Maintainability

through

MHF and DIT

Hunter Smith and Grant Butler

What is Maintainability ?

- Maintainability focuses on designing and developing software systems that facilitate easy modification, enhancement, and debugging.

- Maintainable code must be:
  - Modular
  - Clear and Concise
  - Well Documented
  - Tested
  - Version Controlled
  - Refactorable

How does Maintainability affect a project?

Maintainability impacts the success and longevity of a project in many ways:

- Flexibility of a codebase, changing to different needs.
- Time-to-Market and faster development cycles.
- Quality and Reliability of code, as easier to maintain code is cleaner code.
- Developer Productivity and Satisfaction due to less time wasted on re-work.
- Collaboration and Teamwork with different teams, providing clear and modular code.

How can we measure Maintainability ?

Object Oriented Programming changes how we look at Maintainability

We must consider new paradigms such as encapsulation and inheritance

Two metrics can tackle these new paradigms:

Method Hiding Factor

and

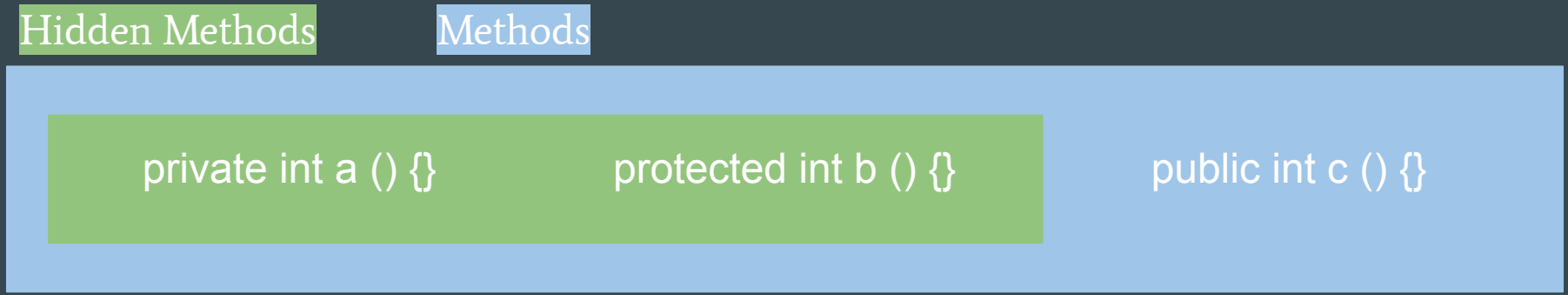
Depth on Inheritance Tree

What is the MHF?

MHF is an Object Oriented metric (Java, C++, etc.)

Represents the Encapsulation OO paradigm

The ratio of all hidden methods to total methods



What is the formula for MHF?

The accessibility of each method, for all methods, added together, divided by the total number of methods.

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{Md(Ci)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

where $M_d(C_i)$ is the number of methods declared in a class, and

$$V(M_{mi}) = \begin{cases} \text{if public:} & 1 \\ \text{else:} & 0 \end{cases}$$

What does MHF output?

The output will be a decimal ranging from 0 to 1

0: no encapsulation, all methods are public

1: complete encapsulation, all methods are hidden

Numbers in between represent how much a program is encapsulated

What does the output mean?

MHFs close to 0 represent bad coding practices

MHFs close to 1 represent low code functionality

The ideal MHF is between 0.08 and 0.25

Includes getter/setter methods, varies based on attributes

Why use MHF?

Encapsulation means lower coupling & less “points of access”

Reduces complexity and improves maintainability

As MHF increases...

- Defect density decreases
- Effort per defect decreases

MHF Tool

There are many open-source tools for computing MHF

I used the Classes and Metrics project on GitHub

Contains 48 Java metrics

https://github.com/yegor256/cam/tree/master

MHF Demo 1

```java
import java.util.Random;
public class AllPublic {
    public static class IntIncrementer {
        private int value;
        public IntIncrementer(int a) { value = a; }
        public void addInt() { value++; }
        public void subInt() { value--; }
        public void execute() {
            subInt();
            Random r = new Random(value);
            int rand = r.nextInt(value);
            for(int i = 0; i < rand; i++)
                addInt();
            System.out.println(value);
        }
    }
    public static void main(String [] args) {
        IntIncrementer intInc = new IntIncrementer( a: 25);
        intInc.addInt(); // CAUSES THE BUG
        intInc.execute();
    }
}
```

```java
import java.util.Random;
public class Encapsulated {
    public static class IntIncrementer {
        private int value;
        public IntIncrementer(int a) { value = a; }
        private void addInt() { value++; }
        private void subInt() { value--; }
        public void execute() {
            subInt();
            Random r = new Random(value);
            int rand = r.nextInt(value);
            for(int i = 0; i < rand; i++)
                addInt();
            System.out.println(value);
        }
    }
    public static void main(String [] args) {
        IntIncrementer intInc = new IntIncrementer( a: 25);
        intInc.execute();
    }
}
```

MHF Demo 2

AllPublic.java output:

```
31

Process finished with exit code 0
```

Encapsulated.java output:

```
43

Process finished with exit code 0
```

```
analyze("AllPublic.java")
MHF:  0.0

analyze("Encapsulated.java")
MHF:  0.5
```

3 public + 1 main = 0/4 private methods

1 public + 1 main = 2/4 private methods

MHF Paper

Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality by Subhas C. Misra and Virendra C. Bhavsar

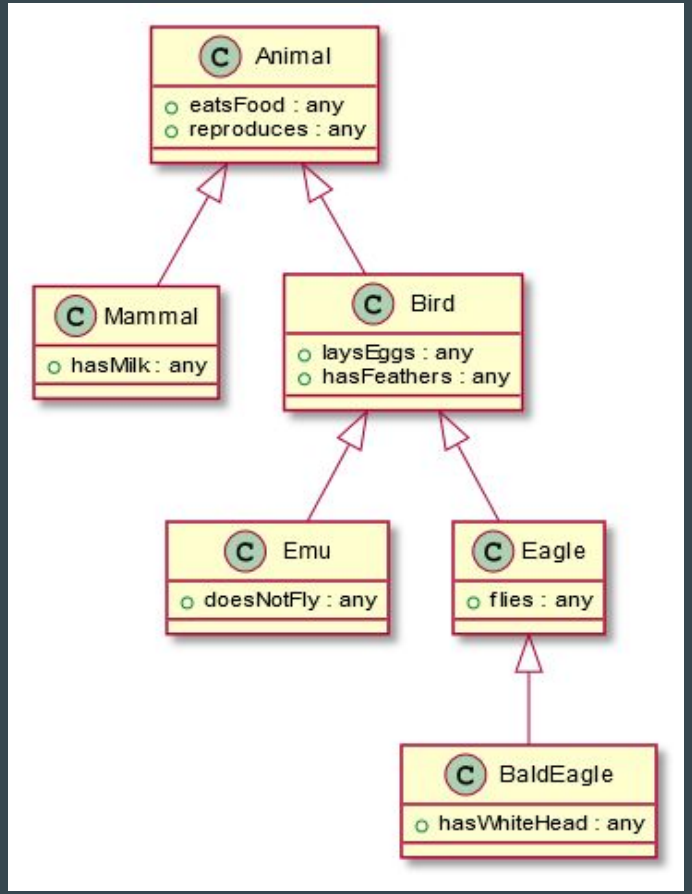
Analysis of over 30 OO C++ programs

“MHF... showed significant negative relationship with BD ... meaning that, an increase in ... method hiding ... should decrease the bug-density” [728]

What is DIT?

Depth on Inheritance Tree (DIT) is an Object Oriented measure of how far down in an inheritance tree a given module or class is.

- It helps measure:
  - Difficulty refactoring a class
  - Inheritance misuse
  - Coupling between classes
  - Dependency Management



How can we interpret DIT ?

- Classes that have DIT values of 1 or 2 are generally easier to understand and maintain.
- DIT values of 3 or greater start becoming hard to refactor and reuse.

When values of DIT go up... Maintainability goes down.

When values of DIT go down... Maintainability goes up.

How to calculate DIT ?

```
function calculateDIT(class):
    if class has no parent classes:
        # Class is not part of any inheritance hierarchy
        return 0
    else:
        maxParentDepth = 0
        for each parentClass in class.getParentClasses():
            # Recursively calculate DIT for each parent class
            parentDepth = calculateDIT(parentClass)
            maxParentDepth = max(maxParentDepth, parentDepth)
        # Depth of current class is one level deeper than the deepest parent class
        return maxParentDepth + 1
```

Demo for DIT

The DIT was calculated for two Python programs using Python Metrics Calculator.

Flask - A python framework for building web apps.	
Class Name	**DIT**
SecureCookieSessionInterface	2
FlaskGroup	2
View	1
JSONTag	1
MethodView	2
DefaultJSONProvider	2

yt-dlp - A youtube video downloading utility.	
Class Name	**DIT**
TurnerBaseIE	3
BilibiliBaseIE	2
BilibiliSpaceListBaseIE	3
PRXBaseIE	2
RaiBaseIE	2
RutubePlaylistBaseIE	3

Demo for DIT

The smaller DIT values for each class indicate that this module of the program is considered:

- maintainable
- simple and readable
- easy to test

yt-dlp - A youtube video downloading utility.	
Class Name	**DIT**
TurnerBaseIE	3
BilibiliBaseIE	2
BilibiliSpaceListBaseIE	3
PRXBaseIE	2
RaiBaseIE	2
RutubePlaylistBaseIE	3

Interesting Note: DIT as a Heuristic

“Yet, from the heuristic point of view, Figure 5(b) is usually easier to understand than Figure 5(a). Arguably, the hierarchy of Figure 5(a) is straightforward as compared to 5(b) yet, when this general trend is extrapolated the premise that 5(b) should be preferred becomes much stronger from the heuristic viewpoint.” (Sheldon)

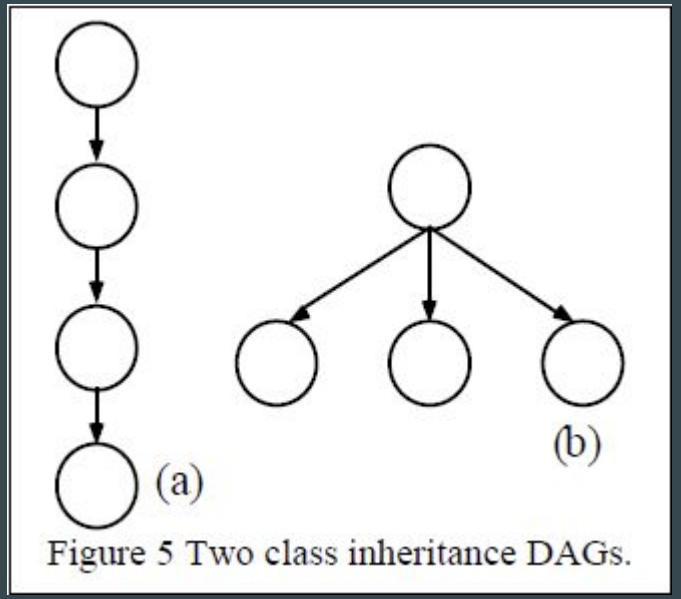


Figure 5 Two class inheritance DAGs.

Interesting Note: DIT as a Heuristic

AU of Figure 5(a): (1+2+3+4)/4=10/4=2.5

AM of (a): 2.5+(3/2+2/2+1/2)/4=3.25

AU of Figure 5(b): (1+2+2+2)/4=7/4=1.75

AM of (b): 1.75+(3/2)/4=2.13

∴ Sometimes, the DIT may be larger, but the class inheritance is less understandable !

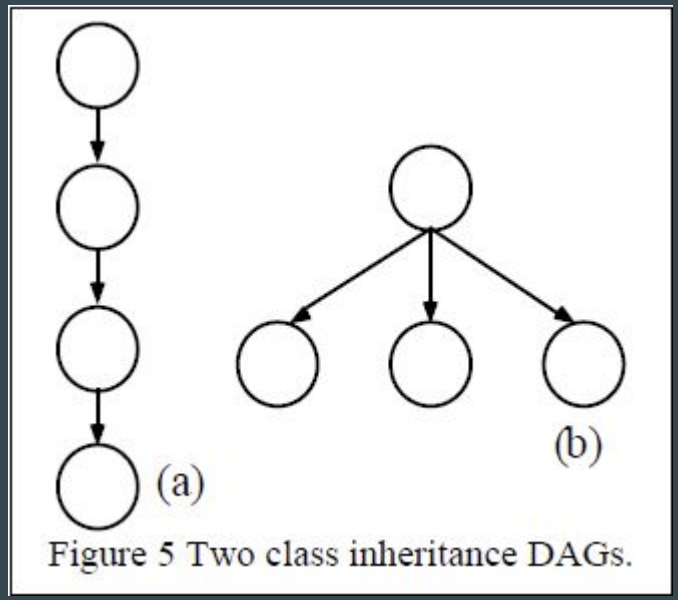


Figure 5 Two class inheritance DAGs.

Average Understandability = AU, Average Maintainability = AM

Papers, please. (DIT)

Sai, A. R., Holmes, C., Buckley, J., & Gear, A. L. (2020). Inheritance software metrics on smart contracts. *Proceedings of the 28th International Conference on Program Comprehension*. https://doi.org/10.1145/3387904.3389284

Sheldon, F. T., Jerath, K., & Chung, H. (2002). Metrics for maintainability of class inheritance hierarchies. *Journal of Software Maintenance and Evolution: Research and Practice, 14(3)*, 147–160. https://doi.org/10.1002/smr.249

QUESTIONS?

...