FIT5171 Tutorial 8
Software complexity & metrics

Week 8–9, 2023

Please do try the questions before coming to the tutorial. Your active participation is the most important!

Weyuker's 9 properties have been proposed to evaluate software metrics. Some of the properties (for example, properties 1, 3, 4 and 8) are quite simple and intuitive. However, some other properties are a bit more complicated and needs further analysis.

In this tutorial we will pick two properties (5, 6) and one software complexity metric from each category (structure, testing and object-oriented) and **informally** prove whether the above properties hold or not. If not, give a counter example.

Structure For structure metrics, we choose the morphology metric Tree Impurity: $TIP = \frac{2(\#E-\#V+1)}{(\#V-1)(\#V-2)}$.

Testing For testing metrics, we choose the simple statement coverage metric C_0.

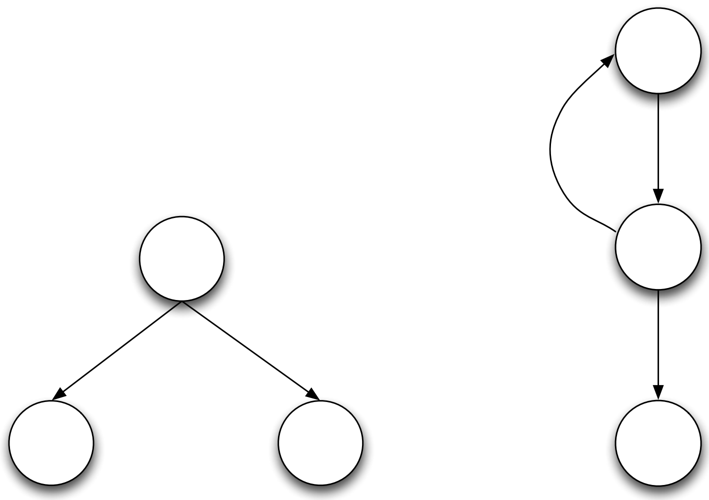
OO For object-oriented metrics we choose the metric Response For a Class: RFC, equal to the number of methods invocable.

We will restrict our discussion to a single language (Java or C#, for example) for simplicity. We also assume that program composition (+) can be either sequence or nesting.

1. Property 5: The complexity of a program segment should be that of the whole program, i.e., $\forall P, Q \bullet M(P) \leq M(P+Q) \land M(Q) \leq M(P+Q)$.

(a) Structure metric *TIP*.

Solution:
TIP measures how much a (program) graph deviates from a pure tree (in which each node has at most one parent and there is no cycle). Intuitively, the more deviated a graph is from a tree, the higher the *TIP* value is and the more complex the graph is.
Two example graphs are shown in Figure 1 below.



(a) Graph for P (b) Graph for Q

Figure 1: Two simple graphs with different *TIP* values.

Graph for P above is a pure tree and hence $TIP(P) = 0$. Graph for Q has an extra edge so $TIP(Q)$ is

$$TIP(Q) = \frac{2 * (3 - 3 + 1)}{(3 - 1) * (3 - 2)} = 1 \tag{1}$$

Solution: (continued)

However, if we compose the two program together sequentially, we will have what's shown in Figure 2.

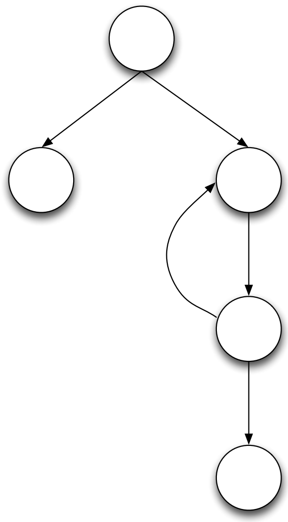


Figure 2: A sequential composition, $P + Q$, of the above two programs.

This graph has 5 nodes and 5 edges, hence the *TIP* value of the above graph in Figure 2, is then

$$TIP(P + Q) = \frac{2 * (5 - 5 + 1)}{(5 - 1) * (5 - 2)} = \frac{1}{6} \qquad (2)$$

Compare Formulas (1) and (2), we can see clearly that for this example, $TIP(Q) > TIP(P + Q)$. Hence property 5 does **not** always hold for *TIP*.

(b) Testing metric C_0.

Solution:
If we treat C_0 as a coverage criteria to mean 100% coverage, then clearly property 5 holds for C_0.
However, if we treat C_0 as a coverage metric that measures the percentage of the statements covered by testing, then it becomes more interesting.
For program P, we denote $C_0(P) = \frac{e_P}{L_P}$, where e_P denotes the number of statements covered by testing and L_P denotes the total number of statements of P. Similarly, we have $C_0(Q) = \frac{e_Q}{L_Q}$ and $C_0(P+Q) = \frac{e_{P+Q}}{L_{P+Q}}$.
We observe that $e_{P+Q} = e_P + e_Q$ and that $L_{P+Q} = L_P + L_Q$. Hence, $C_0(P+Q) = \frac{e_P + e_Q}{L_P + L_Q}$.
Let's assume for a particular program P, $C_0(P) = 100\%$ and it has 10 lines of code. Also assume that there exists a program Q with $10,000$ lines of code but coverage 0%. Hence, for the sequential composition of the two, $P+Q$, $C_0(P+Q)$ is then

$$C_0(P+Q) = \frac{10+0}{10+10000} = \frac{1}{1001} < C_0(P) = 100\%$$

Hence, the above counterexample shows that property 5 does **not** always hold for C_0.

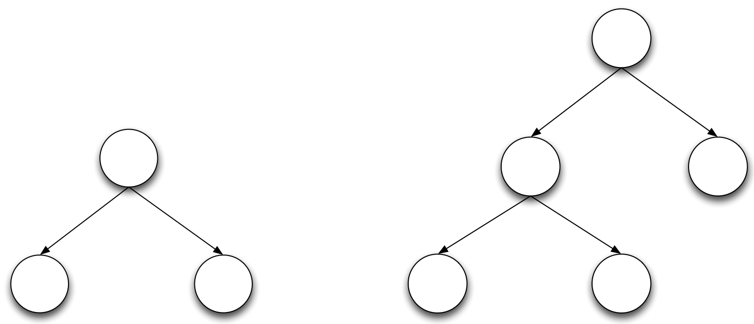
(c) OO metric RFC.

Solution:
RFC measures the complexity of a *class* by simply counting the number of methods in this class and all its super classes.
Because RFC measures the complexity of classes, both P and Q must be classes.
If we take sequence composition to be inheritance between classes, *i.e.*, $P+Q$ is class P with added super class Q. Then $RFC(P+Q) \geq RFC(P)$ and $RFC(P+Q) \geq RFC(Q)$, since the class $P+Q$ contains the set of methods that is the union of the sets of methods in both P and Q. Hence, if the composition is sequence composition, then property 5 holds for RFC.

2. Property 6: The complexity of the composition of two programs P and R may not be the same as the composition of programs Q and R, even though P and Q have the same complexity, i.e., $\exists\, P, Q, R \bullet M(P) = M(Q) \wedge M(P+R) \neq M(Q+R)$.

   (a) Structure metric *TIP*.

Solution:
We'll construct two simple trees for programs P and Q that have the same *TIP* value, 0.



(a) Graph for P (b) Graph for Q

Figure 3: Two simple graphs with the same *TIP* value of 0.

We'll now try to find a graph for program R to prove property 6. We'll try a simple 2-node graph:

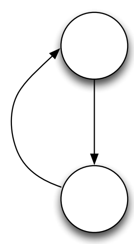
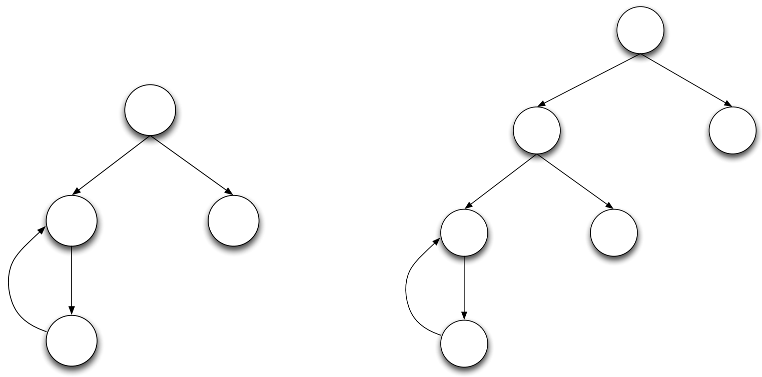


Figure 4: A simple graph for program R.

Solution: (continued)
With sequential composition on the lowest left node, $P + R$ and $Q + R$ then becomes larger graphs and not pure trees shown below in Figure 5.



(a) Graph for $P + R$ (b) Graph for $Q + R$

Figure 5: Graphs for programs $P + R$ and $Q + R$.

For the two graphs, their respective TIP values are:

$$TIP(P + R) = \frac{2 \times (4 - 4 + 1)}{(4 - 2) \times (4 - 1)} = \frac{1}{3}$$
$$TIP(Q + R) = \frac{2 \times (6 - 6 + 1)}{(6 - 2) \times (6 - 1)} = \frac{1}{10}$$

Apparently they're not equal, hence property 6 holds for TIP.

(b) Testing metric C_0.

Solution:
Let P be a program with 10 lines of code with 100% statement coverage. Let Q be a program with 5 lines of code also with 100% coverage. In other words, $C_0(P) = C_0(Q)$.
Let R be a program with 10 lines of code with 0% coverage. The sequential composition of R with P and Q then have coverage values as follows:

$$C_0(P + R) = \frac{10 + 0}{10 + 10} = 50\%$$
$$C_0(Q + R) = \frac{5 + 0}{5 + 10} = 33.3\%$$

They're obviously different. Hence property 6 holds for C_0.

(c) OO metric RFC.

Solution:
As the metric RFC is an object-oriented metric, we'll make programs P, Q and R classes and $+$ the inheritance operator (subclass). Then $P + R$ is R with P being a super class, similar for $Q + R$.
We'll define three Java classes to represent P and Q, as below.

P:
```
public class A {
  void a() {}
}
```

Q:
```
public class B {
  void b() {}
}
```

R:
```
public class C {
  void b() {}
}
```

Then, $P + R$ and $Q + R$ become

$P + R$:
```
public class C extends A {
  void b() {}
}
```

$Q + R$:
```
public class C extends B {
  void b() {}
}
```

By the definition of RFC, it counts all methods invocable in this class and all its super classes. Hence, the metric value for $P + R$ is 2 (`void a()` in `A` and `void b()` in `B`). The metric value for $Q + R$ is 1, since classes `B` and `C` have the same method declaration (`void b()`).
As a result, property 6 holds for RFC.