Feature-Based Test Focus Selection Technique Using Classes Connections Weight

Iyad Alazzam, Yarmouk University, Irbid, Jordan
Mohammed Akour, Yarmouk University, Irbid, Jordan
Shadi Banitaan, University of Detroit Mercy, Detroit, MI, USA
Feras Hanandeh, Hashemite University, Zarqa, Jordan

ABSTRACT

Testing could cost more than fifty percent of all development cost, particularly integration testing consumes around eighty percent of testing cost. Integration testing aims to discover errors in the connections among classes which are collaborate and communicate in order to provide specific services. Though, testing all connections among classes is impractical because of the cost, effort and time constraints. Test focus selection might help testers to concentrate on the main and vital connections among classes which it could be the most error prone ones. The authors proposed approach amalgamates the static and dynamic analysis in order to detect, trace, and weight the connections among classes through method level communications. Their approach harnessed an open source tracing tool (MUTT). The MUTT allows them to return all the methods in all classes that have been called respecting to any specific feature which has triggered by the system user. The experimental results reveal how the proposed approach achieves good mutation testing score on the systems under study.

KEYWORDS

Class Connection, Feature, MUTT, Test Selection

INTRODUCTION

Software testing is a practice and procedure of running software against test cases in order to reveal and discover errors. Software testing is one of the main processes in the software development process, it is considered one of the vital factors that achieve and enhance quality assurance. Software testing is so costly; the testing cost consumes about fifty percent of the development cost (DeMillo, Lipton, & Sayward, 1978). Software testing has three main levels, unit testing, integration testing, and system
testing. Unit testing seeks to ensure that each module or class individually in the system under test is working as it should be without any problem; unit testing is usually done by the developers. Integration testing aims to ensure that the connections among two or more classes or modules are valid and the components are interacting with each other properly.

Software S is a system written using object oriented language. S consists of classes \( C=\{c_1, c_2, \ldots, c_n\} \) where \( n=|C| \). A class \( C \) consists of methods. For every class \( c \in C \), let \( M_c=\{m_1, m_2, \ldots, m_i\} \) is the group of implemented methods in class \( c \), where \( i=|M_c| \) is the total number of methods in class \( c \). The set of the whole methods in software \( S \) is identified as \( M_S \). A feature is a functional requirement. Service, action or functionality explained in the user specification of particular software. A software \( S \) consists of a set of features \( F=\{f_1, f_2, \ldots, f_n\} \) where \( n=|F| \). A feature \( f \) is constructed by collections of methods where \( M_f \subseteq M_S \). Feature \( f \) is represented by collection of methods \( M_f \) in the physical source code.

A method could be related to many classes, a method could be related to many features, and a feature could include methods that are related to other different features. The types of errors that are revealed at the unit level are different than the types of errors that are revealed at integration error. System testing tries to validate the system as a whole is working correctly and usually it focuses on the properties of the system. Integration testing considered being from the most expensive in the software area testing the cost of integration testing is almost eighty percent of the total cost of the testing process. Integration testing is very important in increasing the quality of the software since the majority of errors is discovered are integration testing level, forty percent of the errors are found at integration level of testing. Integration testing mainly focuses on interfaces and the connections among modules and classes. Testing all connections and interfaces at the same level is indisputable not practical and not worthy especially if the system under test is huge and has huge number of connections. Therefore, we propose new approach in integration testing to enhance the process of integration and testing by assessing and evaluating the connections among classes or modules, and focusing on the connections which have the highest strength. The connection with high strength tends to be error prone more than the connection with the low strength. Three common types of automated tests are located in the literature, 1) unit test, where we test only a solo of the source code (i.e. module, class), to double check that whether its behavior is expected and desirable. 2) Integration test, where we test the communication and collaboration between software modules and components. 3) System test, where we test the whole system. Although unit testing assures the components in isolation are functioning correctly, the process of components integration might introduce new errors. However, testing all interaction between components is costly and time consuming. Determining the most error prone connections is not an easy duty, moreover, one of the general dilemmas in inter-class integration testing of object-oriented software is to decide the order in which classes are integrated and tested (Sommerville, 2010).

Pelliccione et al. (Pelliccione, Muccini, Bucchiarone, & Facchini, 2005) proposed an approach refer to as TeStor. The TEst Sequence generaTOR algorithm allows extracting test sequences from both state machine and scenario diagrams. The state diagrams provide the behaviors of component while the sequence diagrams determine what the included test is. The outcome of their approach is the test sequences, the test sequence generation process in TeStor does not compute the parallel composition of the state machine models, it does not require any additional formalism, and it is completely tool supported.

In integration testing and in order to execute some class’s methods, the presence of other classes is mandatory; the dependencies relationship between classes needs to be tested to make sure that the communication are fault free. Having more and more interaction between software modules will lead eventually to increase the complication of integration testing task, time and resources needed to complete integration test. The class integration and test order (CITO) problem are investigated and addressed by several researchers (Abdurazik, Offutt, 2007; Abdurazik & Offutt, 2009; Lionel, Briand, & Labiche, 2002). The coupling measurement is harnessed to picture class dependency relationships and predicates the best integration test order. Test data selection criterion serves as the
Internationalization of Business in Emerging Countries and National Culture: The Brazilian Case
Fabrício Fernando Foganhole dos Santos and Irene Kazumi Miura (2012). Cultural Variations and Business Performance: Contemporary Globalism (pp. 117-164). www.igi-global.com/chapter/internationalization-business-emerging-countries-national/63913?camid=4v1a

Quantum-Behaved Particle Swarm Optimization Based Radial Basis Function Network for Classification of Clinical Datasets