Software Faults Emulation by Software Fault Injection

Soumya S
PG Student
Department Of Computer Science &Engineering
College of Engineering Perumon, Kollam

Baiju A S
PG Student
Department of Computer Science &Systems Engineering
Govt. Engineering College Painavu, Idukki

ABSTRACT

In Computer Science, an emulator is hardware or software or both that duplicates (or emulates) the functions of one computer system (the guest) in another computer system (the host), different from the first one, so that the emulated behavior closely resembles the behavior of the real system (the guest).The computer systems are an affected by software and hardware fault, solved in numerous mechanisms to handle. Fault injection is the method of testing such mechanisms, by providing artificial faults and errors (intending to mimic real faults and errors as closely as possible) in order to activate fault handling components. An emulator is object oriented software systems are used for fault emulation. In this paper first, the software components or classes are identified for fault injection by using the calculation is cohesion metrics. If the class is highly cohesive then that software component or class is used for fault emulation. In existing approaches cohesion measured from only structural information. Disadvantage is lack of high cohesion and lacking of measurement in cohesion. Here we propose unstructured information for cohesion measurement so achieving high cohesion and using this, accurate fault prediction can be performed.

Keywords

Cohesion, Fault injection, Fault emulation

1. INTRODUCTION

The high demand in any software organization process we are seeing for reducing the development cost and development time, decreasing error, increasing the software reliability in nowadays. These properties are satisfied software is good quality provided. But, software involves high complexity and constraints, so to develop and produce reliable software without faults is very difficult. This problem can be handled by predicting quality attributes such as fault proneness, maintenance and testing effort and reliability
during early steps of software development. The performing this paper efficient testing of the software is required. But, testing of the software means one of the activities where efficient resources are required. Since the software products are very large, it is not possible to test each and every class completely as it will not be cost-effective and also it will be very time needing process. So software error highly propagate then software's. Thus, we need to identify the classes where better test is needed. Then also we need to identify the classes which are more bad fault. There are various methods in which can be used to identify classes which are fault. The software fault identified most better method is used in software metrics. Software metrics have been proved as useful predictors of software fault badness. The models predicted using object oriented metrics can be used in early step of software development to predict faulty classes. They will help software practitioners and researchers to concentrate testing resources on the predicted faulty areas during software development. Thus, it will be significantly useful terms of saving time and resources during software development.

2. SOFTWARE FAULT INJECTION PROBLEMS

One of the biggest problem associated with software fault injection is that several fault injection techniques have been proposed and several fault injection studies are published. But, only few studies have addressed the problem of injection of software faults . In spite of the fact that the issue of how accurately the injected faults emulate real software faults remains largely unknown, fault injection has been used with success in several research works where software faults are the most relevant class of faults. Examples of software weaknesses revealed by faults injected at random can be found in. First it should be able to measure the cohesion metrics values such as LCOM5 and c3. Second should be able to identify the software components/classes where the faults can be injected. Third system should predict the fault proneness by using software cohesion metrics. These three properties are in order to perform some good results the software fault emulation system.

3. RELATED WORK

In existing OO systems several different approaches are used to measure cohesion. Most of the existing metrics are adapted from similar cohesion measures for non-OO systems, while some of the metrics are specific to OO software. On the basis of underlying information used to measure class cohesions, one can distinguish the following categories of metrics. They are:
- structural metrics
- semantic metrics
- information entropy-based metrics
- slice-based metrics
- metrics based on data mining and
- metrics for specific types of applications

The specific types of applications are knowledge-based, Aspect oriented and distributed systems.

Examples of different classes of structural metrics are:
LCOM1 (lack of cohesion in methods), LCOM2, LCOM3, LCOM4, Co (connectivity), LCOM5, Coh , TCC (tight class cohesion), LCC (loose class cohesion), ICH (information flow based cohesion).
By using the above metrics our major aim was to establish which structural metrics correlate well with the conceptual cohesion metrics. Our assumption is that if two metrics correlate well, then they measure the same aspects of cohesion. Therefore previous experiments show that LCOM1, 2, 3, and 4 are strongly correlated and thus they are measuring

similar properties of cohesion. This paper presented a new set of metrics for measuring the conceptual cohesion of classes.
The metrics are measured using semantic information embedded in the source code (i.e., comments and identifiers) via an IR approach.

4. BASIC IDEA ABOUT INFORMATION RETRIEVAL APPROACH

Basically information retrieval approach is used to measure cohesion of classes in OO systems. In order to control complexity OO analysis and design methods this approach try to decompose the problem addressed by the software development system into classes. High cohesion and low coupling are two important properties of a good software.
Both of these properties in the case of object oriented classes represents the design principles that aimed at reducing complexity of the system. The most suitable type of representing cohesion for a class is model cohesion ,such that the class represents a single, semantically meaningful concept. This is the type of cohesion we are trying to measure in our approach.
The software designers and the programmers should rarely think about a class as a set of method attribute interactions. Most of them think about the class as a set of responsibilities and these classes approximate the concept from the problem domain implemented by the class. This type of information is recorded in the source code through identifiers and comments. Analysis of this type of information is referred to as semantic information. Semantic information is useful for a wide range of software development and evolution tasks. Among the existing cohesion metrics for OO software LORM [Logical Relatedness of Methods] is the only one that uses this type of information to measure the conceptual similarity of the methods in a class, as determined by the representation of the class methods by a semantic network. LORM uses natural language processing techniques for the analysis needed to measure the conceptual similarity of
methods.
Here we propose a different approach that uses the same type of information and is based on a similar interpretation of cohesion. That means we use here an advanced information retrieval approach for analyzing and extracting semantic information which is previously embedded with the source code. Examples of some other IR method includes vector space model or a Bayes classifier. These IR methods can be used to support software maintenance tasks also. In proposed system we select LSI because we already have a positive experience in using it to address other software maintenance tasks such as concept location, identification of abstract data types in legacy source code, clone detection in software, and recovery of traceability links between software and documentation.

The basic concept of LSI in measuring the conceptual cohesion of classes is similar to some extent to our existing work. That means the source code under analysis is converted into a text corpus, such that from each method only identifiers and comments are extracted. Each method is a document in this corpus and LSI is used to map each document to a vector in a multidimensional space determined by the terms that occur in the vocabulary of the software. This representation is similar to that used in existing search engines such as Google (www.google.com). Once each method is represented as a vector, a similarity measure between any two methods can be defined as the cosine between their corresponding vectors. This similarity measure will express how much relevant semantic information is shared among the two methods, in the context of the entire system. By computing the degree of similarity between methods of a class we can determine whether a class represents a single semantic abstraction (or concept). This information is then correlated to a new measure of cohesion we call Conceptual Cohesion of Classes (C3).

A. *Representation of System*

With the IR based underlying mechanism, in order to define and compute the C3 metric, we use a graph based system representation, similar to those used to compute other cohesion metrics. We assume that an OO system as a set of classes C = {c1, c2,...,cn}. The total number of classes in the system C is n = |C|. A class has a set of methods. For each class ci ϵ C,M(c) = {m1, m2,.., mk} is the set of methods of class c. An OO system C is represented as a set of connected graphs GC = {G1,G2,.., Gn} with Gi representing class ci. Each class ci 2C is also represented by a graph Gi ϵGC such that Gi = (Vi, Ei), where Vi = M(ci) is a set of vertices corresponding to the methods in class ci and Ei ixVi is a set of weighted edges that connect pairs of methods from the class.

Definition for Conceptual similarity between methods (CSM):
For every class ci ϵ C all the edges in Ei are weighted.
For each edge $(m_k, m_j)\epsilon$, we define the weight of that particular edge CSM($m_{k,}m_j$), as the conceptual similarity between the methods $m_k$and m_j. The conceptual similarity between methods m_k and m_j, CSM(m_k, m_j) is computed as the cosine between the vectors corresponding to m_k and m_j in the semantic space constructed by the IR method (in this case LSI).

$$\text{CMS}(m_k, m_j) = \frac{vm_k^t vm_j}{|vm_{k2}|*|vm_{j2}|}$$

where vm_k and vm_j are the vectors corresponding to the $m_k, m_j\epsilon$ M(ci) methods. For each class c ϵ we have a maximum of N=Cϵn distinct edges between different nodes, where n = |M(c)|.

B. Lack of Cohesion in Methods5(LCOM5) Calculation

LCOM5 is purposed by Henderson-Sellers. Actually cohesion metrics can measure togetherness properly of methods of a class. A highly cohesive class provide good qualities. Lack of cohesion in methods can be calculated by using the formula:
LCOM5 = $a - kl/_{l - kl}$
Where l => number of attributes,
k => number of methods,
a => summation of the number of distinct attributes accessed by each method in a class.

C. The conceptual cohesion of classes(c3)

With this system representation we define a set of measures that approximate the cohesion of a class in an OO software system by measuring the degree to which the methods in a class are related conceptually.
Defintion for Average Conceptual Similarity of Methods in a class (ACSM):
The average conceptual similarity of the methods in a class c ϵC is:
ACSM(c)= $\frac{1}{N}\sum_{i=0}^{N} CSM(m_{i,}m_j)$

where $(m_{i,}m_j)$ ϵ, i, j, $m_{i,}m_j\epsilon$ M(c), and N is the number of distinct edges in G, defined in def. 1.

In our view, ACSM(c) defines degree to which methods of a class belong together conceptually and thus it can be used as basis for computing the conceptual cohesion of classes.

Definition for Conceptual Cohesion of a Class (C3)
For a class c ϵ C, the conceptual cohesion of c,C3(c) is defined as following:

C3(c)= ACSM(c) if ACSM(c)>0 else C3(c)=0
Based on the above definitions, C3(c) ϵ[0, 1] ϵ cϵC. If a class c ϵ is cohesive then C3(c) should be closer to one meaning that all methods in the class are strongly related conceptually with each other (i.e., the CSM for each pair of methods is close to one). In this case, the class most likely implements a single concept or a very small group of related concepts (related in the context of the software system).If the methods inside the class have low conceptual similarity values between them (CSM close to or less than zero), then the methods most likely participate in the implementation of different concepts and C3(c) will be close to zero.

D. ARCHITECTURAL DESIGN OF FAULT DETECTION SYSTEM

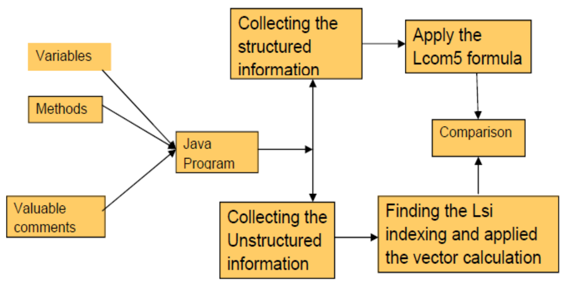
Variables
Methods
Valuable comments
Java Program
Collecting the structured information
Apply the Lcom5 formula
Comparison
Collecting the Unstructured information
Finding the Lsi indexing and applied the vector calculation

Fig. 1. The proposed framework

Our proposed system describes LCOM5 and C3 measure in more detail. First we develop the LCOM5 formula and find out the C3 measure and compare with structure and unstructured data. If the result is equal to one means, the class is less cohesive according to the structured information. Then we are going to retrieve the index terms based on that comments which are present in all the methods. Comments are useful information according to the software engineer.

In concept oriented analysis we are taking the comments.

Based on the comments we are going to measure the class is cohesive or not.

Step-1:
In this Phase we are going to take the structured information like identifiers, (Example Variables).Invocation of declared methods and declared constructors. Here the Java program should be well compiled and it should be valid comments.

Step -2:
In this Phase deals we are going to search the declared variables among all the classes. Because the main theme of the declaring class variable is, it should be used in all methods. So that the declared variables are found among all the methods.

Step -3:
In this Phase we are going to apply the LCOM5 (Lack of cohesion in methods) formula. If the result is equal to one means, the class is less cohesive according to the structured information.

Step -4:
Here we are going to retrieve the index terms based on that comments which are present in all the methods. Comments are useful information according to the software engineer. In concept oriented analysis we are taking the comments. Based on the comments we are going to measure the class is cohesive or not.

Step -5:
In this Phase we are going to check the index terms among the comments which are present in all the comments.

Step -6:
In this Phase we are going to apply the conceptual similarity formula. Based on the result we can say the class is cohesive or less cohesive according to concept oriented.

Step -7:
In this Phase we are going to compare the two results. Based on the results we can say that cohesion according to structure oriented and unstructured oriented.

5. CONCLUSIONS

Classes in object-oriented systems, written in different programming languages, contain identifiers and comments which reflect concepts from the domain of the software system. This information can be used to measure the cohesion of software. To extract this information for cohesion measurement, Latent Semantic Indexing can be used in a manner similar to measuring the coherence of natural language texts. This paper defines the conceptual cohesion of classes, which captures new and complementary dimensions of cohesion compared to a host of existing structural metrics. Principal component analysis of measurement results on open source software applications statistically supports this fact.

6. REFERENCES

[1] J.A. Duraes and H.S. Madeira. Emulation of software faults: A field data study and a practical approach. Software Engineering, IEEE Transactions on, 32(11):849867, Nov. 2006.

[2] Marco Serafini, Andrea Bondavalli, and Neeraj Suri. Online diagnosis and recovery: On the choice and impact of tuning parameters. IEEE Trans. Dependable Secur. Comput., 4(4):295312, 2007.

[3] Alessandro Daidone, Felicita Di Giandomenico, Andrea Bondavalli, and Silvano Chiaradonna. Hidden markov models as a support for diagnosis: Formalization of the problem and synthesis of the solution. In SRDS 06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems, pages 245256. IEEE Computer Society,2006.

[4] Silvano Chiaradonna, Domenico Cotroneo, and Luigi Romano. Effective fault treatment for improving the dependability of cots and legacy-based applications. IEEE Trans. Dependable Secur. Comput., 1(4):223237, 2004. Member-Andrea Bondavalli.

[5] J. Gray. Why Do Computers Stop and What Can Be Done About It? Proc. Of Symposium on Reliability in Distributed Software and Database Systems, 1986.