A Fuzzy Test Cases Prioritization Technique for Regression Testing Programs with Assertions

Ali M. Alakeel
Faculty of Computing and Information Technology
University of Tabuk
Tabuk, Saudi Arabia
alakeel@ut.edu.sa

Abstract*—Program assertions have been recognized as a supporting tool during software development, testing, and maintenance. Therefore, software developers place assertions within their code in positions considered to be error prone or have the potential to lead to software crash or failure. Like any other software, programs with assertions have to be maintained. Depending on the type of modification applied to the modified program, assertions also may have to go through some modifications. New assertions may also be introduced in the new version of the program while some assertions may be kept the same. This paper presents a novel approach for test cases prioritization using fuzzy logic for the purpose of regression testing programs with assertions. The proposed approach builds upon previous research in the fields of assertions-based software testing and assertions revalidation. In a first step, our method utilizes fuzzy logic concepts to measure the effectiveness of a given test case in violating a program assertion. The result of the first step is then used in prioritization test cases during the regression testing of programs with assertions. The main objective of this research is to show that fuzzy logic concepts may be employed to measure the *effectiveness* of a given test case in violating programs assertions during the regression testing of a modified program.

Keywords--Regression Testing; Fuzzy Logic; Program Assertions; Software Testing; Software Test Data Generation.

I. Introduction

Program assertions have been recognized as a supporting tool during software development, testing, and maintenance, e.g., [1-5]. Therefore, software developers place assertions within their code in positions considered to be error prone or have the potential to lead to software crash or failure, e.g., [4]. An assertion specifies a constraint that applies to some state of computation. When an assertion evaluates to a *false* during program execution (this is called assertion violation), there exists an incorrect state in the program. Many programming languages support assertions by default, e.g., Java and Perl. For languages without built-in support, assertions can be added in the form of annotated statements. For example, Korel and Al-Yami [2], presents assertions as commented statements that are pre-processed and converted into Pascal code before compilation. Many types of assertions can be easily generated automatically such as boundary checks, division by zero, null pointers, variable overflow/underflow, etc. For this reason and to enhance their confidence in their software, programmers may be encouraged to write more programs with assertions.

Recognizing the importance of program assertions, some recent research efforts have been devoted for the development of algorithms and methods specifically designed for programs with assertions. For example, Korel et al. reported in [6] an algorithm for assertions revalidations during software maintenance. In [3], an algorithm is presented for the efficient processing and analysis of a large number of assertions present in the program. Also, a regression testing method for program with assertions was proposed in [7].

Like any other software, programs with assertions have to be maintained. Software maintenance usually involves activities during which the software is modified for different reasons. Some of the reasons for which software may be modified are fixing faults, introducing a new functionality, improving the performance of some parts of the software through the introduction of new algorithms, etc. A study in [8] shows that there is a probability of 50-80% of introducing faults to the modified software during software maintenance. For this reason regression testing is performed during software maintenance for the purpose of testing the modified software. There exists many regression testing methods which may be classified as specification-based or code-based. Specification-based regression testing strategies, e.g., [9-11] generate test cases based on the specification of the software, while code-base regression testing, e.g., [7], [12-15] strategies depends on the software structural elements to generate test cases.

Regression testing is a very labor intensive and may be responsible for approximately 50% of software maintenance's cost [16]. In a systematic software development environment, all types of regression testing methods usually involve the usage of an original test suite which is used for the purpose of testing the original program before it has been modified. Sensible regression testing methods have to utilize existing test suite in some form. For example, a simple regression testing strategy would rerun existing testing suite, as it is, on the modified program while introducing new test cases to test new features. Although this method is simple, it is not practical for commercial software because existing test suite is usually very large and may take weeks to rerun on the new modified software. Therefore, regression test selection techniques, test suite minimization technique and test case prioritization techniques are proposed in the literature.

To mitigate the cost associated with running the whole existing test suite, the main objective of regression test selection techniques, e.g., [17-18], and test suite minimization techniques, e.g., [19-20], is to select a representative subset of the original test suite using information about the original program, its modified version and the original test suite. It should be noted that both of the regression test selection and test suite minimization techniques eliminate some elements of the original test suite which may undermine the performance of these techniques. Test case prioritization techniques, e.g., [21-24] order elements of the original test suite based on a given criterion. Furthermore, test case prioritization techniques do not involve the selection of a subset of the original test suite. In this presentation, we will concentrate on test case prioritization techniques, therefore regression test selection and test suit minimization will not be discussed any further.

Depending on the type of modification applied to the modified program which includes assertions, assertions also may go through some modifications. New assertions may also be introduced in the new version of the program while some assertions may be kept the same as in the original program. This paper presents a novel approach for test cases prioritization using fuzzy logic for the purpose of regression testing programs with assertions. The main objective of this research is to show that fuzzy logic concepts may be employed to measure the *effectiveness* of a given test case in violating programs assertions during the regression testing of a modified program. The proposed method builds upon previous research in the fields of assertions-based software testing and assertions revalidation reported in [6-7]. In a first step, our method utilizes fuzzy logic concepts [25-27] to measure the effectiveness of a given test case in violating a program assertion. The result of the first step is then used in prioritization test cases during the regression testing of programs with assertions.

The rest of this paper is organized as follows. Related work is discussed in Section II. We present our proposed fuzzy test cases prioritization model in Section III. Our conclusions and future work is discussed in Section IV.

II. Related Work

Previous research in using fuzzy logic for the purpose of test case prioritization is scant. In [28], a fuzzy expert system is reported where this system is used for a telecommunication application. To build the required knowledge base for the expert system reported in this research, the researchers had to acquire knowledge from different sources such as customer profile, past test results, system failure rate, and the history of system architecture changes. Although this expert system has shown promising results with respect to the *specific* application it was designed for, it is necessary to acquire a new knowledge base for new applications. Also, the proposed method in [28] treats the software under test as a black box; therefore, it cannot be used for the purpose of regressing testing programs with assertions.

A. Regression Tesing for Programs with Assertions

This section briefly introduces the concept of regression testing for programs with assertions. For more detail, the reader is referred to [7]. Given an original program P_o and a modified version of this program P_m, let $A_o= \{a_{o1}, a_{o2}, a_{o3}, \dots a_{on}\}$ be a set of assertions found in P_o and $A_m= \{a_{m1}, a_{m2}, a_{m3}, \dots a_{mz}\}$ be a set of assertions found in P_m. Let $V \subseteq A_m$ be a set of assertions that are nominated for revalidation [6], using previous test suits, during the process of regression testing of P_m. Depending on the type of modification applied to the modified version, P_m, some assertions may have been kept the same; some assertions may have been modified, and new assertions may have been introduced. The main objective of regression testing for programs with assertions reported in [7] is to reduce the cost of regression testing of programs with assertions through the utilization of previous test suits that are used during the initial development process. Furthermore, this method concentrates on assertions that are kept the same and those which are modified; new assertions are not covered because new test cases must be generated to explore these assertions. The main elements of this method are described in the next paragraph.

Let $a_{mi} \in A_m$ be an assertion found in P_m. Assume that a_{mi} was not changed from its original form in P_o nor was it affected by the modifications [6] introduced to produce P_m. Therefore, a_{mi}, will be nominated, by the proposed approach, to belong to the set V, i.e., $a_{mi} \in V$. Suppose that assertions-oriented testing as reported in [2], has been performed on the original version P_o and a set of test cases were generated during this process and were kept for later usage during regression testing. Specifically, let $a_{ok} \in A_o$ be an assertion found in P_o and let $T(a_{ok})= \{t_{k1}, t_{k2}, t_{k3},\dots, t_{kr}\}$ be the set of test cases which were generated to explore this assertion during the application of assertion-oriented testing [2] on the original program P_o. In order to ensure that faults are not introduced during the production of the modified version P_m, regression testing has to be performed on P_m which has a set of assertions A_m. Given $a_{ok} \in A_o$, $T(a_{ok})= \{t_{k1}, t_{k2}, t_{k3},\dots, t_{kr}\}$, and $a_{mi} \in V$, it has been shown in [7] that the old test suit, $T(a_{ok})$, may be used to revalidate assertion a_{mi} during regression testing of the modified version P_m. Furthermore, it has been shown that using previous test suits to revalidate assertions may uncover faults in the modified version if these revalidated assertions were violated. Especially, faults for which assertions were originally designed to guard against in the original version of the program had these faults re-introduced in the modified version P_m [7].

Although the regression testing method for programs with assertions [7] has succeeded in saving the time to develop new test cases through the utilization of previous test suites that was used during the initial testing of the program, this method still consider using *all* test cases found in the previous test suit. Therefore, this method may not perform well in the present of a large previous test suit with thousands of test cases. In this paper we propose a test case prioritizing method which uses fuzzy logic concepts to select only a subset of the previous test cases. The proposed method is described in Sec. III.

B. Test Case Prioritization

The main goal of the prioritization techniques is to increase the probability of detecting faults at an earlier stage of testing [21-24]. Additionally, test case prioritization techniques objective is the utilization of previous test cases for the purpose of future testing. As stated in [21], there may exists several goals of test cases prioritization such as: (1) to increase test suites fault detection rate; (2) to minimize the time required to satisfy a testing coverage criterion; (3) to enhance tester's confidence in the reliability of the software in a shorter time period; (4) to be able to detect risky faults as early as possible; (5) to increase the chances of detecting faults related to software modification during regression testing.

In [21], an extensive study of nine different test case prioritization techniques was presented and compared according to their ability in fault detecting during regression testing. During that study a detection rate function is used to reorder test cases according to their ability to reveal program faults during regression testing. In [24], Extended Finite State Machine (EFSM) system model is proposed to be used instead of real programs to apply the same technique presented by [21] in order to reduce the cost of running test cases in real programs. Bryce et al. [22] presented a test prioritization model for Event-Driven software. This model concentrates on testing those parts related to the interface in GUI applications.

C. *Assertions Revalidation*

To deal with assertions in modified programs during regression testing, an assertions revalidation model was proposed in [6]. This approach is based on data dependency analysis and program slicing. In that research an algorithm is presented which is based on the computation of a static slice [29-30], for each assertion found in both the original and the modified program. These program slices are then compared to decide which assertions are to be revalidated. Although this method is very useful in identifying assertions that need to be revalidated, new test cases to revalidate assertions are generated from scratch for each assertion. For industrial size programs with a possibly large number of assertions, this approach may be very expensive.

D. *Fuzzy Logic Background*

In our daily life we use words and terms which are vague or fuzzy such as:

"The server is *slow*" or

"The weather is *hot*" or

"John is *tall*."

Fuzzy Logic concepts, e.g., [25-27], give us the ability to quantify and reason with words which have ambiguous meanings such the words (*slow*, *hot*, *tall*) mentioned above. In fuzzy sets [25], an object may belong partially to a set as opposed to classical or "crisp" sets in which an object may belong to a set or not. For example, in a universe of heights (in feet) for adult people defined as μ= {5, 5.5, 6, 6.5, 7, 7.5, 8}, a fuzzy subset TALL can be defined as follows:

TALL = [0/5, .125/5.5, .5/6, .875/6.5, 1/7, 1/7.5, 1/8].

In this example, the degree of membership for the members of the universe, μ, with respect to the set TALL may be interpreted as that the value "6" belongs to the set TALL 60% percent of the time while the value 8 belongs to the set TALL all the time.

III. A Fuzzy Test Cases Prioritization Technique

In this paper, our objective is to prioritize test cases according to their relative rate to violate a given program assertion. Note that it has been shown in [2] that violating an assertion implies revealing a programming fault. Our proposed fuzzy logic model for prioritization test cases during regression testing of programs with assertions is described as follows. Given an original program P_o and a modified version of this program P_m, let $A_o= \{a_{o1}, a_{o2}, a_{o3}, \dots a_{on}\}$ be a set of assertions found in P_o and $A_m= \{a_{m1}, a_{m2}, a_{m3}, \dots a_{mz}\}$ be a set of assertions found in P_m. Assume that we are performing regression testing for the modified version P_m using the regression testing method for programs with assertions as reported in [7]. Let $T_o=\{t_1, t_2, t_3,\dots, t_q\}$ be a previous test suite that was used during the process of assertion-oriented test data generation [2] of the original version P_o. For commercial software, testers usually deal with a very large number of test cases which make running all of them impractical. Therefore, given a set of test cases, our objective is to only reorder them according to some criterion that may convince us that some test cases may have better chances in violating a given assertion than the others. In this research, our criterion is the history of the test case during the process of testing the original program P_o.

Our problem is stated as follows. Given an assertion $a_{ok} \in A_o$ and $T(a_{ok})= \{t_{k1}, t_{k2}, t_{k3},\dots, t_{kr}\}$ as the test suite which were generated to explore assertion, a_{ok}, during the application of assertion-oriented testing [2] on the original program P_o. Our goal is to measure the effectiveness of a given test case, $t_{kj} \in T(a_{ok})$, in violating a given program assertion $a_{mr} \in A_m$, during the regression testing process of the modified version, P_m. To solve this problem we propose a fuzzy logic test cases prioritization technique shown in Fig. 1. The following paragraph describes how the proposed approach works.

Let $t_{kj} \in T(a_{ok})$, be a test case which was used to explore assertion $a_{ok} \in A_o$ during the initial testing of a program P_o. To measure the effectiveness of t_{kj} in violating the corresponding assertion $a_{mr} \in A_m$ in the modified version, P_m, during the process of regression testing the program P_m, we create a fuzzy set [25] called Effectiveness as follow. *Effectiveness* = {low, moderate, high}. Test cases related to any assertion $a_{ok} \in A_o$ where a_{ok} belongs to the set "Affected" will have *low* effectiveness in exploring the corresponding assertion in the modified version of the program. Similarly, test cases related to any assertion $a_{ok} \in A_o$ where a_{ok} belongs to the set "Partially Affected" will have *moderate* effectiveness in exploring the corresponding assertion in the modified version of the program. By the same token, test cases related to any assertion $a_{ok} \in A_o$ where a_{ok} belongs to the set "Not Affected" will have *high* effectiveness in exploring the corresponding assertion in the modified version of the program.

$$S(x;\, \alpha, \beta, \gamma) = \begin{cases} 0 & \mathit{for}\ x \le \alpha \\ 2\left(\dfrac{x-\alpha}{\gamma-\alpha}\right)^2 & \mathit{for}\ \alpha \le x \le \beta \\ 1-2\left(\dfrac{x-\gamma}{\gamma-\alpha}\right)^2 & \mathit{for}\ \beta \le x \le \gamma \\ 1 & \mathit{for}\ x \ge \gamma \end{cases}$$

Figure. 2. The S-function

In order to define the membership or grade values for each test case in the fuzzy set *Effectiveness*, we apply fuzzy logic techniques as follows. Each test case is assigned a membership depending on its "effectiveness," i.e., low, moderate or high. The membership value is in the interval [0,1] and reflects the compatibility of each specific test case to the fuzzy set *Effectiveness*. The assignment of membership values (grades) is based on the S-function [27] which is shown in Fig. 2. Note that other fuzzy clustering techniques other than the S-function may be used for the purpose of building up fuzzy sets and the assignment of membership functions. S-functions may be described as follows [27].

- A mathematical function that is used in fuzzy sets as a membership function.
- A simple but valuable tool in defining fuzzy functions such as the word "*tall*".
- The objects $\times$ are elements of some universe X. In this research, $\times$ represents the set of test cases we are dealing with during our prioritization mechanism, where these test cases are elements of the universe of the program possible input data.
- α, β, and γ are parameters which may be adjusted to fit the desired membership data. The parameter α represents the minimum boundary and γ represents the maximum boundary. The parameter β is the middle point between α and γ and is computed as $(\alpha + \gamma) / 2$.
- Depending on the application, a membership function may be controlled from different sources [27]. For example, in an expert system, the membership function will be constructed based on the experts' opinion modeled by the system.
- In this research, values of the parameters α and γ are determined after extermination with the proposed approach. As described previously, the history of each test case will be monitored during this experiment with regard to the ability of this specific test case in violating a given assertion in the program under test.

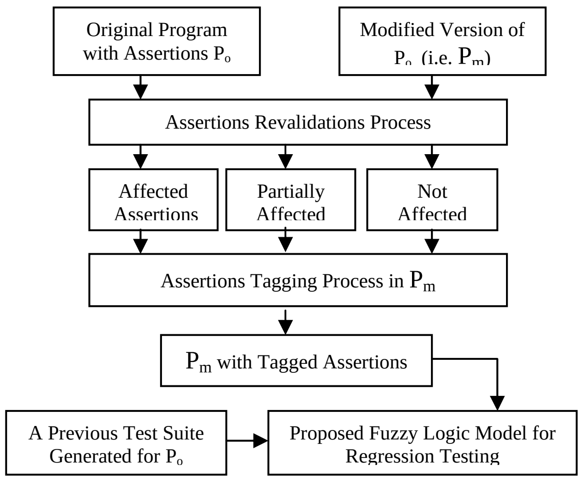
Original Program with Assertions P_o
Modified Version of P_o (i.e. P_m)
Assertions Revalidations Process
Affected Assertions
Partially Affected
Not Affected
Assertions Tagging Process in P_m
P_m with Tagged Assertions
A Previous Test Suite Generated for P_o
Proposed Fuzzy Logic Model for Regression Testing

Figure 1. Fuzzy Regression Testing Model for Programs with Assertions

The model shown in Fig. 1 may be described as follows. First, we analyze both P_o and P_m in order to classify assertions, A_m, found in P_m with respect to how much the modifications inflected on P_m had affected those assertions. To perform this analysis, we use assertions revalidations model [6] to classify the set of assertions, A_m, found in P_m into three different sets: "Affected," "Partially Affected" and "Not Affected." Based on the categorization of assertions in the analysis's step, the next step is to categorize test cases according to their expected effectiveness during regression testing of the modified, version of the program, i.e., P_m. Because the "effectiveness" of a test case is a "fuzzy" term which is very hard to measure in crisp value, we propose using fuzzy logic techniques to deal with measuring the effectiveness of a given test case as described previously.

IV. CONCLUSION and FUTURE WORK

In this paper, we presented a new technique for test cases prioritization to be used during regression testing of programs with assertions. The proposed model employs fuzzy logic concepts to measure the *effectiveness* of a given test case in violating programs assertions during the regression testing of a modified program. Our proposed method builds upon the concepts of previous research in the fields of assertions-based software testing and assertions revalidation. Furthermore, the proposed method is intended to be used in conjunction with traditional black-box and white-box software testing methods. In order to evaluate the proposed method, we intend to perform and extensive experimental study using a variety of programs with assertions. The results of this experiment will then be compared with existing test case prioritizations techniques reported in the literature.

REFERENCES

[1] Rosenblum, D., "Toward A Method of Programming WithAssertions," Proceedings of the International Conference on Software Engineering, pp. 92-104, 1992.

[2] Korel B. and Al-Yami A., "Assertion-Oriented Automated Test Data Generation," Proc. 18th Intern. Conference on Software Eng., Berlin, Germany, pp. 71-80, 1996.

[3] Alakeel A., "An Algorithm for Efficient Assertions-Based test Data Generation," Journal of Software, vol. 5, no. 6, pp. 644-653, 2010.

[4] Alakeel A. and Mahashi M., "Using Assertion-Based Testing in String Search Algorithms," Proceedings of The Third Int. Conf. on Advances in System Testing and Validation Lifecycle, Barcelona, Spain, pp. 1-5, 2011.

[5] Alakeel A., "A Framework for Concurrent Assertion-Based Automated Test Data Generation," European Journal of Scientific Research, vol. 46, no. 3, pp. 352-362, 2010.

[6] Korel B. , Zhang Q., and Tao L., "Assertion-Based Validation of Modified Programs," Proc. 2009 2nd Int'l Conference on Software Testing, Verification and Validation, Denver, USA, pp. 426-435, 2009.

[7] Alakeel A., "Regression Testing Method for Programs with Assertions," American Journal of Scientific Research, no. 11, pp. 111-122, 2010.

[8] Hetzel W. and Hetzel B., "The Complete Guide to Software Testing," John Wiley & Sons, Inc., New York, NY, 1991.

[9] Beydeda S. and Gruhn V.,"An Integrated Testing Technique for Component-Based Software," Proc. ACS/IEEE Int'l Conference on Computer Systems and Applications, pp. 328-334, 2001.

[10] Tsai W., Bai X., Paul R., and Yu L., "Scenario-Based Functional Regression Testing," Proc. IEEE Int’l Conference on Sofware and Applications, pp. 496-501, 2001.

[11] Korel B., Tahat L., and Vaysburg B., "Model Based Regression Test Reduction Using Dependence Analysis," Proc. IEEE Int'l Conferance on Software Maintenance, pp. 214-233, 2002.

[12] Chen Y., Rosenblum D., and Vo K., "Testtube: System for Selective Regression Testing," Proc. IEEE. Int'l Conference on Software Engineering, pp. 211-220, 1994.

[13] Gupta R., Harrold M., and Soffa M., "An Approach to Regression Testing Using Slices," Proc. IEEE Int'l Conference on Software Maintenance, pp. 299-308, 1992.

[14] Korel B. and Al-Yami A., "Automated Regression Test Generation," Proc. ACM Int'l Symposium on Software Testing and Analysis, pp. 143-152, 1998.

[15] Rothermel G. and Harrold M., "A Safe, Efficient Regression Test Selection Technique," ACM Tran. on Software Eng. and Methodology, vol. 6, no. 2, pp.63-68, 1996.

[16] Beizer B., “Software System Testing and Quality Assurance,” Thomson Computer Press, 1996.

[17] Rothermel G. and Harrold M., "Selecting Tests and Identifying Test Coverage Requirements for Modified Software," Proc. IEEE Int'l Conference on Software Maintenance, pp. 358-367, 1994.

[18] Masri W., Podgurski A., and Leon D., "An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows," IEEE Trans. Software Eng., vol. 33, no. 7, pp. 454-477, 2007.

[19] Logyall J., Mathisen S. , Hurley P., and Williamson J., "Automated Maintenance of Avionics Software," Proc. IEEE Aerospace and Electronics Conference, pp. 508-514,1993.

[20] Tsai W., Bai X., Paul R., and Yu L., "Scenario-Based Functional Regression Testing," Proc. IEEE Int’l Conference on Sofware and Applications, pp. 496-501, 2001.

[21] Rothermel G., Untch R., Chu C., and Harrold M., "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng., vol. 27, no. 10, pp. 929-948, 2001.

[22] Bryce C, Sampath S., and Memon A., “Develping a Single Model and Test Prioritization Strategies for Event-Driven Software,” IEEE Trans. Software Eng., vol. 37, no. 1, pp. 48-64, 2010.

[23] Korel B., Koutsogiannakis G., and Tahat L., "Application of System Models in Regression Test Suite Prioritization," Proc. IEEE Int'l Conference on Software Maintenance, pp. 247-256, 2008.

[24] Korel, B., Tahat L., and Harman M., "Test Prioritization Using System Models," Proc. IEEE Int'l Conference on Software Maintenance, pp. 559-568, 2005.

[25] Zadeh L., “Fuzzy Sets,” Information and Control, no. 8, pp. 338-353, 1965.

[26] Kosko B., “Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence,” Prentice-Hall, Englewood Cliffs, NJ, 1992.

[27] Giarratano J., “*Expert Systems: Principles and Programming*,” PWS-KENT Publishing Company, Boston, 1989.

[28] Xu Z., Gao K., and Khoshgoftaar T., “Application of Fuzzy Expert System in Test Case Selection for System Regression Test,” IEEE International Conference on Information Reuse and Integrationon , pp. 120-125, 2005.

[29] Horowitz S., Reps. T., and Binkley D., "Interprocedural Slicing using Dependence Graphs," ACM Trasn. Programming Languages and Systems, vol. 12, no. 1, pp. 26-60, 1990.

[30] Weiser M., 1984, "Program Slicing," IEEE Trans. Software Engineering, vol. 10, no. 4, pp. 352-357, 1984.