CODE QUALITY METRICS

Nicoleta Suciu
nicoleta.suciu@wayfare.ro
ABOUT ME

Name: Nicoleta Suciu
Job: Software Developer
Experience: 3 years
Zodiacal Sign: Aries
Height: …no idea
Why the code quality metrics are needed

Which are the metrics

How we can use them

Tools
good code is like a good joke
- it needs no explanation
Work That Frustrates Per Minute

Complexity metrics

Object-oriented metrics

Code quality metrics

Coverage metrics

Quality metrics
Complexity Metrics

- Cyclomatic complexity
- Essential complexity
Cyclomatic complexity

Measures the number of linearly independent paths within the source code.

Measures complexity at class or method level.

Based on graph theory.
```plaintext
statement1
if expression1 then
  statement2
else
  statement3
end
do
  statement4
  if expression2 then
    statement5
  else
    statement6
  end
while expression3
statement7
```
How to compute the cyclomatic complexity

\[ V(G) = E - N + 2 \]

Where: 
- \( E \) – is the total number of edges
- \( N \) – is the total number of nodes

Example:
\[ V(G) = 13 - 10 + 2 = 5 \]
Benefits of cyclomatic complexity

- Reduces the coupling of code
- Increases the code readability
- Ease of testing
Essential complexity

Cyclomatic complexity of the reduced control flow graph.

How to reduce the complexity:

- Validate and remove unwanted if statements
- Refactoring
- Remove unnecessary/redundant else conditions

$V(G) = 13 - 10 + 2 = 5$
Example

```
do iCnt = 1 to 10:
    if iCnt eq 4 then
        cFirst = cSecond.
end.
```

\[cFirst = cSecond.\]
Quality metrics

- Code quality metrics
  - Object-oriented metrics
  - Complexity metrics
  - Work That Frustrates Per Minute
- Coverage metrics
Quality Metrics

Average Percentage of Faults Detected

Fault Severity and Priority

Production Incidents
Average Percentage of Faults Detected (APFD)

Measures the rate of faults relative to the percentage of the test suite that is executed.

Needs prior knowledge of faults -> used only for evaluation.

Can be used with prioritized and non-prioritized test case.
How to calculate it

\[ APFD = 1 - \frac{TF1 + TF2 + \cdots + TFm}{n \times m} + \frac{1}{2 \times n} \]

Where:
- \( T \) is the test suite under evaluation
- \( m \) is the number of faults contained in the program
- \( n \) is the total number of test cases
- \( TFi \) is the position of the first test in \( T \) that exposes fault \( i \)
Example

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>F5</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No. of faults</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Time</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

APFD for non-prioritized test case:

\[
APFD = 1 - \frac{2+3+1+4+2}{5*5} + \frac{1}{2*5} \\
= 0.62
\]

APFD for prioritized test case:

\[
V = \text{fault/time} \\
APFD = 1 - \frac{1+3+4+1+1}{5*5} + \frac{1}{2*5} \\
= 0.7
\]
Fault Severity and Priority

Different types of failures, divided in two categories:

• Failure Severity
• Failure Priority
Severity vs Priority of a bug

I look only at the **Priority** field and so QA shouldn’t waste time writing down the **Severity**.

Yaa... 90% of the time Priority and Severity have the same value...

I think... QA just wants to exaggerate the severity of some Bugs

Developers
What is fault severity and fault priority?

- Severity is defined as the degree of impact a defect has on the system.
- Priority of a defect would indicate the urgency with which it would need to be fixed.
# Categorization of fault severity and priority

<table>
<thead>
<tr>
<th>Category</th>
<th>Class</th>
<th>Description</th>
<th>Category</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Severity</td>
<td>Critical</td>
<td>Complete shut-down of the system</td>
<td>Priority</td>
<td>High</td>
<td>Must be solved asap., the system cannot be used</td>
</tr>
<tr>
<td></td>
<td>Major</td>
<td>Severe defect that collapse the system, certain parts are still functional</td>
<td>Priority</td>
<td>Medium</td>
<td>Must be solved, but can wait until next release</td>
</tr>
<tr>
<td></td>
<td>Medium</td>
<td>Causes undesirable behaviour, but still functional</td>
<td>Priority</td>
<td>Low</td>
<td>Defect does not have any major impact</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>It won't cause any major break-down of the system</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Different levels of faults

- Key feature does not work
- Features that is rarely used does not work
- Company logo is spelled wrong
- Cosmetic errors
Who defines the fault severity and priority?

• QA classifies the defect under the appropriate severity based on the complexity and criticality of the defects.

• Any business stakeholders define the priority of the defects.
Another performance improvement metric is the number of production incidents or software crash rate.

It is imperative to record the initial cause of the defect along with what corrective measures were taken to solve it.

It traces the root problem that caused an incident.
Work That Frustrates Per Minute

Code quality metrics

Quality metrics

Coverage metrics

Complexity metrics

Object-oriented metrics
Object-Oriented Metrics

- Coupling Between Objects (CBO)
- Responses For a Class (RFC)
- Weighted Methods Per Class (WMC)
- Depth of Inheritance Tree (DIT)
- Number Of Children (NOC)
Coupling between objects (CBO)

An object is coupled to another object if one of them acts on the other.

Excessive coupling is detrimental to modular design and prevents reuse.

Measures testability, maintenance and reusability of a class.
Coupling between objects (CBO)

- CBO can be obtained through method calls, field accesses, inheritance, arguments, return types, and exceptions.

- CBO doesn’t care about the direction of dependency.
Coupling between objects (CBO)

- High CBO =>
  - Low modularity and reusability
  - Low maintainability
  - High complexity

- Used to track =>
  - Integrity of classes
  - Unnecessary interconnections in inappropriate places
Response for a class (RFC)

RFC measures the number of different methods that can be executed when an object of that class receives a message.

High RFC
- Testing and debugging of a class becomes complicated
- Higher the RFC, higher the complexity
Weighted Methods
Per Class (WMC)

WMC is defined to be the summation of all method’s complexity defined in a given class.

Method complexities can be considered to be unity, therefore WMC will be equal with the number of methods in a class.
Weighted Methods Per Class (WMC)

Indicator of class quality in an object-oriented environment

Predictor of how much time and effort is required to develop and maintain a class

High WMC value:
• Higher impact on the children
• More application specific, low reusability & maintainability
Example

We will use 2 different complexity measurements
1. Thomas McCabe’s Cyclomatic Complexity:
   \[ CC = E - N + 2P \]
2. Alternative:
   - total number of decision points + 1

Values:
Method foo:
1. CC = 7
2. CC = 2

Method bar:
1. CC = 5
2. CC = 2

WMC for class:
1. WMC = 12
2. WMC = 4
Depth of Inheritance Tree (DIT)

DIT provides for each class a measure of the inheritance levels from the class node to the root of the tree.

Measure of how many ancestor classes can potentially affect this class
Depth of Inheritance Tree (DIT)

Higher DIT => higher complexity, but greater potential for reuse

DIT value 2 => indicate improper use of object-oriented techniques

DIT can indicate if a design is ‘top heavy’ or ‘bottom heavy’
NOC refers to the number of immediate sub-classes subordinated to a class in the class hierarchy.

or

How many sub-classes are going to inherit the methods of the parent class
Number Of Children (NOC)

High NOC:
- Improper abstraction of parent class
- Misuse of sub-classing
- More testing for the base class
- High reuse for the parent class
Work That Frustrates Per Minute

Complexity metrics

Object-oriented metrics

Quality metrics

Code quality metrics

Coverage metrics
Coverage Metrics

- Function Coverage
- Statement Coverage
- Branch Coverage
Function Coverage

Has each function in the program been called?

Aims to measure the quantity of functions, methods, classes that are covered by the test suite.

Suited for testing the actual business requirements
TEST ALL THE THINGS
PRIORITIES

1.

2.

3.
Statement Coverage

Has each statement in the program been executed?

Covers only the true conditions in a method

Makes sure that all code blocks are executed
How to measure it

Statement coverage = \( \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100 \% \)

Higher this number => better exercised code

100 % statement coverage => not imply fully functional test
Example

1. READ X
2. READ Y
3. Z = X + 2*Y
4. IF Z > 50 THEN
5. PRINT “Z VALUE”
6. ENDIF

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Tests</th>
<th>Lines Covered</th>
<th>Statement Coverage Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>X = 2, Y = 3</td>
<td>1 TO 4, 6</td>
<td>83 %</td>
</tr>
<tr>
<td>Test2</td>
<td>X = 0, Y = 25</td>
<td>1 TO 4, 6</td>
<td>83 %</td>
</tr>
<tr>
<td>Test3</td>
<td>X = 47, Y = 1</td>
<td>1 TO 4, 6</td>
<td>83 %</td>
</tr>
<tr>
<td>Test4</td>
<td>X = 20, Y = 25</td>
<td>1 TO 6</td>
<td>100 %</td>
</tr>
</tbody>
</table>
Statement Coverage != Function Coverage

class src.CodeMetrics:
    method public static character Format(input iiValue as integer):
        define variable iDefaultValu as integer no-undo initial 42.

        if iiValue ne ? then
            iiValue = iDefaultValue.

        return substitute("The value is &l.", iDefaultValue).
    end method.
end class.
Statement Coverage != Function Coverage

Full Code Coverage Test

```java
@Test.
method public void Test():
    define variable cResult as character no-undo.

    cResult = CodeMetrics:Format(?).

    Assert:Equals("The value is 42.", cResult).
end method.
```
Statement Coverage != Function Coverage

Correct Tests

```java
@Test.
method public void TestDefaultValue(  ):
    define variable cResult as character no-undo.

    cResult = CodeMetrics:Format(?).

    Assert:Equals("The value is 42.", cResult).
end method.

@Test.
method public void TestGivenValue(  ):
    define variable cResult as character no-undo.

    cResult = CodeMetrics:Format(7).

    Assert:Equals("The value is 7.", cResult).
end method.
```
Branch Coverage

Has each branch of each control structure been executed?

Aims to measure how many branches in the code base have been executed during testing.

Covers both the TRUE and the FALSE conditions
How to measure it

Branch coverage = \( \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100 \) %

100 % branch coverage => imply full statement coverage

100 % statement coverage => does NOT imply full branch coverage
Example

1. READ X
2. READ Y
3. Z = X - 2*Y
4. IF Z < 0 THEN
5. PRINT “Z negative”
6. ENDIF

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Tests</th>
<th>Decision outcome exercised</th>
<th>Branch Coverage Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1-1</td>
<td>X = 20, Y = 15</td>
<td>‘Z &lt; 0’ -&gt; TRUE</td>
<td>50 %</td>
</tr>
<tr>
<td>Test1-2</td>
<td>X = 10, Y = 2</td>
<td>‘Z &lt; 0’ -&gt; FALSE</td>
<td>50 %</td>
</tr>
</tbody>
</table>

Test1-1 gives 100 % statement coverage

For 100% branch coverage -> both tests need to be run
Work That Frustrates Per Minute

Code quality metrics

Quality metrics

Coverage metrics

Object-oriented metrics

Complexity metrics
Work that Frustrate per Minute (CODE REVIEW)

Major impact on the quality of the code => one of the most powerful tools available

One of the biggest defect detection rate
Code Quality Measurement:
WTFs/Minute

Good Code

Bad Code

http://commodot.com
How can you control WTFPM?

- Regular code reviews
- Ongoing education
- Learning from mistakes
Tools...
SonarQube

Open source platform developed by SonarSource

OpenEdge pug-in developed by RiverSide Software

Supports 20+ programming languages

50+ official plugins
SonarQube

Includes quality gates status:
• Reliability
• Security
• Maintainability

Code quality metrics:
• Duplicated Code
• Lines of code
• Technical debt
• Code coverage
• Comments density
Glimpse of overall health:

Shows where you stand in terms of quality
Conclusions...
THANK YOU