tasks in Microsoft Office applications. Documenting and transforming these macros can be labor-intensive and prone to errors. This project aims to automate the extraction, documentation, and visualization of VBA macros, enhancing operational efficiency and maintainability.

PLATFORM ARCHITECTURE:

Frontend:

- A web interface built with Flask and HTML/CSS for users to upload Excel files containing VBA macros.
- Provides a form for uploading files and a result page displaying the documentation and flow diagram.

Backend:

- A Flask server handles file uploads, extracts VBA code, generates documentation, and creates flow diagrams.
- Contains logic for processing VBA code, generating documentation, and creating flow diagrams.

Processing Logic:

- Python scripts extract VBA code from the uploaded files, generate documentation using python-docx, and create flow diagrams using Graphviz.

ALGORITHM USED:

1.File Upload and Extraction:

- Users upload an Excel file (.xlsm) through the web interface.
- The backend saves the file and extracts its contents using the zipfile module.

2.VBA Code Extraction:

- The backend scans the extracted files for VBA code files (.vb, .bas, .cls).
- Reads and aggregates the content of these files.

3.Documentation Generation:

- Formats the aggregated VBA code into Word document using python-docx library.
- Organizes headings and code sections for readability.

4.Flow Diagram Creation:

- Analyzes the VBA code to extract functions and their calls.
- Generates a flow diagram using Graphviz, illustrating the structure and call relationships between functions.

5.Result Display:

- Presents the generated documentation and flow diagram to the user through the web interface.

DATASET USED:

The dataset in this project consists of Excel files (.xlsm) that contain VBA (Visual Basic for Applications) macros. These macros are often used to automate tasks within Microsoft Office applications, such as data processing, calculations, and user interface interactions. The primary focus of the dataset is the VBA code embedded within these Excel files, which is extracted, documented, and analyzed by the application.

Key Characteristics:

1. File Format: The dataset comprises Excel macro-enabled files with the extension .xlsm.

2. VBA Code Files: Within the .xlsm files, the VBA code is typically stored in modules with extensions such as .vb, .bas, and .cls.
3. Content: The VBA code may include functions, subroutines, and various logical operations that automate specific tasks within the Excel environment.
4. Structure: The structure of the VBA code can vary significantly, including simple scripts to complex, multi-module projects with intricate dependencies and function calls.

Use Cases:

1. Automation Tasks: The VBA macros automate repetitive tasks, improve efficiency, and reduce human error in data processing.
2. Documentation: Generating documentation for existing VBA code to improve maintainability and knowledge transfer.
3. Visualization: Creating flow diagrams to visualize the structure and dependencies within the VBA code for better understanding and debugging

SOLUTION:

The solution is implemented as a Flask web application with key components:

1.Flask Application:

- Handles routing and rendering of HTML templates for file upload and result display.

2.HTML/CSS Templates:

- **upload.html:** Form for file uploads.
- **result.html:** Displays links to download the generated documentation and view the flow diagram.

3.Python Scripts:

- **extract_vba_code(file):** Extracts VBA code from the uploaded Excel file.
- generate_documentation(vba_code): Creates a Word document with the extracted VBA code.
- **generate_flow_diagram(vba_code):** Generates a flow diagram of the VBA functions and their calls using Graphviz.
- **extract_functions(vba_code):** Identifies functions in the VBA code.
- **extract_function_calls(vba_code, func):** Identifies function calls within a specific function.

CODE IMPLEMENTATION:

Flask Application and Processing Logic:

```
from flask import Flask, request, render_template, send_file
import zipfile
import os
import shutil
import docx
from docx import Document
import graphviz

app = Flask(_name_)

def extract_vba_code(file):
    with zipfile.ZipFile(file, 'r') as zip_ref:
        zip_ref.extractall('temp')
    vba_code = ""
    for root, dirs, files in os.walk('temp'):
        for file in files:
            if file.endswith('.vb') or file.endswith('.bas') or file.endswith('.cls'):
                with open(os.path.join(root, file), 'r') as vb_file:
                    vba_code += vb_file.read()
    shutil.rmtree('temp')
    return vba_code

def generate_documentation(vba_code):
    doc = Document()
    doc.add_heading('VBA Macro Documentation', 0)
    doc.add_heading('Extracted VBA Code:', level=1)
    doc.add_paragraph(vba_code)

    filename = "VBA_Documentation.docx"
    doc.save(filename)
    return filename

def generate_flow_diagram(vba_code):
    dot = graphviz.Digraph(comment='VBA Macro Flow Diagram')
```

```
    # Placeholder for logic extraction and diagram creation
    # This part needs to be customized based on the actual structure and logic of your VBA
code
    functions = extract_functions(vba_code)
    for func in functions:
        dot.node(func, func)
    for func in functions:
        for call in extract_function_calls(vba_code, func):
            dot.edge(func, call)

    diagram_path = 'flow_diagram'
    dot.render(diagram_path, format='png')
    return f'{diagram_path}.png'

def extract_functions(vba_code):
    # Extract function names from the VBA code
    functions = []
    lines = vba_code.splitlines()
    for line in lines:
        if line.strip().lower().startswith("sub "):
            func_name = line.strip().split()[1].split('(')[0]
            functions.append(func_name)
    return functions

def extract_function_calls(vba_code, func):
    # Extract function calls within a function from the VBA code
    calls = []
    lines = vba_code.splitlines()
    inside_func = False
    for line in lines:
        if line.strip().lower().startswith(f"sub {func.lower()}"):
            inside_func = True
        elif line.strip().lower().startswith("end sub"):
            inside_func = False
        elif inside_func and "(" in line and ")" in line:
            call = line.strip().split('(')[0].split()[-1]
            calls.append(call)
    return calls
```

```
@app.route('/')
def upload_file():
    return render_template('upload.html')

@app.route('/analyze', methods=['POST'])
def analyze_file():
    file = request.files['file']
    file.save("uploaded_file.xlsm")
    vba_code = extract_vba_code("uploaded_file.xlsm")
    documentation_file = generate_documentation(vba_code)
    diagram_file = generate_flow_diagram(vba_code)
    return render_template('result.html', documentation_file=documentation_file,
diagram_file=diagram_file)

if _name_ == '_main_':
    app.run(debug=True)
```

HTML Templates and CSS:

```
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

.container {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;
    border: 1px solid #ddd;
    border-radius: 4px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1, h2 {
    text-align: center;
}

form {
```

```
    display: flex;
    flex-direction: column;
    align-items: center;
}

input[type="file"], input[type="submit"] {
    margin: 10px 0;
}
```

CONCLUSION:

Automating the documentation and transformation of VBA macros is essential for modernizing legacy systems and improving operational efficiency. This project offers a user-friendly web application that extracts, documents, and visualizes VBA code from Excel files. By leveraging Flask, Python, and supporting libraries, the solution provides detailed documentation and flow diagrams, making the VBA code more maintainable and understandable. This automation saves time, reduces errors, and ensures comprehensive documentation of VBA macros. Overall, this project demonstrates the effective use of modern technologies to enhance the management and modernization of legacy VBA systems.