© INNOVATIVE RESEARCH THOUGHTS | Refereed | Peer Reviewed | Indexed
ISSN : 2454 – 308X | Volume : 07 , Issue : 04 | October - December 2021

OBJECT ORIENTED MATRICS FOR TEST CASE SELECTION A STUDY

Shivneet Singh

Email id :- redhu.shivneet3@gmail.com

Abstract: A software crisis is a group of issues that have arisen during the process of creating computer programs. The failure to produce software on schedule, within budget, and meeting criteria is indicative of a software crisis. Producing high-quality software in the allotted time frame at a reasonable cost and in a way that fits the demands of the end user is the primary focus of software engineering. In this study, both classical and OO software metrics have been covered. Existing software matrix-related research has been discussed. The Levenshtein distance technique has been used to talk about slicing in this study. Applications of program slicing to certain software engineering tasks have been discussed in this work. The technique using the suggested distance mechanism has been explained. Both cases have developed slices. The corresponding matrix for that scenario has been constructed. There has been some discussion regarding the research's final results.

Keywords: Software Engineering, Software matrix, OOPS, WMC, DIT, Test case selection

[1] INTRODUCTION

A software project may be developed in a more orderly, methodical, and disciplined manner when software engineering principles are adhered to. One of the primary focuses of Software Engineering is increasing efficiency throughout the development and programming phases.

It's all about being able to plan and budget for future software projects. Creating products that meet the needs of the market. Reliable, efficient, and easy-to-understand software systems are also being developed. All aspects of the final software product benefit from this enhancement.

There is consensus that selecting test cases is the best way to eliminate extraneous or redundant data from tests. When the scope of a specific test case is large, it might have an effect on the SDLC's overall performance.

Importance of test case selection

Selecting the right set of test cases is essential to boosting a program's efficiency. It's been an essential part of our testing efforts. Testing software takes into account both time and resource limitations. An important part of any development process is optimizing the test suite.

[2] SOFTWARE METRICS

Software Metrics is a term used to measure the software items i.e. software product, software process, person involved in

software production or an organization such as data processing department.

Broadly software metrics are categorized into two parts:

1. Product Metrics
2. Process Metrics

Product metrics. are used to evaluate the program's characteristics. Reliability, functionality, performance, usability, cost, size, complexity, and style are all examples of product metrics.

Process metrics . are used to evaluate the method through which the software is obtained. Cost metrics, effort measurements, progress metrics, and reuse metrics are all types of process metrics. It's useful for checking whether a project is on track and for estimating the size of the finished system.

There have been several software development approaches such as

1) Function-oriented
2) Object-oriented
3) Component-based
4) Aspect-oriented

Traditional function-oriented metrics consists of following

1. Line of code (LOC)
2. Token count
3. Halstead's model
4. McCABE'S cyclomatic metric

Object-oriented metrics consists of following:

1. Weighted method per class (WMC)
2. Depth of inheritance (DIT)
3. Coupling between objects (CBO)
4. Lack of cohesion method (LCOM)
5. Number of children (NOC)

MOOD'S METRICS consists of following

1. Method hiding factor (MHF)
2. Attribute hiding factor (AHF)
3. Method inheritance factor (MIF)
4. Attribute inheritance factor (AIF)
5. Coupling factor (CF)

[3] TOOLS AND TECHNOLOGY

Matlab and SPSS software tools will be used for data analysis. Data will be collected from the academic environment like books, internet and will also be tried to get data from the software industry.

[4] OBJECT ORIENTED MATRICS

Metrics calculate various measures for projects, packages, types, members, and constructors. Metrics are guidelines for where something in the application might need refinement and changes. Metrics results can be saved to files in various formats, so they can be analyzed by spreadsheet tools, or, for example, sent via e-mail to the development team as part of the nightly build.

Weighted method per class (WMC)

All of the class methods' complexity levels are added together to provide the WMC metric. It's a measure of the

time and energy needed to create and sustain a certain kind. Class with a high WMC is difficult to reuse and maintain because of its complexity (application specificity). Considering that classes should have at least one function, RefactorIT sets the default lower limit for WMC to 1, and the maximum limit to 50.

Depth of inheritance (DIT)

The DIT measures how deeply a class is nested inside an inheritance tree, from the defined class all the way up to the superclass that declares all other superclasses. java.lang is a required package for all Java classes. Object as their highest-level superclass; this class is zero-deep. Accordingly, a class that directly extends java.lang. The metric value of the object is 1. Its value is 2, and the value of any of its subclasses is 2. For classes and interfaces, RefactorIT provides the following definition of DIT:

1. All interface types have a depth of 1
2. The class java.lang.Object has a depth of 0
3. All other classes have a depth of 1 plus the depth of their super class

Classes farther down in the hierarchy tend to inherit more methods and state variables, making it harder to anticipate how they will act. In this context, a DIT value of 0 implies a root, whereas values of 2 and 3 suggest a greater degree of reuse. Low DIT values may indicate that the benefits of object-oriented design and inheritance are being underutilized. For this reason, Refactor IT suggests capping the DIT at 5, since deeper trees represent higher design complexity due to the increased number of methods and classes.

Coupling between objects (CBO)

Coupling between objects (CBO) is a count of the number of classes that are **coupled** to a particular class i.e. where the methods of one class call the methods or access the variables of the other.

Lack of cohesion method (LCOM)

Lack of cohesion implies classes should probably be split into two or more sub-classes. Any measure of disparateness of methods helps identify flaws in the design of classes. Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

Number of children (NOC)

Number of Direct Subclasses of a Class, this metric measures the number of direct subclasses of a class. The size of NOC

approximately indicates how an application reuses itself.

[9] CONCLUSION

It is been concluded that more children a class has, the more responsibility there is on the maintainer of the class not to break the children's behavior. As a result, it is harder to modify the class and requires more testing. The upper recommended limit for a class in RefactorIT is 10 and the lower limit is 0. If NOC exceeds 10 children for a class, this may indicate a misuse of subclassing.

It becomes more specialized and it can be hard to understand a system with many inheritance layers. However, there is a greater potential reuse of inherited methods.

[10] SCOPE OF RESEARCH

The challenges related to data mining could be tackled by test case selection approach.

Here older approaches are not similar to modern one. These recognize only those components which are directly affected. On the other hand, slicing may recognize those down-stream components which are indirectly affected. It can also be determined by Slicing that two components have same implementation behavior or not.

REFERENCES

1. Pooja rana, Rajender singh " A study of component based complexity metrics" ,international journal of emerging research in mgt. and tech. , volume-3, nov-2014, pp. (159-165)
2. Mandeep walia, Anu gupta et.al "Software Metrics Usage & Research-Gaps and Future", IJSWS , mar-may 2015, pp. (01-10)
3. A.Aloysius and K.Maheshwaran "A Review on component based software metrics" , intern.J. Fuzzy Mathematical Archive,vol.7 , 2015 , pp.(185-194)
4. Parminder Kaur and Navdeep Batolar"A Review On Quality Assurance Of Component-Based Software System", IOSR-JCE, vol.17, jun 2015,pp.(53-57).
5. Sachin kumar, Pradeep Tomar et.al "Coupling Metric To Measure The Complexity Of Component Based Software through Interfaces " , vol.4, april 2014, pp. (157-162).
6. Jianguo chen,et.al,complexity metrics for component based software systems,international journal of digital content technology and its application, vol-5,nov 3,march 2011
7. Usha kumari and Sucheta Bhasin, A journey of software metrics:

traditional to aspect oriented paradigm, proceedings of the 5th national conference; INDIACom-2011

8. Latika Kharb, Rajender Singh: Assessment of Component Criticality with Proposed Metrics. Paper Accepted in INDIACom-2008: 2nd National Conference: Computing For Nation Development, Bharati Vidyapeeth's Institute of Computer Applications and Management. 08 - 09th February, 2008. New Delhi.
9. Nasib S. Gill, Component-based measurement: few useful guidelines, ACM SIGSOFT Software Engineering Notes, v.28 n.6, November 2003.
10. Eun Sook Cho, Min Sun Kim, Soo Dong Kim, "Component Metrics to Measure Component Quality," Asia-Pacific Software Engineering Conference, vol. 0, no. 0, pp. 419, Eighth Asia-Pacific Software Engineering Conference (APSEC'01), 2001.
11. Latika Kharb, Rajender Singh: Aspect-Oriented Software Engineering: A New Approach to Develop Secure Software. In INDIACom-2007-National Conference on "Computing for Nation Development sponsored by AICTE, IETE, & CSI. February 23--24, 2007.
12. Jianjun Zhao, "Measuring Coupling in Aspect-Oriented Systems", 10th International Software Metrics Symposium (METRICS'2004), Chicago, USA, September 14-16, 2004.
13. Nasib S. Gill, Few important considerations for deriving interface complexity metric for component-based systems, ACM SIGSOFT Software Engineering Notes, v.29 n.2, March 2004.
14. Navneet kaur and Ashima Singh “Component Complexity Metrics: A Survey” , inte, rn. J. advanced research in comp.sci and software eng. vol.3, june 2013 , pp. (1056-1061) .