

Assignment 2 Final

by Nathan Morris

Submission date: 14-Jan-2023 12:57PM (UTC+0000)

Submission ID: 194987197

File name: 42098_Nathan_Morris_Assignment_2_Final_476351_1804581403.docx (97.93K)

Word count: 1436

Character count: 7287



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

University of Wales Trinity Saint David
School of Computing

ACCA7000

Programming and Software Testing

Java Programming, Testing and Development

By Nathan Morris (2222137)

GitHub account - FuzzyString

Contents

1. Abstract.....	3
2. Introduction	4
3. Main Report	5
3.1 High Cohesion	5
3.2 Loose coupling	5
3.4 Data hiding	6
3.5 Information hiding	6
4. Conclusion.....	8
Bibliography	9

1. Abstract

This report sets out to discuss the importance of high cohesion, loose coupling, data hiding and information hiding within software development. The principles all have a similar objective; software should be developed in a way where code can be easily read, understood and maintained. The modern world is utilising software and technology to perform tasks that humans would otherwise perform. As software becomes more complex, it is important that the principles are followed so that software can be maintained and further technological advances can be made.

2. Introduction

The report aims to discuss the importance of high cohesion, loose coupling, data hiding and information hiding within software development. Each principle will be discussed with relevant references made to academic literature where relevant. The report will then finish with a conclusion that offers an opinion on why these principles are important for the future of software and software development.

3. Main Report

3.1 High Cohesion

"When cohesion is high, it means that the methods and variables of the class are co-dependent and hang together as a logical whole." [1]

High cohesion as described by Roberts above is when classes are grouped together by functionality and functionality within classes is strongly related. High cohesion is important in software development and OOP in particular, code should be written in a transparent and organised manner which can be read and deciphered by engineers. Code written using this principle is easier to change and maintain. Code written within a class which contains strongly related functionality is easier to change and maintain than a class that contains many functions that are not strongly correlated. A developer can concentrate on the key functionality of a class rather than having to understand how changes to the functionality will affect code that is not related to that class. Roberts also states that classes should also be small for the same reasons. Classes will naturally be small if the coder writes code with high cohesion in mind, if a class starts to gain functionality which starts to deviate from the original class, it can be split out into another class.

An example of low cohesion can be found below

```
public class Calculator {  
  
    private int doubleInt = 2;  
  
    public int doubleNumber(int numberToBeDoubled) {  
        return doubleInt * numberToBeDoubled;  
    }  
  
    public int halfNumber(int numberToBeHalved) {  
        return numberToBeHalved / doubleInt;  
    }  
  
    public String returnNameAndAge(String name, int age)  
    {  
        return (name + age);  
    }  
}
```

The method returnNameAndAge is relatively unrelated to the other functions in the Calculator class. The function should be moved to a separate class that describes the function better for the code to adhere to high cohesion. Functions that are not correlated are harder for developers to find and follow, if this function is changed it is not clear if it will affect code in other classes or not.

3.2 Loose coupling

Coupling is defined as

"the measure of the strength of association established by a connection from one module to another." [2]

by Stevens et al. Coupling is the measure of how independent modules or classes are within a code structure. Steven et al. continue to state that strong coupling increases complexity and makes code harder to understand. Conversely loosely coupled code contains modules that are independent of each other who's functionality can be separated from one another. Loose coupling is important to software developing so that software can be split into small manageable components that are independent of each other. Where there are more components that are loosely coupled, there is a lower chance of a bug causing the failure of a whole programme as the bug will be contained to a specific component of code. Conversely if a programme is constructed with highly coupled code, there is a higher chance that the programme will fail as each component entwines and relies on other components. Loosely coupled code is more transparent and less complex, an engineer should be able to make changes to a module and predict the outcome of those changes. With a highly coupled architecture changes become more difficult as the code changes affect more modules and require more analysis on larger pieces of code

3.4 Data hiding

Data hiding is a concept where data that is not required by a user of software is hidden from the user. In the book *The Object-Oriented Thought Process*, Matt describes data hiding as

"The data hiding mechanism inherent with encapsulation is the means by which nonessential data is hidden from the user." [3]

The objective of data hiding is that particular code is hidden so that it cannot be accessed and modified without the relevant authorisation. The relevant code should be accessed via a function, with the underlying code or variables being out of bounds for a user. In Java code can be hidden by using the access modifier `private`. When `private` is used, the line of code can then only be accessed within the class that the code forms part of. An example of data hiding can be found below

```
public class Calculator {  
    private int doubleInt = 2;  
  
    public int doubleNumber(int numberToBeDoubled) {  
        return doubleInt * numberToBeDoubled;  
    }  
}
```

A Calculator class has been created with a function that doubles a number. The function can be accessed from any class, the variable used in the calculation to double an integer cannot be accessed or modified outside of the class. In this example, it is important that the variable remains private so that it cannot be modified. Any change in the variable will impact on the function's calculation. If the variable was accessible and changed from 2 to 3 the user would not get the doubling of a number which is the primary purpose of the function.

3.5 Information hiding

Information hiding was first introduced by David Panas in 1972 in his paper "On the Criteria to Be Used in Decomposing Systems Into Modules" [4]. In the paper David states

that code should be split into modules where the modules contain 'hidden' code that cannot be modified or changed by software users. All the code within each module is as independent as possible from all other modules. The importance of information hiding is the flexibility and ease of use that it gives to the programmer. If a change in the programme is required, rather than having to make many changes throughout a codebase, changes to a particular module can be made which impacts the output of the programme. The functionality for each module can remain hidden as each module only depends on output from a module, modules should not be concerned with the mechanics behind other modules and should not rely on them to work. A developer working on certain modules does not have to understand how every module works. As the modules are designed in such a way that the workings could be 'hidden', the developer only needs to understand the output from particular modules and how they impact the modules that they are working on.

4. Conclusion

Each four of the principles described in this report play a key role within software development. The principles all have a similar objective; software should be developed in a way where code can be easily read, understood, maintained and built on. The principles have played a part in software development for some time. However, in a world where technology is playing a greater role in society, software is becoming more complex and humans are relying more so on software advances to completed daily tasks (for example the recent emergence in AI in motor vehicles). Larger and more complex code bases require more Engineers to develop and maintain them, having code that is transparent and can be easily build on is more important than ever. It is therefore important that these principles continue to be followed.

Bibliography

- [1] R. C. Martin, Clean Code A Handbook of Agile Software Craftsmanship, Prentice Hall, 2009, p. 140.
- [2] W. M. G. a. C. L. Stevens, Structured Design, in Classics in, Yourdon Press, 1979, p. 209.
- [3] M. Weisfeld, The Object-Oriented Thought Process, Fourth Edition ed., Addison-Wesley, 2013, pp. 97 (ecopy - page might vary to physical text page number).
- [4] D. Panas, "On the Criteria to Be Used in Decomposing Systems Into Modules," *Carnegie-Mellon University*, 1972.

Assignment 2 Final

GRADEMARK REPORT

FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

Engagement with Literature Skills (15): 7

Knowledge and Understanding Skills (15): 11

Cognitive and Intellectual Skills (15): 11

Practical Application Skills (50): 40

Transferable Skills for Life and Professional Practice (5): 3

A reasonable report, though a little more detail wouldn't hurt - you have the word count left to play with. A modest use of sources, a slightly greater coverage wouldn't hurt.

A very good implementation, great use of testing, well designed and written. Splitting it into logical units is fine - I think your use of classes will evolve significantly in semester 2, however!

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

