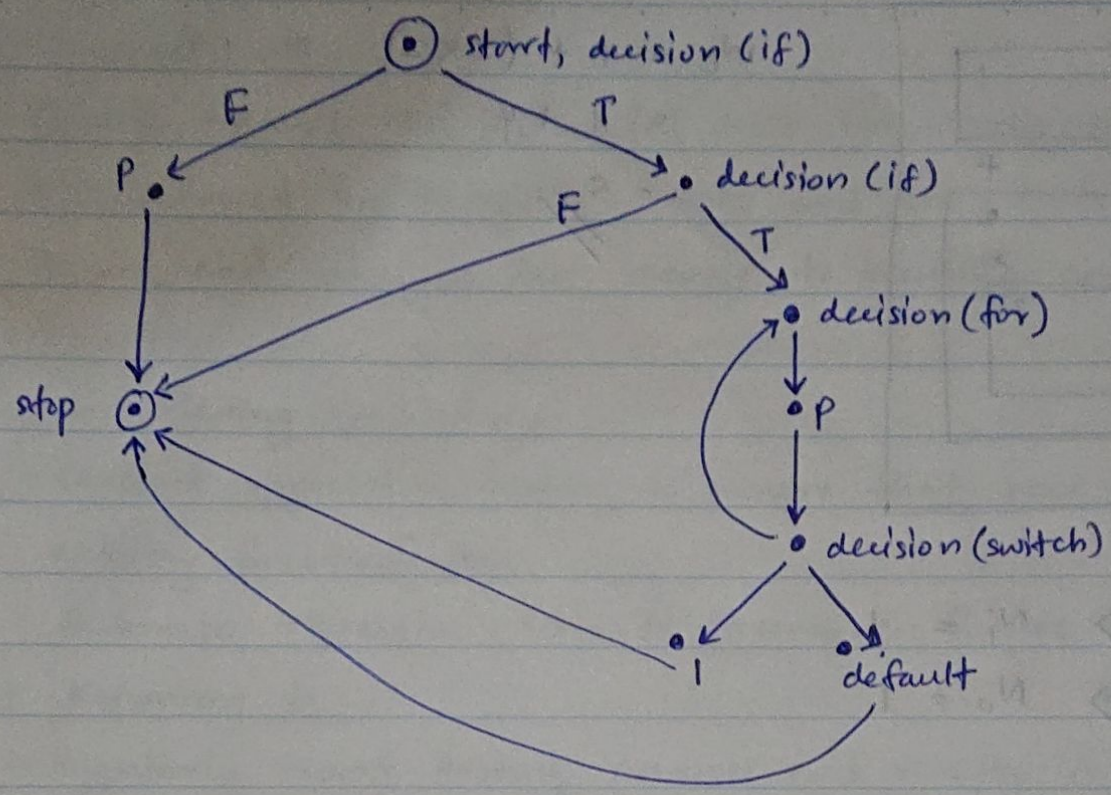
SEPQM 2023

(01)

a)



b) $V(G) = e - n + 2$

$= 12 - 9 + 2$

$= 3 + 2$

$= 5$ //

e ⇒ no. of edges = 12

n ⇒ no. of nodes = 9

c) $V(a) = d + 1$

$= 4 + 1$

$= 5$ //

d = no. of decision nodes = 4

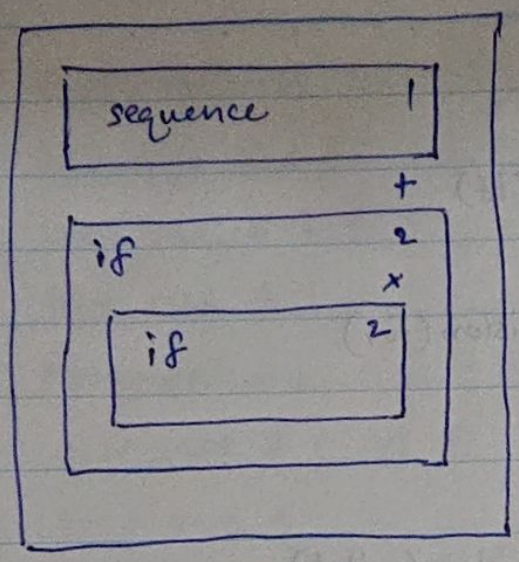
d) Dissambled byte code typically results in a similar cyclomatic complexity, if the logical structure remains intact.

∴ cyclomatic complexity remains $V(a) = 5$ //

e) Cyclomatic complexity measures the no. of linearly independent paths through a program's source code, focusing on decision points rather than the procedural steps. The procedural nodes do not introduce new decision paths. So, the complexity remains governed by the decision structures like for, if statements, while.... etc. rather than procedural nodes.

(02)

a)



$W_c = 1 + (2 \times 2)$

$= 1 + 4$

$= 5$ //

b) no. of inputs ⇒ $N_i = 1$

no. of outputs ⇒ $N_o = 1$

c) $S_f = (N_i + N_o) \times W_c$

$= (1 + 1) \times 5$

$= 2 \times 5$

$= 10$ //

(03) (a structured test plan should be established)

a) i) Requirements Analysis and test planning ⇒

- identify the critical functionalities for the 1st release : user, coure, assignmen management ~~and~~ and assignment submission.
- develop a detailed test plan specifying test cases, test data, expected results, and testing criteria for these main functionalities.

Manual Testing ⇒

- Since there might be limited time to set up comprehensive automated tests initially, begin with manual testing to ensure basic functionalities work as expected.
- Create test cases for each functionality.

Automated Testing ⇒

- Implement automated unit tests for core functions (Ex: JUnit)
- Write automated integration tests to ensure different components interact correctly.

Test Execution ⇒

- Execute manual and automated tests iteratively, starting with unit tests, followed by integration tests and UI tests.
- Record and analyse test results to identify and fix defects promptly.

Regression Testing ⇒

- Conduct regression testing to ensure that new changes do not break existing functionality.
- Automate regression tests to streamline future testing efforts.

Test Reporting ⇒

- Regularly report testing progress and results to the team.
- Use continous Integration tools (CI) like Jenkins to automate the building and testing process, providing realtime feedback to test results.

ii) (establish a robust CI/CD pipeline and testing framework.)

Continuous Integrations and continuous Deployment (CI/CD) ⇒

- Set up a CI/CD pipline using Jenkins or similar tool to automate the build, test, and deployment process.
- Ensure that the pipeline runs automated tests on every code commit and deployment to the test environment daily.

Automated Testing Framework ⇒

- Develop a comprehensive suite of automateed tests, including unit, integration, and UI tests.
- Prioritize writing tests for new features and regression tests for existing functionalities.

Test Environment Management ⇒

- Mantain a stable test environment that mirrors the production environment to ensure consistency in testing.
- Use containerization (Ex: Docker) to manage and deploy consistent test environments.

Bi-weekly Release Testing ⇒

- Before each bi-weekly release, perform extensive regression testing to validate the accumilated changes.
- Use automated tests to quickly identify and address any issues.

Test documentation and Monitoring ⇒

- Document test cases, test scripts, and test results to maintain a clear testing history.
- Monitor test coverage and update tests regularly to cover new functionalities and edge cases.

Continuous feedback and improvement ⇒

- Collect feedback from automated and manual tests to continuously improve the testing process.
- Conduct regular retrospectives to identify and implement process improvements.

5) Unit Testing

Type : automated tests that validate individual components / functions.

Level : unit level

benifit : defects and fixes bugs early in development cycle, ensuring that each unit of code works as intended.

example : writing unit tests for the 'addUser' function to verify that the users are correctly added to system.

Integration Testing

Type : automated or manual tests that validate the interactions between integrated components.

Level : integration level

benifit : ensures that different modules work together correctly, detecting issues in the interactions between units.

example : testing the workflow of adding a user and then assigning them to a course

System Testing

type : end-to-end testing of the entire system to validate that it meets the requirements

level : system level

benifit : ensures that the entire application works as expected in a production-like environment, covering ~~all~~ complete user scenarios.

Example : testing the complete user journey, from a student subscribing to a course, viewing assignments and submitting answers to ensure all functions work together.

Acceptance Testing

type : Manual or automated testing to validate the system against user requirements.

level : acceptance level.

benifit : confirms that the system meets the business requirements and ready for deployment.

example : conducting user ~~acep~~ acceptance testing with system users to ensure the system meets their requirements.

c) TDD (Test-Driven Development) -> tests are written before the code.

benifits :

Ensures code quality ⇒

- writing tests before code forces developers to think about the requirements and design before implementation, leading cleaner code.

Ex : writing a test case for 'addUser' function ensures that developers consider all edge cases and validation rules before implement method.

Early detection of bugs ⇒

- TDD allows for immediate detection of bugs as tests are run frequently during development.

ex : if developer introduces a bug in 'createAssignment' function, the pre-written tests will fail, highlighting the issue immediately.

Comprehensive test coverage ⇒
- TDD encourages writing tests for every functionality, leading to high test coverage and ensuring that most of the code is tested.
  Ex: tests written for user management, course management, assignment submission, ----, reducing the likelihood of untested code

Facilitates refactoring ⇒
- With tests in place, developers can change or improve the code confidently, knowing the tests will catch any new issues.
  Ex: if we need to change how assignments are evaluated, existing tests will ensure the changes don't break anything.

Improves design and documentation ⇒
- TDD leads to better-designed code & provides clear documentation on how the code should behave.
  Ex: tests show what each function is supposed to do

(04)

1) code coverage question

2) Example : Online shopping website.
Assume an online shopping application where users can apply discount codes. The system acceppts discount codes under the following conditions :
- Discount code length must be between 5-10 characters.
- Discount code should be alphanumeric.
- Discount code should not have been used before.

Equivalance Partioning (EP) ⇒ to divide input data into valid and invalid partitions.

valid partition : 5-10 characters long, alphanumeric, unused codes.

invalid partitions : less than 5 characters,
more than 10 characters,
non-aphanumeric characters
already used codes.

Boundry value Analysis ⇒ to test the boundries of the input data.

Valid boundry : exactly 5 and 10 characters.

Invalid boundry : 4 and 11 characters.

Equivalance Partioning (EP) helps identify general valid and invalid partitions.
Boundry value Analysis (BVA), ensures edge cases at the boundries are tested
By using both, we can ensure comprehensive coverage, since equivalance partioning finds major issues by testing representative values. and boundry value analysis catches errors that occur at the boundries.

3) Equivalance partitions

temparature > 80° F

temparature between 65°F - 80°F (inclusive)

temparature between 50°F - 65°F (inclusive)

temparature < 50°F

Boundry values

80° F and 81° F

65° F and 64° F

50°F and 49° F

optimal test cases ⇒

EP and BVA for >80°F --- test case 1 ⇒ 81° F (EP - above 80°F)

--- test case 2 ⇒ 80° F (BVA - boundry value)

Above 80° F → "wear light and breathable cloths"

65° F to 80° F → "wear comfirtable cloths"

50°F to 65° F → "wear a light jacket or sweater"

below 50°F → "wear a warm coat and layers"

For temparatures above 80° F

Test case 1 : 81° F ---> "wear light and breathable cloths"

For temparatures between 65° F and 80° F

Test case 2 : 80° F ---> "wear comfirtable cloths"

Test case 3 : 65° F ---> "wear comfirtable cloths"

Test case 4 : 72° F ---> "wear comfirtable cloths"

For temparatures between 50°F and 65°F

test case 5 : 65°F ---> } "wear a light jacket or sweater"
test case 6 : 50°F ---> }
test case 7 : 60°F ---> }

For temparatures below 50°F

test case 8 : 49°F ---> "wear a warm coat and layers"
test case 9 : 20°F ---> "wear a warm coat and layers"