TEST CASE REDUCTION METHODS BY USING CBR

Siripong Roongruangsuwan and Jirapun Daengdej

Autonomous System Research Laboratory
Faculty of Science and Technology
Assumption University, Thailand
p4919742@au.edu, jirapun@scitech.au.edu

ABSTRACT

It has been proven that software testing usually consumes over 50% of the costs associated with the development of commercial software systems. Particularly, regression testing activities has been shown to be a critically important phase of software testing. Many reduction techniques have been proposed to reduce costs. Unfortunately, the cost is usually over budget and those methods are failed to reasonably control costs. The primarily outstanding issue is non-effective methods to remove redundancy tests while a bigger size of tests and a significant amount of time are still remaining. To resolve the issue, this paper proposes an artificial intelligent concept of case-based reasoning (CBR). CBR has an uncontrollable costs issue as same as testing. There are many effective algorithms researched over a long period of time. This study introduces three methods combined between CBR's deletion algorithm and testing activities. Those methods aim to minimize size of tests and time, while preserving fault detection.

Index Terms - test case reduction, test reduction, test reduction CBR, CBR for testing and test reduction techniques

1. INTRODUCTION

Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test [7], with respect to the context in which it is intended to operate. Software Testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of implementation of the software. Test techniques include the process of executing a program or application with the intent of finding software bugs. It can also be stated as the process of validating and verifying that software meets the business and technical requirements that guided its design and development, so that it works as expected. Software Testing can be implemented at any time in the development process; however, the most test effort is employed after the requirements have been defined and coding process has been completed.

Many researchers [10], [11], [12], [13], [14], [15], [23], [24], [25], [26], [29], [36], [37], [39] have proven that these test case reduction methods can reserve the fault detection capability. There are many outstanding research issues in this area. In this paper, the research issues are: redundancy test cases are still remained, an uncontrollable growth of test cases and existing reduction methods consume a great deal of time and cost during a reduction process. The literature review [16] shows that there are many techniques to resolve those three issues. One of effective approaches is to apply the concept of artificial intelligent. There are many artificial intelligent concepts, such as neutral network, fuzzy logic, learning algorithms and case-based reasoning (CBR). CBR is one of the most popular and actively researched areas in the past. The researches [4], [8], [16], [26] show that CBR has identical problems as same as software testing topic. In software testing field, particularly during regression testing activities, the key research issues are: (a) too many redundancy test cases after reduction process (b) a decrease of test cases' ability to reveal faults and (c) uncontrollable grow of test cases. Meanwhile, the key research issues in CBR field are: (a) there are too many redundancy cases in the CBR system (b) a size of CBR system is continuously growing all the time and (c) existing CBR deletion algorithms take longer time to remove all redundancy cases in the CBR system. Those issues in CBR field can be elaborated as follows: Fundamentally, there are four steps in the CBR system, which are: retrieve, reuse, revise and retain. These steps can lead to a serious problem of uncontrollably growing cases in the system. However, the study shows that there are many proposed techniques in order to control a number of cases in the CBR system, such as add algorithms, deletion algorithms and maintenance approaches. CBR have been investigated by CBR researchers in order to ensure that only small amounts of efficient cases are stored in the case base. The previous work [28] shows that deletion algorithms are the most popular and effective approaches to maintain a size of the CBR system. There are many researchers have proposed several deletion algorithms [4], [8], [31], such as random method, utility approach and footprint algorithm. These algorithms aim to: (a) remove all redundancy or unnecessary cases (b) minimize size of system and reduction time and (c) preserve the ability of solving problems. Nevertheless, each technique has strength and weakness. Some methods are suitable for

removing cases. Some methods are perfectly suitable for reducing time. Some may be used for reserving the problem solving capability. Eventually, the previous work [28] discovered several effective methods (e.g. confidential case filtering method, coverage value algorithm and confidential coverage approach) to remove those cases, minimize size of CBR and reduce amount of time, while preserving the ability of CBR system's problem solving skill. Therefore, this paper applies those effective deletion techniques to resolve the problems of software testing. In the light of software testing, the proposed techniques focus on how to maintain the test case or test data while the ability to reveal faults is still preserved. It is assumed that *test cases* or *test data* in this paper are treated as *cases* in the CBR system. Also, there is an assumption that a given set of test cases are generated by a path-oriented test case generation technique. The path-oriented technique is widely used for a white-box testing, which this paper does not address how to generate test cases with path-oriented methods.

Section 2 discusses an overview of test case reduction techniques and processes. Also, section 2 discusses a concept of CBR. Section 3 provides a definition of terminologies used in this paper. Section 4 lists the outstanding research issues motivated this study. Section 5 proposes three new test case reduction methods. Section 6 describes an evaluation method and discusses a result. The last section represents all source references used in this paper.

2. LITERATURE REVIEW

This section describes an overview of test case reduction techniques and the concept of CBR. The following describes those two areas in details.

2.1. Test Case Reduction Techniques

This section discusses and organizes test case reduction (or TCR) techniques researched in 1995-2006. This study shows that there are many researchers who proposed a method to reduce unnecessary test cases (also known as redundancy test cases), like Offutt [5], Rothermel [12], McMaster [24] and Sampth [27]. These techniques aim to remove and minimize a size of test cases while maintaining the ability to detect faults. The literature review [1], [10], [11], [12], [13], [14], [15], [24], [25], [36], [37], [39] shows that there are two types of reduction techniques, which are: (a) pre-process and (b) post-process. First, the pre-process is a process that immediately reduces a size of test cases after generating. Typically, it is occurred before regression testing phase. Second, the post-process is a process that maintains and removes unnecessary test cases, after running the first regression testing activities. Although these techniques can reduce the size of test cases, but the ability to reveal faults seems slightly to be dropped. However, Jefferson Offutt [5] and Rothermel [10], [11], [12], [13], [14], [15], [30], [31], [33] has proven that these test case reduction techniques have many benefits, particularly during the regression testing phase, and most of reduction techniques can maintain an acceptable rate of fault detection. The advantages of these techniques are: (a) to spend less time in executing test cases, particularly during the regression testing phase (b) to significantly reduce time and cost of manually comparing test results and (c) to effectively manage the test data associated with test cases. This study proposes a new "2C" classification of test case reduction techniques, classified based on their characteristics, as follows: (a) coverage-based techniques and (b) concept analysis-based techniques.

2.2. Case-Based Reasoning (CBR)

Over the time, CBR is growing. When the uncontrollable case-based growth is occurred, the performance of CBR is decreasing. Therefore, the maintenance process is required in order to preserve or improve the performance of the system. The process of maintaining CBR is called CBM. David C. Wilson [8] presented the overall concepts of CBR and case based maintenance. This paper focused on the case based maintenance (CBM) approach in term of the framework. In other words, this paper described the type of data collection and how the case based maintenance works. There were so many policies for CBM, for example, addition, deletion, and retain.

"CBM was defined as the process of refining a CBR system's case-base to improve the system's performance. It implements policies for revising the organization or contents (representation, domain content, accounting information, or implementation) of the case-base in order to facilitate future reasoning for a particular set of performance objectives."

These studies [4], [5], [6], [8], [19], [20], [28] reveal that several deletion algorithms have been proposed. For example, a random approach (RD), utility deletion algorithm (UD), footprint deletion algorithm (FD), footprint utility deletion algorithm (FUD) and iterative case filtering algorithm (ICF).

RD is the simplest approach, which removes the case randomly. UD deletes the case that has minimum utility value. Footprint algorithm uses the competence model and removes the auxiliary case from the system. FUD is a hybrid approach between Utility algorithm and Footprint algorithm, and is concerned with the competence model and the utility value. Finally, ICF focuses on the case, which the reachability set is greater than the coverage set [19], [28].

3. DEFINITION

This section describes a definition of terminologies.

Definition 1: Barry [4] defined the *CBR, case base, auxiliary case and pivotal case* as follows:

"Case-Based Reasoning is one of the Artificial Intelligence-based algorithms, which solve the problems by searching through the case storage for

the most similar cases. CBR has to store their solved cases back to their memory or storage in order to learn from their experience."

"Case Base is a collection of cases in CBR, which can be defined as the following: Given a case - base C = {c1… cn}, for c ε C whereas C = CBR, c = case"

Definition 2: *"Auxiliary Case is a case that does not have a direct effect on the competence of a system when it is deleted. The definition of auxiliary case can be described as follows:*

Auxiliary cases do not affect competence at all. Their deletion only reduces the efficiency of the system. A case is an auxiliary case if the coverage it provides is subsumed by the coverage of one of its reachable cases."

Definition 3: *"Pivotal Case is the case that does have a direct effect on the competence of a system if it is deleted.*

A case is a pivotal case if its deletion directly reduces the competence of a system (irrespective of the other cases in the case-base) [2], [3]. Using the above estimates of coverage and reachability a case is pivotal if it is reachable by no other case but itself."

4. RESEARCH CHALLENGES

This section discusses the details of research issues motivated this study. The literature review reveals that [7], [22], [24], [25], [27], [38] those research issues are: (a) too many redundancy test cases after reduction process (b) a decrease of test cases' ability to reveal faults and (c) uncontrollable grow of test cases. These research issues can be elaborated in details as follows: First, the literature review shows that redundancy test cases are test cases tested by multiple test cases. Many test cases that are designed to test the same things (e.g. same functions, same line of code or same requirements) are duplicated. Those duplicated tests are typically occurred during testing activities, particularly during regression testing activities [7], [22], [24], [25], [27], [38]. Those duplicated tests can be eventually removed in order to minimize time and cost to execute tests. The study shows that there are many proposed methods to delete those duplicated test cases such as McMaster's work [24] [25], Jeff's method [7] and Khan's approach [22]. Also, the study shows that one of the most interesting research issues is to minimize those duplicated tests and reduce cost of executing tests. Although there are many proposed methods to resolve that issue, that issue is still remaining. Thus, it is a challenge for researchers to continuously improve the ability to remove duplicated tests. Second, test cases are designed to reveal faults during software testing phase. The empirical studies [10], [11], [12], [23], [30], [31], [33], [39] describe that reducing test cases may impact to the ability of detect faults. Many reduction methods decrease a capability of testing and reveal those faults. Therefore, one of outstanding research challenges for researchers is to remove tests while preserving the ability to defect faults. Last, this paper shows that uncontrollable grow of test cases can be typically occurred during software testing process and evolution. Even if there are many reduction methods proposed to control and limit growth of tests, unfortunately it appears that a number of test cases is still large. Obviously, the greater size of test cases takes longer time and cost to execute.

5. PROPOSED METHODS

For evolving software, test cases are growing dramatically. The more test cases software test engineers have, the more time and cost software test engineers consume. The literature review shows that regression testing activities consume a significant amount of time and cost. Although, a comprehensive set of regression selection techniques [10], [11], [12], [13] has been proposed to minimize time and cost, there is an available room to minimize size of tests and clean up all unnecessary test cases. Thus, removing all redundancy test cases is desirable. There are many approaches to reduce redundancy test cases and applying an artificial intelligent concept in the test case reduction process is an innovated approach. The literature review [16], [28] shows that there are many areas of artificial intelligent concept, such as artificial neutral network, fuzzy logic, learning algorithms and CBR concept. Also, it reveals that CBR has a same research issue as software testing has. The issue is that cases in the CBR system will be consistency growing bigger and larger all the time. There are four steps in CBR that can uncontrollably grow a size of the system: retrieve, reuse, revise and retain. Therefore, many CBR papers aim to reduce all redundancy cases, known as "deletion algorithms". The smaller size of CBR system is better and desirable. Due to the fact that CBR has the same problem as software testing and this paper focuses on reduction methods, therefore, this paper proposes to apply CBR deletion algorithms to the test case reduction techniques. This paper introduces three reduction methods that apply CBR deletion algorithms: TCCF, TCIF and PCF methods. Those techniques aim to reduce a number of test cases generated by path-oriented test case generation technique. This technique is used for white-box testing only. However, the generation methods are out of the scope of this paper.

5.1. Example of Test Cases

Given a set of test cases generated, this study discusses the use of a number of case maintenance techniques, which have been investigated by CBR researchers in ensuring that only small amount of cases are stored in the case base, thereby reducing number of test cases should be used in software testing. Similar to what happen to software testing, a number of CBR researchers have focused on finding approaches especially for reducing cases in the CBR

systems' storages. This paper proposes to use the path coverage criteria in order to reduce redundancy test cases. This is because path coverage has a huge benefit of required very thorough testing activities. The following describes in details of the above path coverage using in the software testing field. Let S = {s1, s2, s3, s4, s5} to be a set of stage in the control flow graph. The control flow graph can be derived from the source-code or program. It is a white-box testing. Thus, each state represents a block of code. The techniques that aim to generate and derive test cases from the control flow graph are well-known as path-oriented test case generation techniques. These techniques are widely used to generate test cases. There are many research papers on this area. However, the test case generation techniques are out of scope in this paper.

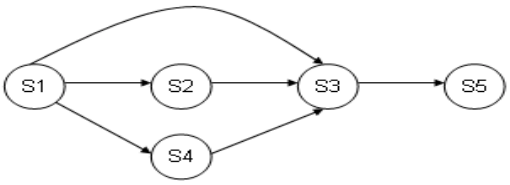

Figure 1 An Example of Control Flow Graph

From the above figure, this paper assumes that each state can reveal a fault. Thus, an ability to reveal faults of five states is equal to 5. Also, it is assumed that every single transaction must be tested. This example is used in the rest of paper.

Let $TC_n = \{s_1, s_2, \ldots, s_n\}$ where TC is a test case and s_n is a stage or node in the path-oriented graph that is used to be tested. From the above figure, a set of test cases can be derived as follows:

$TC_1 = \{s_1, s_2\}$
$TC_2 = \{s_1, s_3\}$
$TC_3 = \{s_1, s_4\}$
$TC_4 = \{s_1, s_2, s_3\}$
$TC_5 = \{s_1, s_3, s_5\}$
$TC_6 = \{s_1, s_4, s_3\}$
$TC_7 = \{s_1, s_2, s_3, s_5\}$
$TC_8 = \{s_1, s_4, s_3, s_5\}$
$TC_9 = \{s_2, s_3\}$
$TC_{10} = \{s_2, s_3, s_5\}$
$TC_{11} = \{s_3, s_5\}$
$TC_{12} = \{s_4, s_3\}$
$TC_{13} = \{s_4, s_3, s_5\}$

The following describes the proposed methods that apply the concept of CBR in details:

5.2. Test Case Complexity for Filtering (TCCF)

A complexity of test case is the significant criteria in this proposed method [2], [19]. In this paper, the complexity of test case measures a number of states included in each test case.

Let Cplx(TC) = {High, Medium, Low} where Cplx is a complexity of test case, TC is a test case and the complexity value can be measured as:

- *High* when a number of states are greater than an average number of states in the test suite.
- *Medium* when a number of states are equal to an average number of states in test suites.
- *Low* when a number of states are less than an average number of states in the test suites.

The procedures of this method can be described briefly in the following steps.

The first step is to determine a coverage set. From figure 1, each coverage set can be identified as follows:

Coverage (1) = {TC_1}
Coverage (2) = {TC_2}
Coverage (3) = {TC_3}
Coverage (4) = {TC_1, TC_4, TC_9}
Coverage (5) = {TC_2, TC_5, TC_{11}}
Coverage (6) = {TC_3, TC_6, TC_{12}}
Coverage (7) = {TC_1, TC_4, TC_7, TC_9, TC_{10}, TC_{11}}
Coverage (8) = {TC_3, TC_6, TC_8, TC_{11}, TC_{12}, TC_{13}}
Coverage (9) = {TC_9}
Coverage (10) = {TC_9, TC_{10}, TC_{11}}
Coverage (11) = {TC_{11}}
Coverage (12) = {TC_{12}}
Coverage (13) = {TC_{11}, TC_{12}, TC_{13}}

The second step is also to determine a reachability set. The reachability set can be figured out from the above coverage set, based on the given definition in this paper. Therefore, the reachability set can be identified as follows:

Reachability (TC_1) = {1, 4, 7}
Reachability (TC_2) = {2, 5}
Reachability (TC_3) = {3, 6, 8}
Reachability (TC_4) = {4, 7}
Reachability (TC_5) = {5}
Reachability (TC_6) = {6, 8}
Reachability (TC_7) = {7}
Reachability (TC_8) = {8}
Reachability (TC_9) = {4, 7, 9, 10}
Reachability (TC_{10}) = {7, 10}
Reachability (TC_{11}) = {5, 7, 8, 10, 11, 13}
Reachability (TC_{12}) = {6, 8, 12, 13}
Reachability (TC_{13}) = {8, 13}

Next, the step is to define an auxiliary set. The given definition of auxiliary set is to find a test case that does not have a direct effect on the ability to reveal faults when it is removed. From figure 1, therefore, the auxiliary set can be identified as follows:

Auxiliary set = $\{TC_1, TC_2, TC_3, TC_4, TC_5, TC_6, TC_9, TC_{10}, TC_{11}, TC_{12}, TC_{13}\}$

Afterward, the method computes a complexity value for all test cases in the above auxiliary set. From figure 1 and test suites that contain 13 test cases, the average number of states is equal to 3. Therefore, the complexity value for each test case can be computed as follows:

$Cplx(TC_1) = Low$, $Cplx(TC_2) = Low$, $Cplx(TC_3) = Low$, $Cplx(TC_4) = Medium$, $Cplx(TC_5) = Medium$, $Cplx(TC_6) = Medium$, $Cplx(TC_9) = Low$, $Cplx(TC_{10}) = Medium$, $Cplx(TC_{11}) = Low$, $Cplx(TC_{12}) = Low$ and $Cplx(TC_{13}) = Medium$

Finally, the last step removes test cases with minimum of complexity value from the auxiliary set. Thus, TC_1, TC_2, TC_3, TC_9, TC_{11} and TC_12 are removed.

5.3. Test Case Impact for Filtering (TCIF)

The study [21] shows that software is error-ridden in part because of its growing complexity. Software is growing more complex every day. The size of

software products is no longer measured in thousands of lines of code, but it measures in millions. Software developers already spend approximately 80 percent of development costs [21] on identifying and correcting defects, and yet few products of any type other than software are shipped with such high levels of errors. Other factors contributing to quality problems include marketing strategies, limited liability by software vendors, and decreasing returns on testing and debugging, according to the study. At the core of these issues is difficulty in defining and measuring software quality. Due to the fact that defining and measuring a quality of software is important and difficult, the impact of inadequate testing must not be ignorance. The impact of inadequate testing could be lead to the problem of poor quality, expensive costs and huge time-to-market. In conclusion, software testing engineers require identifying the impact of each test case in order to acknowledge and understand clearly the impact of ignoring some test cases. In this paper, an impact value is an impact of test cases in term of the ability to detect faults if those test cases are removed and not be tested.

Let Imp(TC) = {High, Medium, Low} where Imp is an impact if a test case is removed, TC is a test case and the impact value can be measured as:

- *High* when the test case has revealed at least one fault for many times.
- *Medium* when the test case has revealed faults for only one time.
- *Low* when the test case has never revealed faults.

The procedure of this method is similar to the previous method. The only different is that this method aims to use an impact value instead of complexity value. Therefore, the fire three steps are to: identify coverage set, define reachability set and determine an auxiliary set. Afterward, the next step is to compute and assign an impact value. The method computes the impact value for all test cases in the above auxiliary set. From figure 1, the impact value for each test case can be computed as follows:

$$Imp(TC_1) = Low, Imp(TC_2) = High, Imp(TC_3) = Medium, Imp(TC_4) = Low, Imp(TC_5) = High, Imp(TC_6) = Medium, Imp(TC_9) = Low, Imp(TC_{10}) = Low, Imp(TC_{11}) = Low, Imp(TC_{12}) = Low \text{ and } Imp(TC_{13}) = Low$$

Finally, the last step removes test cases with minimum of impact value from the auxiliary set. Thus, TC_1, TC_4, TC_7, TC_9, TC_{10}, TC_{11}, TC_{12} and TC_{13} are removed.

5.4. Path Coverage for Filtering (PCF) Method

Code coverage analysis is a structural testing technique (also known as white box testing). Structural testing compares test program behaviour against the apparent intention of the source code. This contrasts with functional testing (also referred to black-box testing), which compares test program behaviour against a requirements specification. Structural testing examines how the program works, taking into account possible pitfalls in the structure and logic. Functional testing examines what the program accomplishes, without regard to how it works internally. Structural testing is also called path testing since you choose test cases that cause paths to be taken through the structure of the program. The advantage of path cover is that it takes responsible for all statements as well as branches across a method. It requires very thorough testing. This is an effective substitute of other coverage criteria. The path coverage is used as coverage value in this technique. The Coverage value is combined into the addition policy for adding significant case [17]. Within the adding algorithm along with the coverage weight value stated in the review, the concept of deletion algorithm and the coverage have been proposed. The coverage value can specify how many nodes that the test case can cover. In other words, the coverage value is an indicator to measure that each test case covers nodes. It means that the higher coverage value is, the more nodes can be contained and covered in the test case. Let *Cov*(n) = value where *Cov* is a coverage value, value is a number of test cases in each coverage group and n is a coverage relationship.

The procedure of this method can be elaborated briefly as the following steps. From figure 1, the first step is to identify a coverage set, which has been already identified in the previous method. The next step is to calculate a coverage value. This paper proposes to calculate a coverage value based on a number of test cases in each coverage group. Therefore, the coverage value can be computed as follows:

Cov(1) = 1, *Cov*(2) = 1, *Cov*(3) = 3, *Cov*(4) = 3, *Cov*(5) = 3, *Cov*(6) = 4, *Cov*(7) = 6, *Cov*(8) = 6, *Cov*(9) = 1, *Cov*(10) = 3, *Cov*(11) = 1, *Cov*(12) = 1 and *Cov*(13) = 3.

The last step removes all test cases with minimum coverage value, in the potential removal set. Therefore, TC_1, TC_2, TC_9, TC_{11} and TC_{12} are removed.

6. EVALUATION

This section describes an experiments design, measurement metrics and results. This paragraph designs an experiment used to evaluate and determine the best reduction methods. This paper proposes the following three steps. First, the experiment proposes to randomly generate 2,000 test data used in the telecommunication industry. In this experiment, the test data is represented as test case. Second, the experiment executes reduction methods with the generated test cases and compares among the following reduction methods: RD, UD, FD, FUD, ICF and three proposed methods (e.g. TCCF, TCIF and PCF). This step randomly simulates defects for each test case in order to determine an ability to reveal faults. Third, the experiment aims to run the above methods for 10 times in order to calculate the average value for each metric. The metrics used in this

experiment are described in details in next section. Afterward, the experiment compares the values and evaluates a result by generating a comparison graph in order to determine the most recommended reduction approach.

The following table lists the description of each test data that need to be generated randomly.

Table 1 An Example Form of Test Cases

Attribute	**Description**	**Data Type**
Test Id	A unique index to reference test data. The value is a sequence number, starting at 1.	Numeric
Full Name	A first and last name who own the mobile phone.	String
Name	A mobile brand name. The value is a range of iPhone, BlackBerry, Nokia, LG, Sony Ericsson and Samsung.	String
Coverage Value	A value of Coverage set, which is defined by the user.	Numeric
Impact Value	An impact value of each case, in this work. This can be matched to the impact value.	Numeric
wCoverageValue	The weight value for coverage set	Numeric
A set of states	A set of states that required to be tested. State is directly derived from control flow graph. The control flow graph is a result of path-oriented test case generation techniques.	Array
Complexity	An indicator to represent a complexity of test case. The complexity of test cases represents how difficult to execute each test case.	Numeric
Impact	An indicator to represent an impact value in case that test case is ignored.	Numeric
Coverage	An indicator to represent how many states each test case cover.	Numeric
Status	An indicator to represent that test case can reveal faults or not. The status can be only either pass or fail. If the status is fail, it mean that fault is detected.	Boolean

The following table describes an approach to generate random data using the above attributes respectively.

Table 2 Approach to Generate Random Test Case

Attribute	**Approach**
Test Id	Generate randomly from the following combination: t + *Sequence Number*. For example, t1, t2, t3, …, tn.
Name	Random from the following values: iPhone, BlackBerry, Nokia, LG, Sony and Samsung.
ImpValue	Set as a zero (0) at the beginning
wCoverageValue	Set as a one (1) at the beginning
A set of states	There are two elements needed to be randomly generated: (a) a number of states that needed to be tested by each test case and be generated between 1 and 100. (b) states themselves that described as follows: Generate randomly from the following combination: s + *Sequence Number*. For example, s1, s2, s3, …, sn.
Cplx	Random from the following values: 1-100
Impact	Random from the following values: 1-100
Coverage	Compute a number of states from “a set of states” field

The paragraph lists the measurement metrics used in the experiment. The first measurement is a number of test cases. The large number of test cases consumes time, effort and cost more than the smaller size of test cases. Many reduction or minimization approaches [1], [10], [11], [12], [13], [14], [15], [24], [25], [36], [37], [39] have been proposed to minimize size of test cases. This has proven that size is one of important metrics in software testing area. The second is an ability to reveal faults. It aims to measure the percentage of faults detection. One of the goals of test case with a set of data is to find defects. Thus, this metric is important criteria to measure and determine which reduction methods can preserve the high ability to reveal faults. The last measurement is a total of reduction time: It is the total number of times running the reduction methods in the experiment. This metric is related to time used during execution time and maintenance time of test case reduction methods. Therefore, less time is desirable. This paragraph discusses an evaluation result of the above experiment. This section presents the reduction methods results in term of: (a) a number of test cases (b) ability to reveal faults and (c) total reduction time. The comparative methods are: RD, UD, FD, FUD, ICF, TCCF, TCIF and PCF. Additionally, this section shows a graph format. There are two dimensions in the following graph: (a) horizontal and (b) vertical axis. The horizontal represents three measurements whereas the vertical axis represents the percentage value.

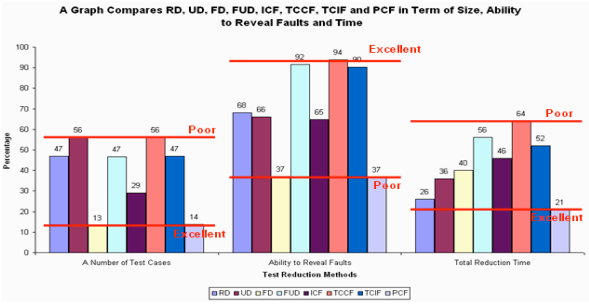

Figure 2 A Graph Comparison of Deletion Methods

The above graph presents that both of FD and PCF minimize a number of test cases by far better than other reductions methods, approximately over 15%. Meanwhile, both of them are the worst methods for preserving an ability to reveal faults. FUD, TCCF and TCIF are best top three methods to reserve a capability to detect faults. They are greater than other methods over 22%. Unfortunately, they are also the worst three methods that require a lot of time during a reduction process. In the mean time, both of RD and PCF take the least total reduction time among other methods. The evaluation result suggests that FD and PCF is perfectly suitable for a scenario that does not directly concern about an ability to reveal faults and total reduction time. Both of FD and PCF are two of the most excellent methods to minimize a number of test cases. Meanwhile, FUD, TCCF and TCIF are the most recommended methods to delete tests while preserving the ability to detect faults. In addition, both of RD and PCF are excellent in case that total time is matter.

7. CONCLUSION

This paper reveals that there are many research challenges and gaps in the test case reduction area. Those challenges and gaps can give the research direction in this field. However, the research issues that motivated this study are: (a) too many redundancy test cases after reduction process (b) a decrease of test cases' ability to reveal faults and (c) uncontrollable grow of test cases. This paper combines the concept of software testing and CBR. Those two concepts could be used together on practical software development scenarios. The proposed maintenance algorithms are significant approaches for removing unnecessary test cases and are used for controlling the growth of test cases. Those approaches are aimed at maintaining the large test cases by minimizing the time consumed by execution & maintenance and reducing the size of the test cases along with preserving the ability to reveal faults as much as possible. Also, the evaluation reveals that they have been achieved by removing a number of test cases, minimizing time for executing & maintenance and preserving the fault-detection ability with sample of 2,000 test cases. However, the primarily limitation of those approaches is about the path coverage. The path coverage may be not an effective coverage factor for a huge system that contains million lines of code. This is because it requires an exhaustive time and cost of identify coverage from a huge amount of codes. Thus, one of the future works is to apply other coverage factors for those approaches. Finally, this paper recommends researchers to improve the ability to reduce duplicated or unnecessary test cases from multiple test suites, enhance the capability to reduce test cases in the large commercial system and develop a systematic approach to identify an impact and complexity of tests.

8. REFERENCES

[1] A. Jefferson Offutt, Jie Pan and Jeffery M. Voas, "Procedures for Reducing the Size of Coverage-based Test Sets", 1995.

[2] Barry Smyth & Keane. "Remembering To Forget: A Competence Preserving Deletion Policy for Case-Based Reasoning Systems" In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 377-382. Morgan-Kaufman, 1995.

[3] Barry Smyth Ph.D. Thesis. "Case Based Design" Department of Computer Science, Trinity College, Dublin Ireland, 1996.

[4] Barry W. Boehm, "A Spiral Model of Software Development and Enhancement", TRW Defense Systems Group, 1998.

[5] Boris Beizer, "Software Testing Techniques, Van Nostrand Reinhold", Inc, New York NY, 2nd edition. ISBN 0-442-20672-0, 1990.

[6] Bo Qu, Changhai Nie, Baowen Xu and Xiaofang Zhang, "Test Case Prioritization for Black Box Testing", 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), 2007.

[7] Cem Kaner, "Exploratory Testing", Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, 2006.

[8] David C. Wilson. Ph.D. Thesis "A Case-Based Maintenance: The husbandry of experiences." Department of Computer Science, Indiana University, 2001.

[9] E. Lehmann and J. Wegener, "Test case design by means of the CTE XL", In Proc. of the 8th European International Conf. on Software Testing, Analysis & Review (EuroSTAR 2000), 2000.

[10] Gregg Rothermel, Roland H. Untch, Chengyun Chu and Mary Jean Harrold, "Prioritizing Test Cases for Regression Testing", IEEE Transactions on Software Engineering, 2001.

[11] Gregg Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study", In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 179-188, Oxford, England, UK, 1999.

[12] Gregg Rothermel, Mary Jean Harrold, Jeffery Ostrin and Christie Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", In *Proceedings of IEEE International Test Conference on Software Maintenance (ITCSM'98)*, Washington D.C., pp. 34-43, 1998.

[13] Gregg Rothermel, Mary Jean Harrold, Jeffery von Ronne and Christie Hong, "Empirical Studies of Test-Suite

Reduction", In Journal of Software Testing, Verification, and Reliability, Vol. 12, No. 4, 2002.
[14] Gregg Rothermel and Mary Jean Harrold, "A Safe, Efficient Regression Test Selection Technique", ACM Transactions on Softw. Eng. And Methodology, 6(2): 173-210, 1997.
[15] Gregg Rothermel and Mary Jean Harrold, "Analyzing Regression Test Selection Techniques", IEEE Transactions on Software Engineering, 22(8):529-551, 1996.
[16] Jirapun Daengdej, Ph.D. Thesis, "Adaptable Case Base Reasoning Techniques for Dealing with Highly Noise Cases" The University of New England, Australia, 1998.
[17] Jun Zhu and Quiang Yang. "Remembering To Add Competence-preserving Case Addition Policies for Case Base Maintenance." In *Proceedings of the 16th International Joint Conference in Artificial Intelligence*, 234-241. Morgan-Kaufmann, 1999
[18] Mary Jean Harrold, Rajiv Gupta and Mary Lou Soffa, "A Methodology for Controlling the Size of A Test Suite", ACM Transactions on Software Engineering and Methodology, 2(3):270-285, 1993.
[19] Nicha Kosindrdecha and Jirapun Daengdej, "A Deletion Algorithm for Case-Based Maintenance Based on Accuracy and Competence", Assumption University, Thailand, 2003
[20] Nicha Kosindrdecha and Siripong Roongruangsuwan, "Reducing Test Case Created by Path Oriented Test Case Generation", AIAA 2007 Conference and Exhibition, Rohnert Park, California, USA, 2007.
[21] NIST, "The economic impacts of inadequate infrastructure for software testing", 2002.
[22] Saif-ur-Rebman Khan and Aamer Nadeem, "TestFilter: A Statement-Coverage Based Test Case Reduction Technique", 2006.
[23] Sara Sprenkle, Sreedevi Sampath and Amie Souter, "An Empirical Comparison of Test Suite Reduction Techniques for User-session-based Testing of Web Applications", Journal of Software. Testing, Verification, and Reliability, 4(2), 2002.
[24] Scott McMaster and Atif Memon, "Call Stack Coverage for Test Suite Reduction", *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 539-548, Budapest, Hungary, 2005.
[25] Scott McMaster and Atif Memon, "Call Stack Coverage for GUI Test-Suite Reduction", Proceedings of the 17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006), NC, USA, 2006.
[26] Scott McMaster and Atif Memon, "Fault Detection Probability Analysis for Coverage-Based Test Suite Reduction", IEEE, 2007.
[27] Sreedevi Sampath, Sara Sprenkle, Emily Gibson and Lori Pollock, "Web Application Testing with Customized Test Requirements – An Experimental Comparison Study", 17th International Symposium on Software Reliability Engineering (ISSRE'06), 2006.
[28] Siripong Roongruangsuwan and Jirapun Daengdej, "Techniques for improving case-based maintenance", Assumption University, Thailand, 2003
[29] Siripong Roongruangsuwan and Jirapun Daengdej, "Test Case Reduction", Technical Report 25521. Assumption University, Thailand, 2009.
[30] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies", IEEE Trans. on Software Engineering, vol. 28, 2002.
[31] S. Elbaum, A. G. Malishevsky and G. Rothermel, "Prioritizing Test Cases for Regression Testing", In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 102-112, 2000.
[32] S. Elbaum, P. Kallakuri, A. G. Malishevsky, G. Rothermel, and S. Kanduri, "Understanding the effects of changes on the cost-effectiveness of regression testing techniques", Journal of Software Testing, Verification, and Reliability, 13(2):65-83, 2003.
[33] Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter and Gregg Rothermel, "An Empirical Study of Regression Test Selection Techniques", 2000.
[34] W. Eric Wong, J. R. Horgan, Saul London and Hira Agrawal, "A Study of Effective Regression Testing in Practice", 8th IEEE International Symposium on Software Reliability Engineering (ISSRE'97), 1997.
[35] W. Eric Wong, Joseph R. Horgan, Saul London and Aditya P. Mathur, "Effect of Test Set Minimization on the Fault Detection Effectiveness of the All-Uses Criterion", In *Proceedings of the 17th International Conference on Software Engineering*, pages 41-50, 1995.
[36] Xiaofang Zhang, Baowen Xu, Changhai Nie and Liang Shi, "An Approach for Optimizing Test Suite Based on Testing Requirement Reduction", Journal of Software (in Chinese), 18(4): 821-831, 2007.
[37] Xiaofang Zhang, Baowen Xu, Changhai Nie and Liang Shi, "Test Suite Optimization Based on Testing Requirements Reduction", International Journal of Electronics & Computer Science, 7(1): 9-15, 2005.
[38] Xue-ying MA, Bin-kui Sheng, Zhen-feng HE and Cheng-qing YE, "A Genetic Algorithm for Test-Suite Reduction", IEEE, China, 2006.
[39] Yanbing Yu, James A. Jones and Mary Jean Harrold, "An Empirical Study of the Effects of Test-Suite Reduction on Fault Localization", *Proceedings of ICSE'08*, Germany, 2008.