

Chapter 93

Enhancing Ability of Fault Detection for Component Systems Based on Object Interactions Graph

Fuzhen Sun, Lejian Liao, Jianguang Du and Guoqiang Li

Abstract Test case prioritization is a technique to schedule the test case in order to maximize some objective function. Early fault detection can provide a faster feedback generating a scope for debuggers to carry out their task at an early stage. In this paper, a method is proposed to prioritize the test cases for testing component dependency in a Component Based Software Development (CBSD) environment using Greedy Approach. The OIG (Object Interaction Graph) is traversed to calculate the total number of inter component object interactions and intra component object interactions. Depending upon the number of interactions, the objective function is calculated and the test cases are ordered accordingly. This technique is applied to the components developed in Java for a software system and found to be very effective in early fault detection as compared with non-prioritize approach.

Keywords Test case prioritization · Fault detection · Object interaction graph (OIG) · Component based software development (CBSD)

93.1 Introduction

A technique like test case prioritization has to be devised, which will lead to early fault detection. Test case prioritization aims at finding an execution order for the test cases which maximizes a given objective function. Among the others, the most

F. Sun (✉) · L. Liao · J. Du · G. Li

Beijing Engineering Research Centre of High Volume Language Information Processing & Cloud Computing Applications, Beijing Key Laboratory of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing, China
e-mail: 10907023@bit.edu.cn

F. Sun

School of Computer Science and Technology, Shandong University of Technology, Zibo, China

important prioritization objective is probably discovering faults as early as possible, which refers to maximize the rate of fault detection.

The major challenges in Component Based Software Development (CBSD) are testing component dependency. CBSD uses the reusable components as the building blocks for constructing the complex software system (component based system). Component based system promotes the software quality and productive. This building block approach has been increasingly adopted for software development, especially for large-scale software systems.

Previous work on test case prioritization [1–5] is based on the computation of a prioritization index, which determines the ordering of the test cases (e.g., by decreasing values of the index) [6, 7]. Srivastava [8] suggested prioritizing test cases according to the criterion of increased Average percentage of Faults detected (APFD) value. Rothermel et al. [9] have described several techniques for test case prioritization and empirically examined their relative abilities to improve how quickly faults can be detected by those suites. More importance is given to coverage based prioritization here [10, 11].

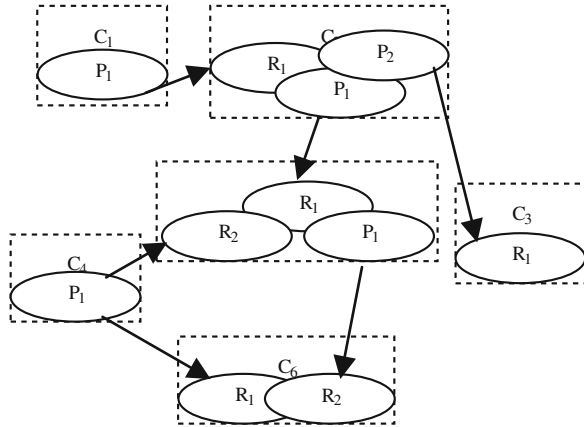
In this paper for describing each component we have taken the help of sequence diagrams, then a Object Interaction Graph (OIG) from sequence diagrams is constructed which shows the interrelation among the components. A new test prioritization algorithm is presented which is applied on OIG to count the maximum number of inter component interactions and intra component interactions made by the test cases.

93.2 Proposed Test Case Prioritization Model

In CBSD Component interface is defined as the only way that a component communicates with the external environment. There are two kinds of interface: service providing and service required. When the services are provided by an interface, it is called service providing interface and when the interface of a component requiring a service it is called service required interface. All components should be plug-compatible i.e. a service required interface can be connected to a service providing interface. We have defined a Component as follows: Component $C = (P, R)$, where $P = P_1, P_2, P_n$ is the set of providing services interface, $R = R_1, R_2, R_m$ is the set of required services interface. The providing and required services of a component C is denoted by $C.P$ and $C.R$ respectively and $C.P \cap C.R = \emptyset$.

In Fig. 93.1 the required services of $C_1 P C_2$ are the union of $C_1.R_1$ and $C_2.R_2$ with the remove of satisfied services in S . With the definition of composition the providing and required services are propagated to the interface of composed component, so the composition could be carried parallel. A Component interaction graph (OIG) is used to describe the interrelation of components. A complete component interaction graph (OIG) makes the testing quite easy.

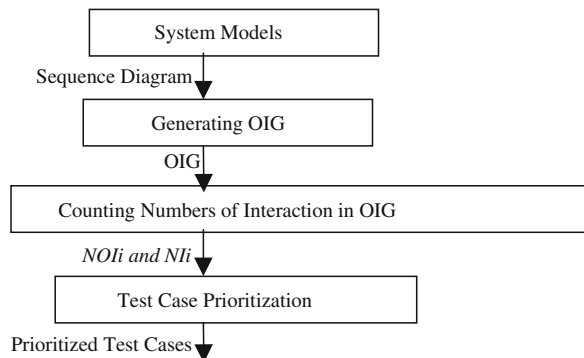
Fig. 93.1 Object interaction graph(OIG)



To facilitate regression testing by optimizing the time and cost, we propose a method to prioritize the test cases by using model based prioritization method by extracting the benefits of Unified Modelling Language (UML). UML provides lifecycle support in software development and is widely used to describe analysis and design specifications of software. It is a big challenge to study the test case generation from UML diagram (Fig. 93.2).

We have used sequence diagram from the set of diagrams present in UML 2.0. As Sequence diagram represents various object interactions through message passing, it can act as an input to the proposed model. We are generating an OIG from the sequence diagrams present. The methodology we have used for generating the graph has been discussed in Sect. 93.2.1 Further in Sect. 93.2.2 we have discussed how to traverse the OIG to calculate the number of inter component object interaction and intra component object interaction. Section 93.2.3 describes about objective function evaluation and the prioritization technique.

Fig. 93.2 A frame work for generating prioritized test cases



93.2.1 Generating OIG Form System Models

We have used sequence diagram for system modeling, and the object interactions can be very well identified by using a sequence diagram. During regression testing any modification in the code will have no effect on the sequence diagram. The object interaction can be categorized into two different types. One of them is intra component object interactions and the other one is inter component object interactions.

Sequence diagrams in UML are used to model how an object communicates with other objects in its life time. A complete object interaction graph (OIG) makes the testing quite easy. An OIG is a directed graph where $OIG = (V, E)$, V represents a set of nodes. For generating Object Interaction Graph (OIG), each object present in the sequence diagram is represented as a node in the graph. The intra component object interactions form the edges of the graph and represented in Solid arrows. The inter component object interactions form the edges of the graph and represented in Dashed arrows.

Algorithm: GENERATE OIG

Input: Sequence Diagrams of various components of the system representing message passing between objects

Output: Object Interaction Graph (OIG)//It is a directed graph

1. Initialize OIG to be empty
2. For $i = 1$ to n/n is the total number of objects
3. Add a node N_i to OIG == N_i represents i th node.
Object shared by different components treated as a single node.
4. For $i = 1$ to n
5. For $j = 1$ to n
6. For each incoming message from object O_i to O_j == All guard conditions are ignored
7. if (interaction types == intra) Establish an edge between O_i to O_j (i.e. N_i and N_j) and represent it as "Solid arrow" as well as append the pre and post conditions.
8. Else Establish an edge between O_i to O_j (i.e. N_i and N_j) and represent it as "Dashed arrow" as well as append the pre and post conditions.
9. The possible start and end of the scenario sequences are represented with solid arrows.

93.2.2 Traversing OIG

When the OIG is generated from the system models, it has to be traversed to count the number of inter component and intra component object interactions. NOI_i represents the number of Object Interactions discovered by test case t_i with in one component of the software and NI_i represents the number of Object Interactions

discovered by test case t_i between two different components of the software. We follow the depth first search (DFS) methodology for traversing the graph. The type of interaction is decided depending upon the color of the edge in the graph. If the edge color is found to be “Solid arrow”, it represents an intra component object interaction, where as edges colored as “Dashed arrow” represents inter component object interaction

Algorithm: IN_CALCULATE

Input: Test case t_i & Object Interaction Graph (OIG)

Output: NOI_i and NI_i

1. Initialize both NOI_i and NI_i to 0.
2. Traverse each interaction in the OIG for t_i in DFS
3. If (edge color == ‘VISITED’ && current edge is not visited already)
4. $NOI_i + +$ // Increment the value for intra component interaction
5. Else
6. $NI_i + +$ // Increment the value for inter component interaction
7. Return NOI_i and NI_i .

93.2.3 Generating Prioritized Test Cases

Once we get the value for NI_i and NOI_i by using the algorithm described in Sect. 93.2.3, prioritization process starts. For each test case t_i , the value of NI_i and NOI_i are added. We have considered the total number of intra component interaction where as the total number of inter component object interactions is found out by multiplying it with RP i.e. total number of providing service interface and required service interface. If the faults due to component integration are detected early, it will give a better coverage. The added result is divided with unit time U to determine value of the objective function i.e. factor criteria FC_i . We try to maximize the objective function using a Greedy approach.

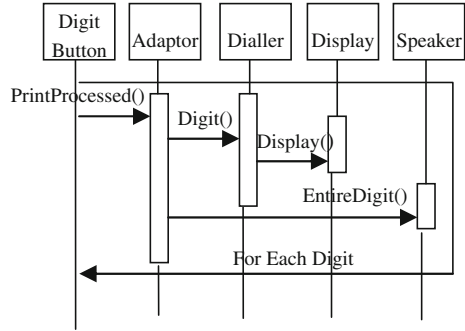
Algorithm: TEST_PRI

Input: Regression Test Suite T

Output: Prioritize Test Suite T’

1. Traverse the test suite T, for each test case t_i present, call **IN_CALCULATE** (t_i) to calculate NOI_i and NI_i
2. Define some unit time U
3. Calculate objective function (FC_i) for test case t_i as $FC_i = (NOI_i + RP*NI_i)/U$. (1)
//RP represents total number of providing service interface.
4. Generate T’ by Sorting the test suit T in ascending order of FC_i for each t_i .
5. Store T’ in the test case repository for regression testing.

Fig. 93.3 Sequence diagram for dialing the number



93.3 Case Study: A Cellular Network Manager

We have taken the case study of a Cellular Network Manager to explain the proposed model. Two components i.e. “Dialing a Phone” and “Cellular Network Connection” have been presented.

From the sequence diagram of both the components given in Figs. 93.3 and 93.4, corresponding OIG are designed as given in Fig. 93.5.

Three test cases are considered to test the prioritization algorithm. The test cases are designed to test the Dialer Display (t1), to test the Speaker (t2) and to test the Cellular Radio Display (t3). Table 93.1 contains the value of NOIi, NIi and FCi. Here the unit time U is considered to be 1 unit.

From the Table 93.1 we conclude that the prioritized test sequence is: **t3, t2, t1 or t3, t1, t2** The proposed model found to be very effective as it increases the Average Percentage of Fault Detection (APFD) when it is compared with generalized model based method and few code based methods like LOC count and Function count. The comparison made is summarized in Table 93.2.

The cost and time required for regression testing can be minimized by using the prioritization technique discussed in this paper. Here we have proposed a model based prioritization method by considering the number of Object Interactions per unit time as the objective function. Here more importance is given to number of inter component object interactions present because maximum faults are expected

Fig. 93.4 Sequence diagram for cellular connection

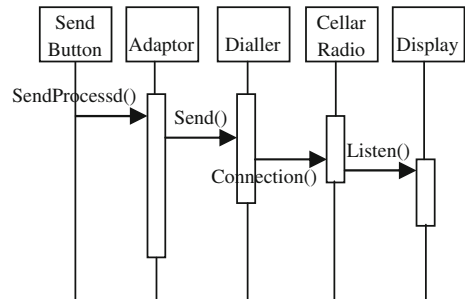


Fig. 93.5 OIG for a cellular network manager

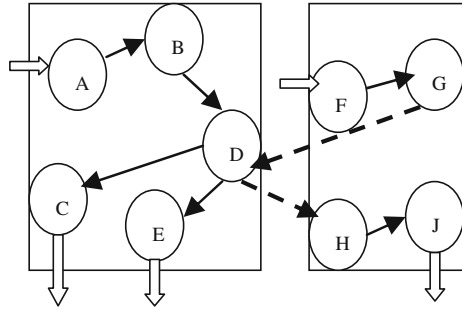


Table 93.1 Objective function (FC_i) evaluation

Test cases	NOI _i	NI _i	FC _i
t ₁	3	0	3
t ₂	3	0	3
t ₃	2	4	6

Table 93.2 a comparative study

Name of prioritized technique	Approximate increase in APFD value (%)
Code based approach (LOC count, function count etc.)	30
General model based approach	35
Model based approach using the dependency criteria in CBSD	45

to be present when components interact with each other. The proposed model found to be very effective as it increases the Average Percentage of Fault Detection (APFD) when it is applied to few of the projects developed in Java by java 45–50 %.

93.4 Conclusion

In this paper, the authors have taken the help of sequence diagrams of describing each component to construct an Object Interaction Graph (OIG) which shows the interrelation among the components. Furthermore, A new test prioritization algorithm is presented which is applied on OIG to count the maximum number of inter and intra component interactions. The experiments show that this approach is mainly applicable to test the component composition in case of component based software maintenance.

The proposed method can further be extended to prioritize test cases to perform regression testing for real time systems and distributed systems. The authors are also working on adding new criterion like frequency of data base access number of state changes in UML state chart diagram.

References

1. Elbaum, Z.S., Malishevsky, A., Rothermel, G.: Test case prioritization: a family of empirical studies. *IEEE Trans. Softw. Eng.* **28**(2), 159–182 (2012)
2. Kim, J.M., Porter, A.A.: A history-based test prioritization technique for regression testing in resource constrained environments. In: *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM Press, pp. 119–129 (2011)
3. Rothermel, G., Untch, R., Chu, C., Harrold, M.J.: Test case prioritization. *IEEE Trans. Softw. Eng.* **27**(10), 929–948 (2012)
4. Srikanth, H., Williams, L., Osborne, J.: System test case prioritization of new and regression test cases. In: *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE)*. IEEE Computer Society, pp. 62–71(2005)
5. Thiagarajan, S.J.: Effectively prioritizing tests in development environment. In: *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. ACM Press, pp. 97–106 (2009)
6. Kim, J., Porter, A.: A history-based test prioritization technique for regression testing in resource constraint environments. In: *Proceeding of the 24th International Conference on Software Engineering*, pp. 19–129 (2011)
7. Korel, B., Tahat, L., Harman, M.: Test prioritization using system models. *21st IEEE International Conference Software Maintenance (ICSM'05)*, pp. 559–568 (2005)
8. Korel, B., Koutsogiannakis, G., Tahat, L.: Application of system models in regression test suite prioritization. In: *Proceeding of the 24th International Conference Software Maintenance (ICSM'08)*, pp. 247–256 (2008)
9. Li, Z., Harman, M., Hierons, R.: Search algorithms for regression test case prioritization. *IEEE Trans. Softw. Eng.* **33**(4), 225–237 (2007)
10. UML 2.0 Reference Manual, Object Management Group (2003)
11. Thiagarajan, S.J.: Effectively prioritizing tests in development environment. In: *Proceeding ACM International Symposium on Software Testing and Analysis, ISSTA-02*, pp. 97–106 (2002)