Project Management

Tags

Useful Metrics for determining Software Quality:

White-box: After the software has been developed

- How much effort will be required for maintenance
- How much effort was used for development
- How vast is the test the testing effort (how easy is it to test?)

White box Metrics:

- **Lines of Code :**
  - Easy to compute
  - often sufficient for an approximate measure of size
  - Widely used

  However:
  - What defines a line ?

        - All lines of code are not equal, e.g. a system with lots of variable names and enums will have more lines of code than one without however will be more efficient.
        - Similarly, Object Orientated based programming languages will have a different metric for lines of code than functional coding like C

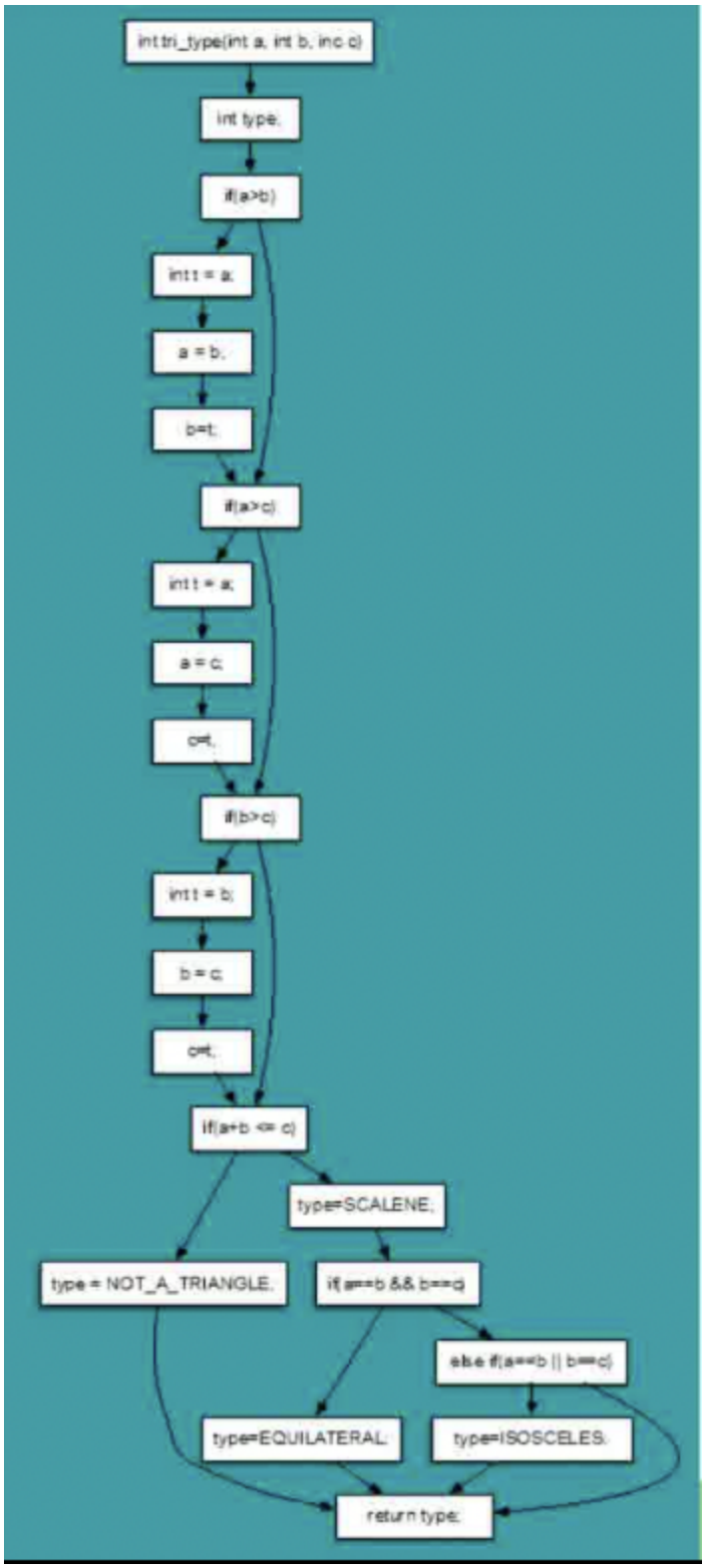
- **Cyclomatic Complexity :**
- Cyclomatic complexity is a software metric used to measure the complexity of a program. This metric, introduced by Thomas J. McCabe in 1976, quantifies the number of linearly independent paths through a program's source code. It is used to indicate the complexity of a program based on its control flow graph.
    - Calculated from the control flow graph :
        - V(G) = E - N + 2P
        - E = number of edges
        - N = number of nodes
        - P - Number of procedures (usually 1)

    example: Triangle Example “

```
int tri_type(int a, int b, int c) {
char[] type;
if (a > b) {
        int t = a; a = b; b = t;
    }
if (a > c) {
        int t = a; a = c; c = t;
}
if (b > c) {
    int t = b; b =c; c=t;
}
if (a + b <= c) {
 type = NOT_A_TRIANGLE;
}
```

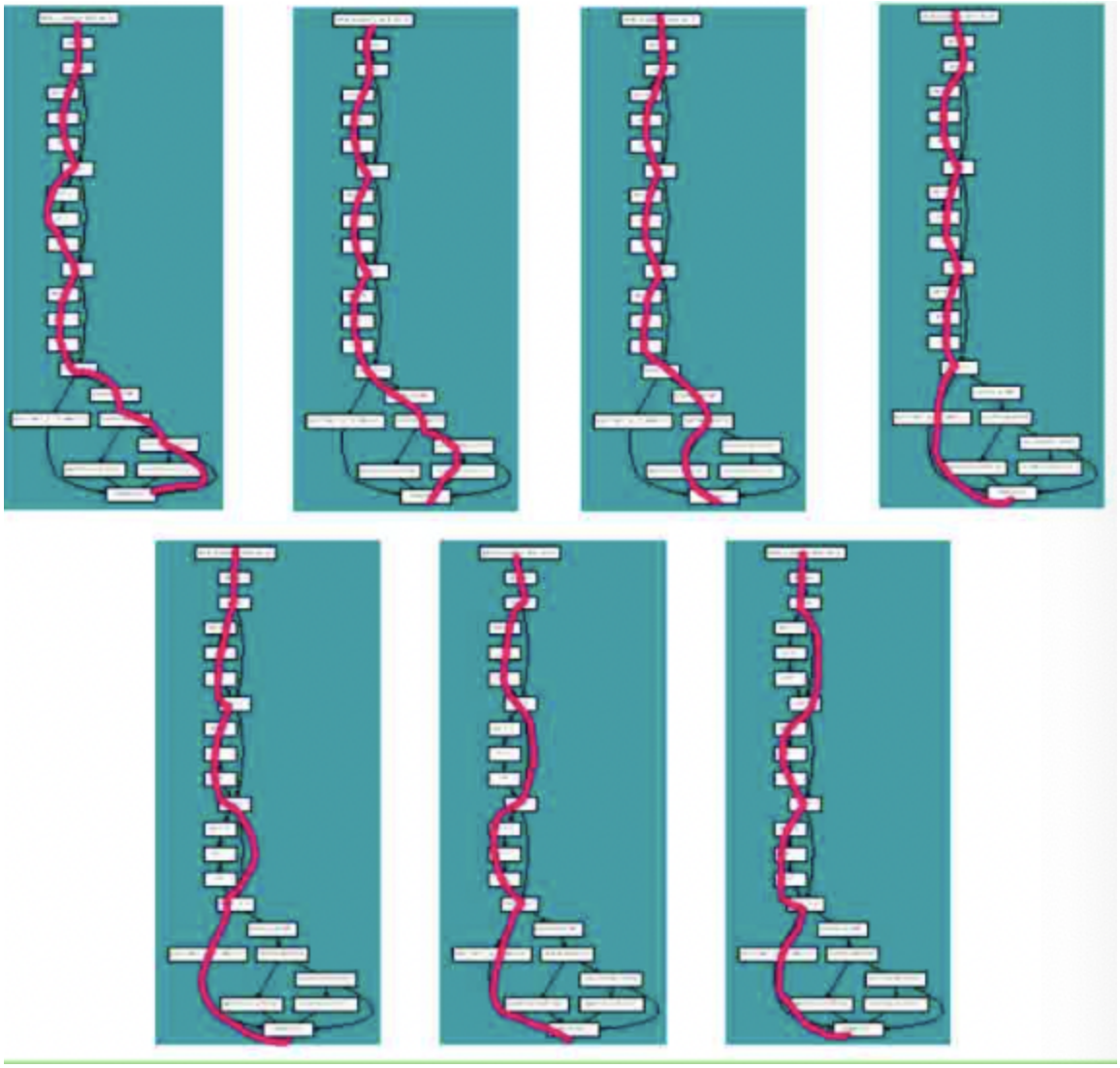
```
else {
type = SCALENE;
if (a== b && b == c) {
        type = EQUALATORAL
    }
else if (a == b || b == c) {
        type = ISOSCELES;
    }
}
return type;
}
```

- Here Nodes = 22 (white boxes) Edges = 27 (black arrows)


int tri_type(int a, int b, inc c)
int type;
if(a>b)
int t = a;
a = b;
b=t;
if(a>c)
int t = a;
a = c;
c=t;
if(b>c)
int t = b;
b = c;
c=t;
if(a+b <= c)
type=SCALENE;
type = NOT_A_TRIANGLE;
if(a==b && b==c)
else if(a==b || b==c)
type=EQUILATERAL;
type=ISOSCELES;
return type;

- P = 1

- V = 27 - 22 + 2(1) = 7. i.e. the different paths you can take or the cyclomatic complexity.
- **Meaning you would need at least 7 tests to cover all paths**



Example 2:

Consider the following Python function:

```
pythonCopy code
def example_function(x):
    if x > 0:
        if x < 10:
            return "x is between 1 and 9"
        else:
            return "x is greater than 9"
    elif x == 0:
        return "x is zero"
    else:
        return "x is negative"
```

Control Flow Analysis:

- There are four possible paths through this function:
  1. x>0x>0 and x<10 (true-true branch)

     x<10
  2. x>0x>0 and x>=10 (true-false branch)

     x>=10
  3. x==0x==0 (middle branch)
  4. x<0x<0 (false branch)

Calculating Cyclomatic Complexity:

- There are 3 decision points in the function: two 'if' and one 'else if'.
- Each decision point adds two edges (one for true, one for false).

Using the formula V(G)=D+1:

- D=3 (for three decision points)
- $V(G)$=3+1=4

Thus, the cyclomatic complexity of this function is 4, indicating that you would need at least four tests to cover all possible paths.

Why Measure Cyclomatic Complexity?

The cyclomatic complexity measure offers several benefits:

- **Predicting Maintainability**: Higher complexity scores suggest that a program is potentially harder to maintain and understand.
- **Identifying Risks**: Programs with high complexity are more prone to errors; thus, they are seen as riskier.
- **Testing Efforts**: It helps in determining the minimum number of test cases needed for thorough testing. Each path ideally should be tested once.

Black-box: Before the development process:

- Cost of the development process
- Programming intensity required

Black box Metrics:

Story Points:

- Story points are an arbitrary units of measurement that evaluate the amount of work required to implement a user story for a system.
- Other factors such as the complexity of the task, the risk involved and any other things that might impact the implementation of a user story is also contained within a story point measurement.
- **Benefits:**
    - **Relative estimation:** Story points are a relative measure of the amount of effort required based on a teams past experience
    - **Facilitates Planning**

- **Improves Accuracy Over time**

How are the Story points estimated?

- One method is called **Planning Poker.**
  - Here each team assigns a point value to each user story based on their understanding of the complexity of the task. The team discusses their estimation and reach a consensus.

Example:

Suppose a software development team has the following user stories to estimate:

- **Story A**: Add a login feature for the application.
- **Story B**: Integrate a third-party API to process payments.
- **Story C**: Create a dashboard to display user analytics from existing data.

After discussion and using the Planning Poker technique, the team might assign:

- **Story A**: 5 points due to its straightforward nature but important security considerations.
- **Story B**: 13 points because, while the concept is simple, the integration with an external system and handling of financial transactions add complexity and risk.
- **Story C**: 8 points because it requires creating new visualizations but utilizes existing data, making it moderately complex.

How Planning Poker Works

The process of Planning Poker typically involves the following steps:

1. **Gather the Team**: The development team, which usually includes software developers, testers, and sometimes the product owner, gathers for a planning session.
2. **Presentation of User Stories**: The product owner or project manager presents each user story or feature that needs to be estimated. This is crucial as it provides context about what needs to be accomplished.

3. **Initial Discussion**: Each story is briefly discussed to clarify doubts and ensure that all team members have a common understanding of what the task involves. This step is critical to make sure everyone is on the same page.
4. **Distribute Cards**: Each participant receives a set of cards, which are typically numbered according to the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21, etc.). These numbers represent the complexity and effort associated with the user story, though some teams use other sequences or scales.
5. **Individual Estimation**: Every participant selects a card that represents their estimate of the effort required for the user story. The selection is done privately to avoid influencing others' estimates.
6. **Reveal Cards**: All participants simultaneously reveal their selected card to the group. This simultaneous reveal helps to avoid any bias or influence in the decision-making process.
7. **Discuss Discrepancies**: If there is a significant spread in the estimates (e.g., if one developer thinks a task is very simple and another thinks it's very complex), the team discusses the reasons behind their estimates. This might involve explaining one's understanding or perspective of the task.
8. **Repeat Estimation**: After discussion, another round of estimation takes place, using the same procedure. This is repeated until the team reaches a consensus or a close approximation thereof.
9. **Finalise Estimation**: Once a consensus is reached on the user story’s points, it is recorded, and the team moves on to the next user story.

Benefits of Planning Poker

- **Team Collaboration**: Planning Poker involves every member of the team in the estimation process, fostering a collaborative environment.
- **Wide Perspective**: By involving multiple perspectives, teams can come up with more accurate and realistic estimations.
- **Avoids Anchoring**: The private selection of cards prevents team members from anchoring to the first number that is suggested, which could bias the estimates.

- **Consensus Building**: It encourages team members to discuss and agree upon the complexity of tasks, which can help in team bonding and alignment.
- **Fun and Engaging**: The game-like nature of Planning Poker can make the planning process more enjoyable and less tedious.

Software Laws: Patents, Copyrights, Contracts

Patent Law:

License issued by the government giving rights for a set period of time (20 years)

The invention must be:

- new
- inventive
- capable of industrial application