Analysis of project: **Effort & Cost Estimation, Requirement & Function Size,** and **Code Structure & Size**

for

EXAM PILOT

Prepared by

MIR MOHAMMAD TAHSIN | MUH2025007M

IMTIAZ CHOWDHURY | MUH2025027M

IRFANUL HAQUE | ASH1925021M

Institute of Information Technology

Noakhali Science and Technology University

Group Assignment of

SE 3204 ~ Software Metrics Lab

Submitted to

Md Hasan Imam

Lecturer

Institute of Information Technology

Noakhali Science and Technology University

Date of Submission: 26 December 2023

Task Distribution

Task	Member
1. Effort and Cost Estimation using COCOMO II model of SPL 2	Mir Mohammad Tahsin (MUH2025007M)
2. Requirement & Function Size of SPL 2	Irfanul Haque (ASH1925021M)
3. Code Structure and code size Measurement using Halstead approach of SPL 2	**Imtiaz Chowdhury (MUH2025027M)**

Table of Contents

Tables

Figures

Document for Code Structure and Code Size Measurement Using the Halstead Approach

Introduction

This document outlines the *Halstead metrics* for ***measuring code structure and size***, providing definitions, measurement processes, and corresponding values. It aims to guide software developers and analysts in effectively assessing code complexity and maintainability.

Halstead Metrics

The Halstead approach focuses on quantifying code complexity based on its *vocabulary* and *length*. It uses four primary metrics:

1. Program Vocabulary **(n)**: Total number of unique operators and operands in the program.
   - **n_1: Number of distinct operators** (keywords, operators, punctuation)
   - **n_2: Number of distinct operands** (variables, constants, literals)
2. Program Length **(N)**: Total number of operator occurrences and operand occurrences in the program.
   - **N_1: Total occurrences of operators**
   - **N_2: Total occurrences of operands**

Derived Metrics:

3. Program Volume **(V)**: Quantifies the size of the code based on the total vocabulary and program length.

   $$\mathbf{V} = N * \log_2(n)$$

   $$= (N_1 + N_2) * \log_2(n_1 + n_2)$$

4. Program Difficulty **(D)**: Measures the difficulty in understanding the code based on the program's vocabulary size and the number of unique operators.

   $$\mathbf{D} = (n_1/2) * (N_2/n_2)$$

5. Program Effort **(E)**: Quantifies the effort to write the code based on volume and difficulty.

   $$\mathbf{E} = V * D$$

6. Program Level: **L** = 1/D
7. Estimated Program Length: **LE** = V/L
8. Programming Time **(T)**: Estimates the time to write the code based on effort.

   $$\mathbf{T} = E/18$$

9. Number of Delivered Bugs: **B** = V/3000

Measurement Process

1. **Identify Operators and Operands:**
    - Manually count distinct operators and operands in the code.
    - Utilize automated tools for counting (e.g., static code analyzers).
2. **Count Occurrences:**
    - Tally the total occurrences of each operator and operand.
3. **Calculate Metrics:**
    - Apply the formulas above to compute the derived metrics.

Halstead's Approach

Definition	Halstead's theory is an analytical estimation technique to measure the size, development effort, and development cost of software products.
Halstead Metrics	All necessary **key word**, **definition** and **formula** are given in Halstead Metrics part.
Value Calculation	• Program Vocabulary, **n =** 1660+63657 = **65317** • Program Length, **N =** 90222+230624 = **320846** • Program Volume **(V)**: **V** = N * $\log_2(n)$ = 320846 * $\log_2(65317)$ = **5131986.606** • Program Difficulty **(D)**: **D** = (1660/2) * (230624/63657) = **3007.020752** • Program Effort **(E)**: **E** = V * D = 5131986.606*3007.020752 = **$1.543199022*10^{10}$** • Program Level: **L** = 1/D = 1/3007.020752 = **$3.325550711*10^{-4}$** • Estimated Program Length: **LE** = V/L = $5131986.606/3.325550711*10^{-4}$ = **$1.543199022*10^{10}$**

	- Programming Time **(T)**: $\mathbf{T} = E/18$ $= 1.543199022*10^{10}/18$ $\mathbf{= 857332790}$ - Number of Delivered Bugs: **B** = V/3000 $= 5131986.606/3000$ $\mathbf{= 1710.662202}$

Table 1: Halstead's Approach (Value Calculation)

Interpreting Values

Metric	**Value**	**Interpretation**
Program Volume	5,131,986.61	The Exam Pilot codebase stands at an impressive size, testament to the depth and breadth of functionality achieved.
Program Difficulty	3,007.02	The intricate nature of the code, likely due to complex algorithms or intricate structures, demanded a high level of expertise from our development team.
Program Effort	1.543199022*10^10	The project's development was a significant undertaking, requiring a substantial investment of time and resources. Meticulous planning, resource allocation, and stringent quality control measures were critical to success.
Program Level	3.325550711*10^-4	The low program level highlights the high degree of cognitive load required to navigate the code's intricacies. This underscores the importance of focused attention, knowledge sharing, and clear documentation practices employed by the team.

Table 2: Interpreting Values

- **Higher Program Volume:** Indicates larger, more complex code.
- **Higher Difficulty:** Suggests more challenging code to understand and maintain.
- **Higher Effort:** Implies greater development time and potential for errors.
- **Lower Level:** Signals more complex code with higher cognitive load.

Applications

	Uses	**Topics**
We can	✓ identify potentially problematic code sections for refactoring.	**Code Complexity Assessment**
	✓ predict development time and effort.	**Effort Estimation**
	✓ estimate potential number of defects.	**Bug Prediction**
	✓ evaluate code quality across different versions or projects.	**Code Comparison**

Table 3: Applications of Code Structure and Code Size Measurement

Additional Considerations

- **Language-Specific Adjustments:** We should consider language-specific variations in operator and operand definitions.

  In our **SPL II (Web based Application):** ***Exam Pilot*** project we use HTML, CSS, php, JavaScript.

- **Normalization:** Again, we should normalize metrics for comparison across different code sizes.

- **Contextual Analysis:** Here we combine *Halstead metrics* with other code quality measures (in Appendix) for a comprehensive assessment.

Conclusion

The Halstead approach provides valuable insights into code structure and size, facilitating software development, maintenance, and quality assurance. By understanding and applying these metrics, we (developers and analysts) can make informed decisions to improve code maintainability, reduce development costs, and minimize defects.

Appendix

Determining Code Size

Lines of Code (LOC)

Definition	Lines of code are the "source code" of the program, and one line may generate one machine instruction or several depending on the programming language.
Measurement Procedure	Automated Program (**vscode-counter**)
Value	23,696

Table 4: LOC

Commented lines of code (CLOC)

Definition	A comment is a programmer-readable explanation or annotation in the source code of a computer program.
Measurement Procedure	Automated Program (**vscode-counter**)
Value	1,164

Table 5: CLOC

Non commented lines of code (NCLOC)

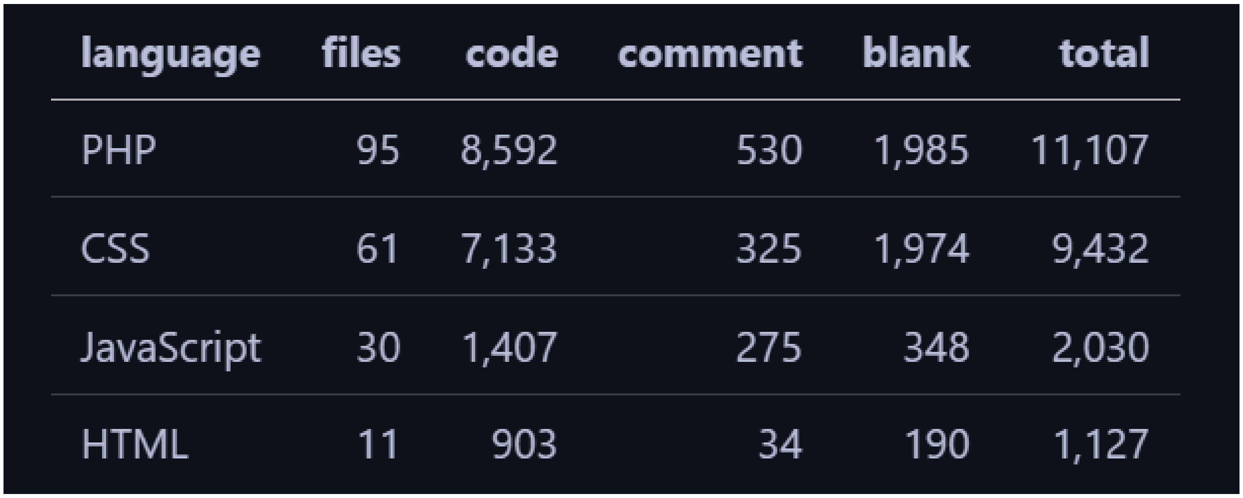
Definition	The number of physical lines that contain at least one character which is neither a whitespace nor a tabulation nor part of a comment.
Measurement Procedure	Automated Program (**vscode-counter**)
Value	18,035

Table 6: NCLOC

Blank Lines of Code (BLOC)

Definition	Blank Lines represents lines without any statement or symbol. They are present in code to increase readability and clarity.
Measurement Procedure	Automated Program (**vscode-counter**)
Value	4,497

Table 7: BLOC



language	files	code	comment	blank	total
PHP	95	8,592	530	1,985	11,107
CSS	61	7,133	325	1,974	9,432
JavaScript	30	1,407	275	348	2,030
HTML	11	903	34	190	1,127

Figure 1: Summary of Code Size

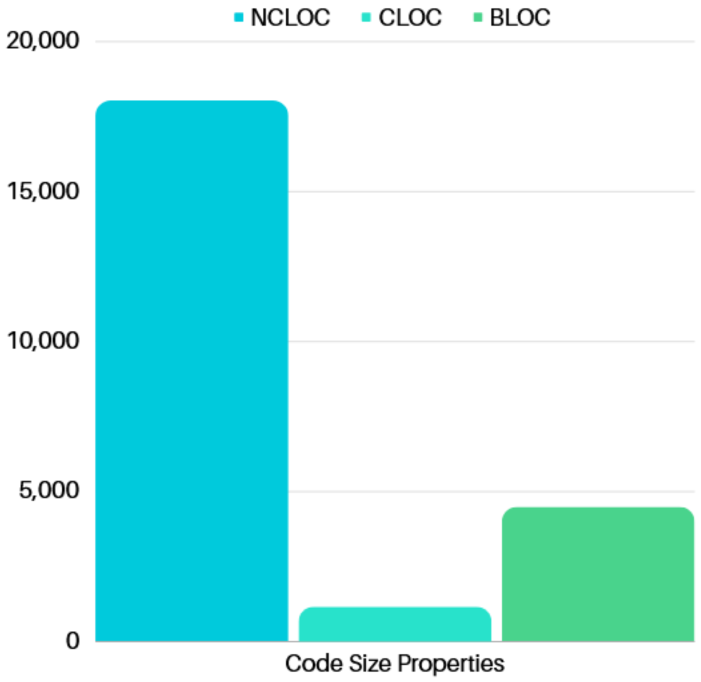


Figure 2: Code Size Properties

Density of comments

Definition	Comment density is the percentage of comment lines in each source code base, that is, comment lines divided by total lines of code.
Measurement Procedure	Manually
Value	0.049

Table 8: Density of comments

Number of bytes of computer storage

Definition	Number of bytes used in the computer storage for the program text.
Measurement Procedure	Automated Program

Value	49 MB

Table 9: Number of bytes of computer storage

Determining Design Size

Category	Subcategory	Metric	Value
Structure	Classes	Namespaces	0
		Interfaces	0
		Traits	0
	Functions	Functions	28
		Named Functions	28 (100.00%)
		Anonymous Functions	0 (0.00%)
Size	Functions	Functions	249
		Average Function Length	8
	Variables	Other	Not in classes or functions
Complexity	Overall	Average Complexity per LLOC	0.21
Dependencies	Global Variables	Global Accesses	338
		Global Variables	12 (3.55%)
		Super-Global Variables	326 (96.45%)
	Attributes	Attribute Accesses	281
		Non-Static	281 (100.00%)
	Methods	Method Calls	338
		Non-Static	338 (100.00%)

Table 10: Design Size Measurement

Cyclomatic Complexity Dd (Decision Density) Graph of Some Important Code Modules

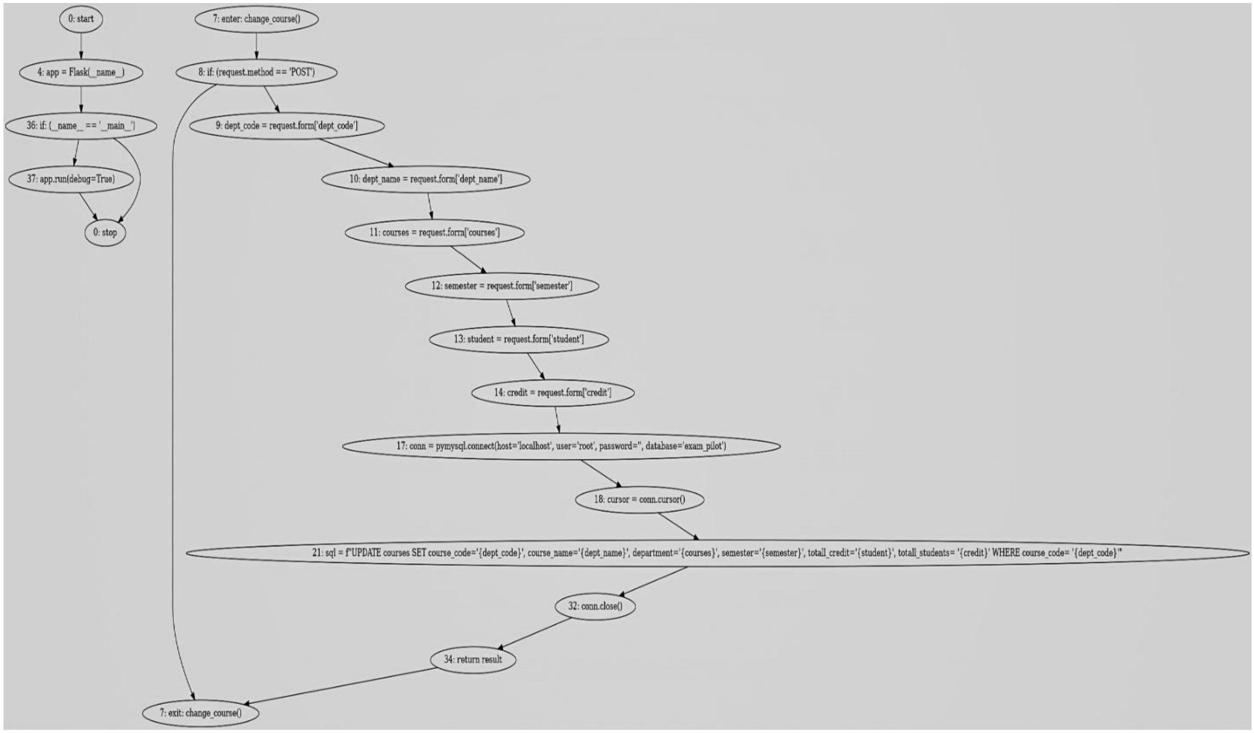

0: start
4: app = Flask(__name__)
36: if: (__name__ == '__main__')
37: app.run(debug=True)
0: stop
7: enter: change_course()
8: if: (request.method == 'POST')
9: dept_code = request.form['dept_code']
10: dept_name = request.form['dept_name']
11: courses = request.form['courses']
12: semester = request.form['semester']
13: student = request.form['student']
14: credit = request.form['credit']
17: conn = pymysql.connect(host='localhost', user='root', password='', database='exam_pilot')
18: cursor = conn.cursor()
21: sql = f"UPDATE courses SET course_code='{dept_code}', course_name='{dept_name}', department='{courses}', semester='{semester}', totall_credit='{student}', totall_students= '{credit}' WHERE course_code= '{dept_code}'"
32: conn.close()
34: return result
7: exit: change_course()

Figure 3: change_course dd graph

Cyclomatic Complexity:
v (change_course) = e -n + 2p = 19 -19 + 2*2 = 4

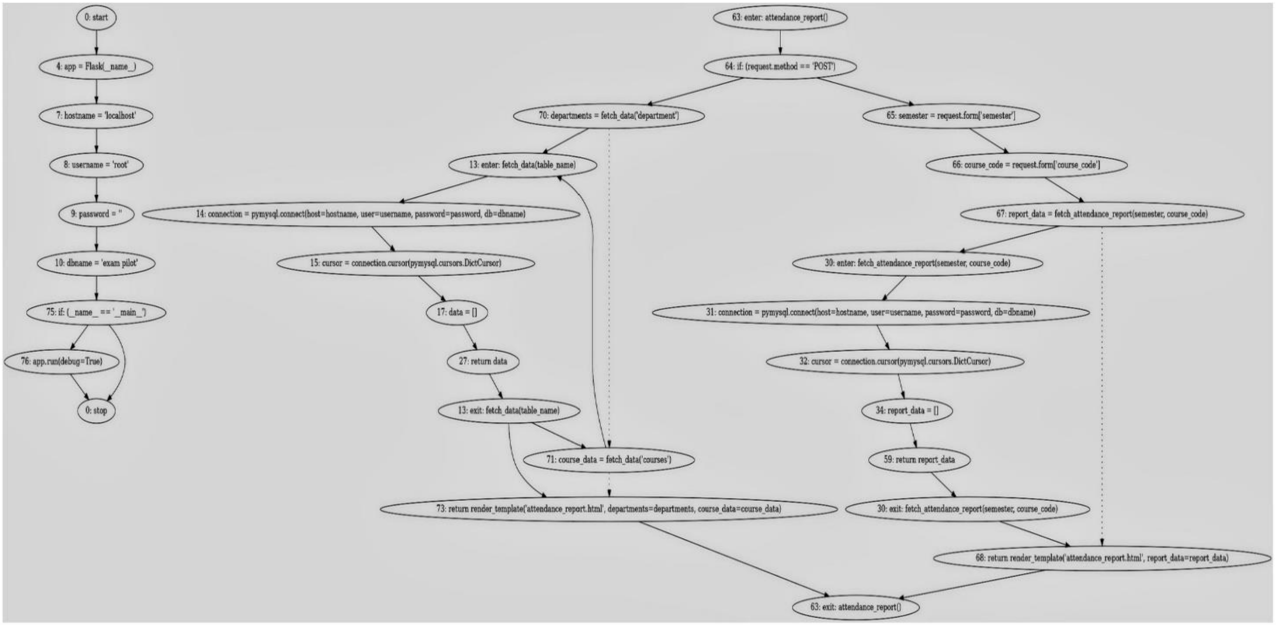

0: start
4: app = Flask(__name__)
7: hostname = 'localhost'
8: username = 'root'
9: password = ''
10: dbname = 'exam pilot'
75: if: (__name__ == '__main__')
76: app.run(debug=True)
0: stop
63: enter: attendance_report()
64: if: (request.method == 'POST')
70: departments = fetch_data('department')
65: semester = request.form['semester']
13: enter: fetch_data(table_name)
66: course_code = request.form['course_code']
14: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
67: report_data = fetch_attendance_report(semester, course_code)
15: cursor = connection.cursor(pymysql.cursors.DictCursor)
30: enter: fetch_attendance_report(semester, course_code)
17: data = []
31: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
27: return data
32: cursor = connection.cursor(pymysql.cursors.DictCursor)
13: exit: fetch_data(table_name)
34: report_data = []
71: course_data = fetch_data('courses')
59: return report_data
73: return render_template('attendance_report.html', departments=departments, course_data=course_data)
30: exit: fetch_attendance_report(semester, course_code)
68: return render_template('attendance_report.html', report_data=report_data)
63: exit: attendance_report()

Figure 4: attendance_report dd graph

Cyclomatic Complexity:
v (attendance_report) = e -n + 2p = 34 -31 + 2*2 = 7

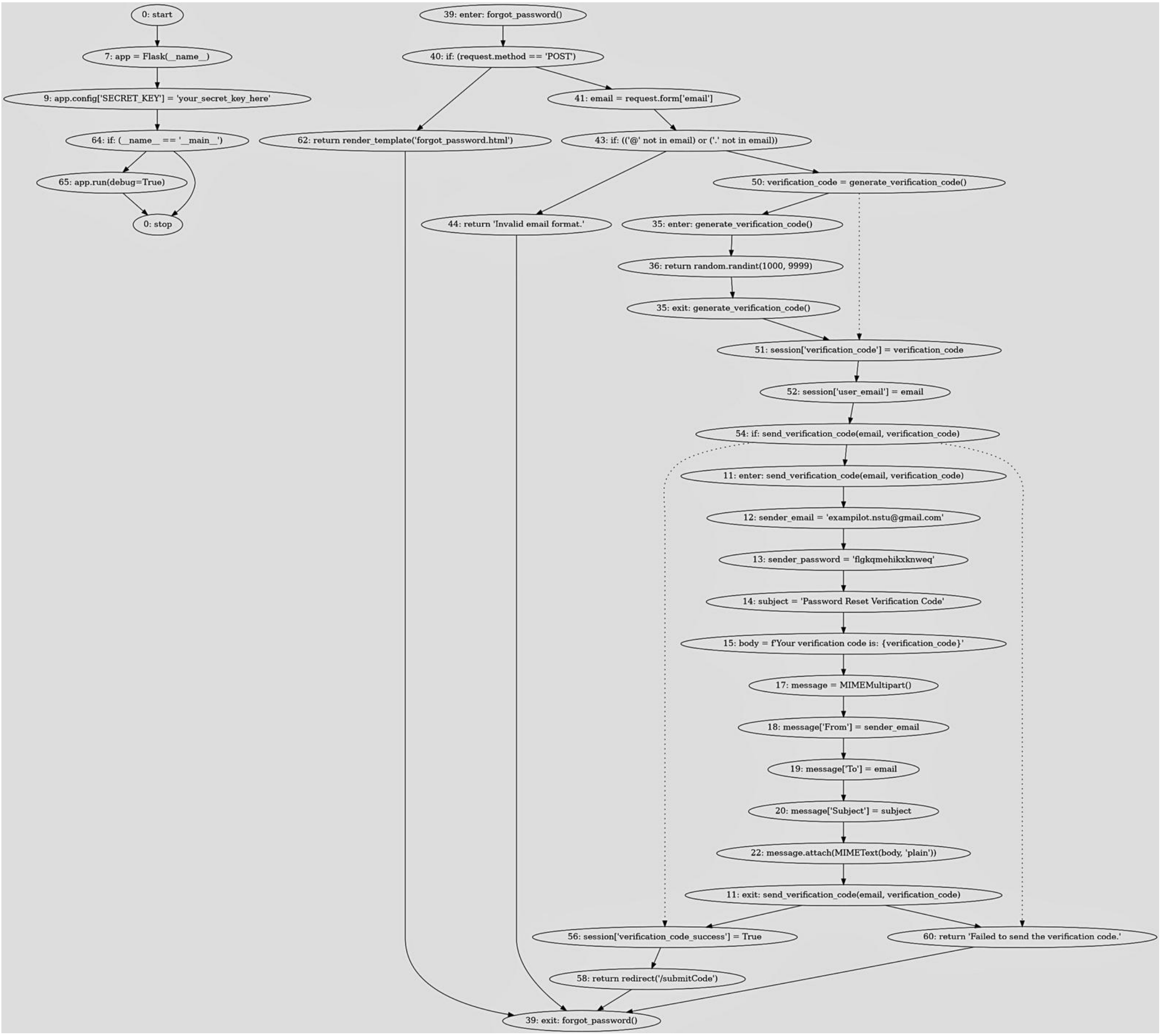

0: start
7: app = Flask(__name__)
9: app.config['SECRET_KEY'] = 'your_secret_key_here'
64: if: (__name__ == '__main__')
65: app.run(debug=True)
0: stop
39: enter: forgot_password()
40: if: (request.method == 'POST')
41: email = request.form['email']
62: return render_template('forgot_password.html')
43: if: (('@' not in email) or ('.' not in email))
50: verification_code = generate_verification_code()
44: return 'Invalid email format.'
35: enter: generate_verification_code()
36: return random.randint(1000, 9999)
35: exit: generate_verification_code()
51: session['verification_code'] = verification_code
52: session['user_email'] = email
54: if: send_verification_code(email, verification_code)
11: enter: send_verification_code(email, verification_code)
12: sender_email = 'exampilot.nstu@gmail.com'
13: sender_password = 'flgkqmehikxknweq'
14: subject = 'Password Reset Verification Code'
15: body = f'Your verification code is: {verification_code}'
17: message = MIMEMultipart()
18: message['From'] = sender_email
19: message['To'] = email
20: message['Subject'] = subject
22: message.attach(MIMEText(body, 'plain'))
11: exit: send_verification_code(email, verification_code)
56: session['verification_code_success'] = True
60: return 'Failed to send the verification code.'
58: return redirect('/submitCode')
39: exit: forgot_password()

Figure 5: forgot_pass dd graph

Cyclomatic Complexity:
v (forgot_pass) = e -n + 2p = 39 -34 + 2*2 = 9

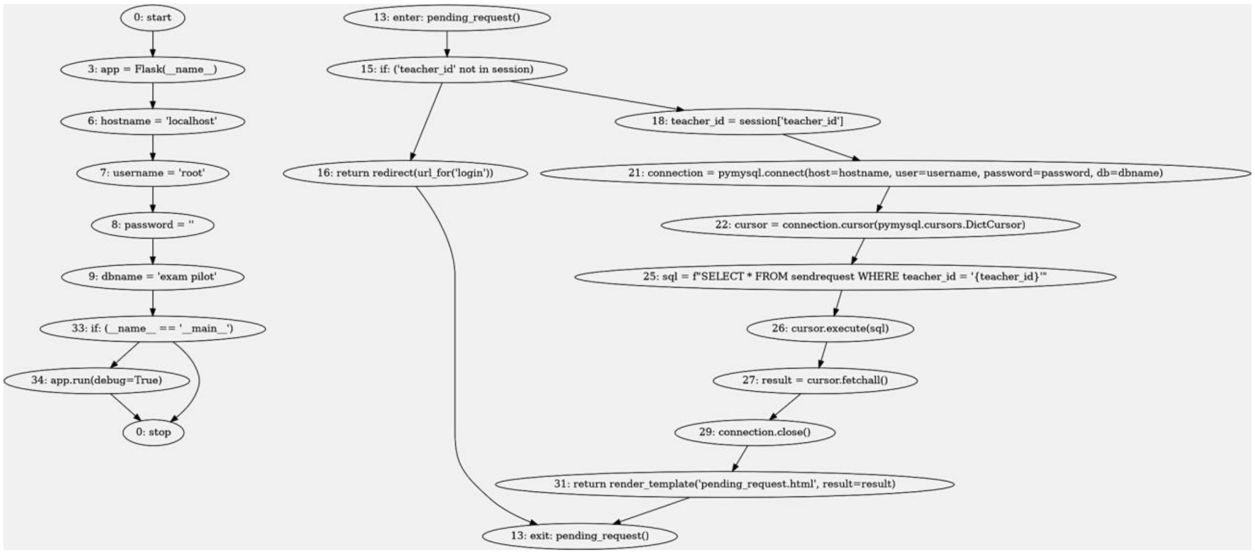

0: start
3: app = Flask(__name__)
6: hostname = 'localhost'
7: username = 'root'
8: password = ''
9: dbname = 'exam pilot'
33: if: (__name__ == '__main__')
34: app.run(debug=True)
0: stop
13: enter: pending_request()
15: if: ('teacher_id' not in session)
16: return redirect(url_for('login'))
18: teacher_id = session['teacher_id']
21: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
22: cursor = connection.cursor(pymysql.cursors.DictCursor)
25: sql = f"SELECT * FROM sendrequest WHERE teacher_id = '{teacher_id}'"
26: cursor.execute(sql)
27: result = cursor.fetchall()
29: connection.close()
31: return render_template('pending_request.html', result=result)
13: exit: pending_request()

Figure 6: Pending Request dd graph

Cyclomatic Complexity:
v (Pending Request) = e -n + 2p = 21 -21 + 2*2 = 4

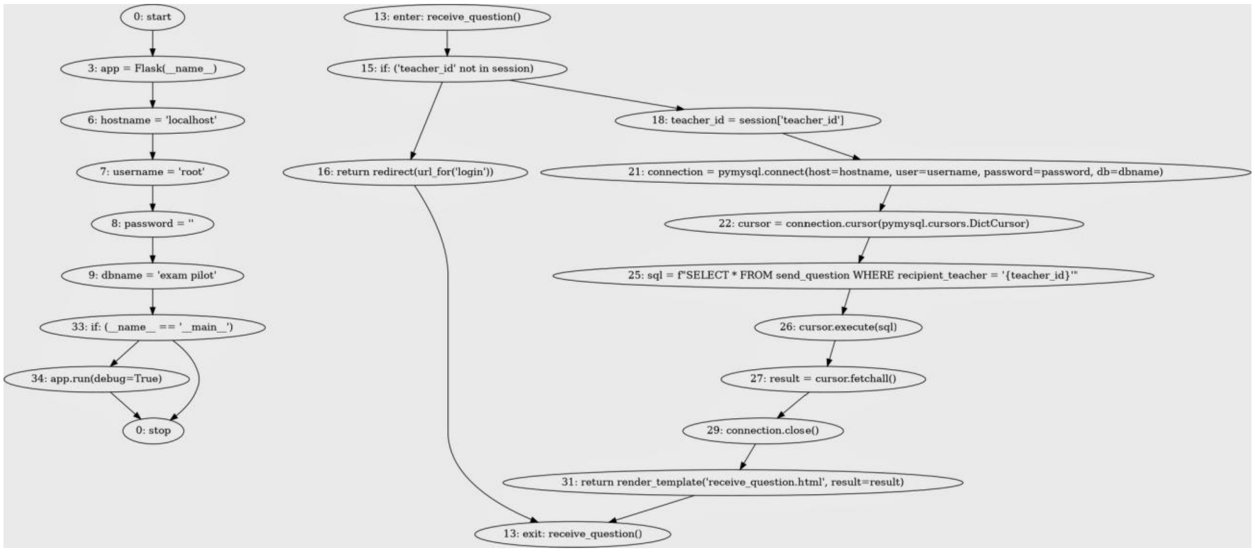

0: start
3: app = Flask(__name__)
6: hostname = 'localhost'
7: username = 'root'
8: password = ''
9: dbname = 'exam pilot'
33: if: (__name__ == '__main__')
34: app.run(debug=True)
0: stop
13: enter: receive_question()
15: if: ('teacher_id' not in session)
16: return redirect(url_for('login'))
18: teacher_id = session['teacher_id']
21: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
22: cursor = connection.cursor(pymysql.cursors.DictCursor)
25: sql = f"SELECT * FROM send_question WHERE recipient_teacher = '{teacher_id}'"
26: cursor.execute(sql)
27: result = cursor.fetchall()
29: connection.close()
31: return render_template('receive_question.html', result=result)
13: exit: receive_question()

Figure 7: Receive Question dd graph

Cyclomatic Complexity:
v (Receive Question) = e -n + 2p = 21 -21 + 2*2 = 4

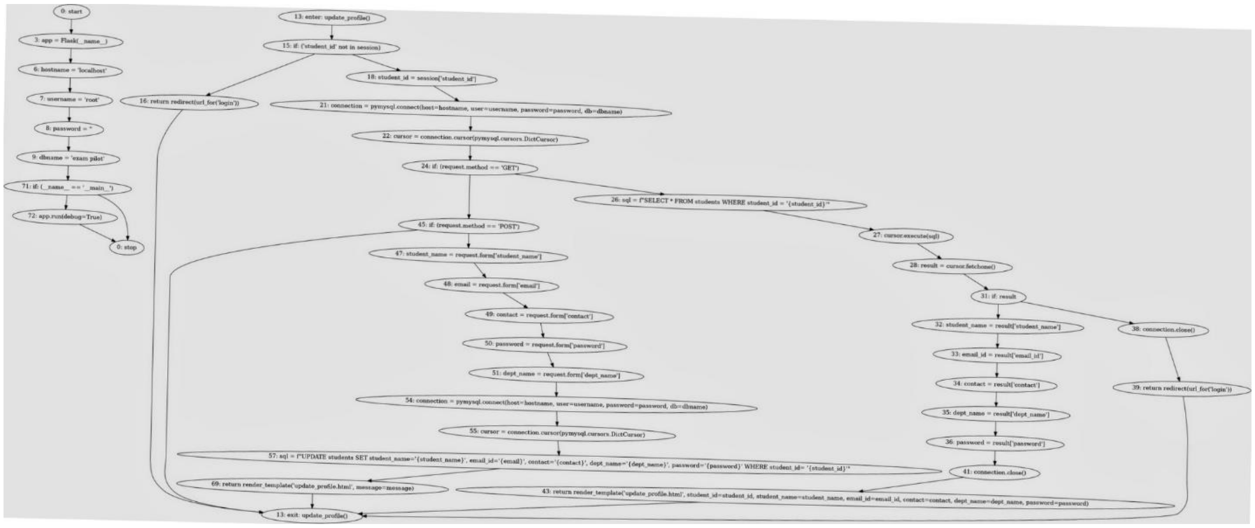

0: start
3: app = Flask(__name__)
6: hostname = 'localhost'
7: username = 'root'
8: password = ''
9: dbname = 'exam pilot'
71: if: (__name__ == '__main__')
72: app.run(debug=True)
0: stop
13: enter: update_profile()
15: if: ('student_id' not in session)
16: return redirect(url_for('login'))
18: student_id = session['student_id']
21: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
22: cursor = connection.cursor(pymysql.cursors.DictCursor)
24: if: (request.method == 'GET')
26: sql = f"SELECT * FROM students WHERE student_id = '{student_id}'"
27: cursor.execute(sql)
28: result = cursor.fetchone()
31: if: result
32: student_name = result['student_name']
33: email_id = result['email_id']
34: contact = result['contact']
35: dept_name = result['dept_name']
36: password = result['password']
41: connection.close()
38: connection.close()
39: return redirect(url_for('login'))
45: if: (request.method == 'POST')
47: student_name = request.form['student_name']
48: email = request.form['email']
49: contact = request.form['contact']
50: password = request.form['password']
51: dept_name = request.form['dept_name']
54: connection = pymysql.connect(host=hostname, user=username, password=password, db=dbname)
55: cursor = connection.cursor(pymysql.cursors.DictCursor)
57: sql = f"UPDATE students SET student_name='{student_name}', email_id='{email}', contact='{contact}', dept_name='{dept_name}', password='{password}' WHERE student_id= '{student_id}'"
69: return render_template('update_profile.html', message=message)
43: return render_template('update_profile.html', student_id=student_id, student_name=student_name, email_id=email_id, contact=contact, dept_name=dept_name, password=password)
13: exit: update_profile()

Figure 8: updateProfile(student) dd graph

Cyclomatic Complexity:
v (updateProfile) = e -n + 2p = 43 -40 + 2*2 = 7