International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011) pp. 730-737

Extracting Test Cases by Using Data Mining; Reducing the Cost of Testing

Ali Ilkhani[1], Golnoosh Abaee[2]

[1] Department of Computer Science
Shahabdanesh Institute of Higher Education, Qoam, Iran
Ali.ilkhani@Shahabdanesh.ac.ir

[2] Department of Computer Science
Shahabdanesh Institute of Higher Education, Qoam, Iran
abaee@Shahabdanesh.ac.ir

Abstract: **In this paper Case-Based Reasoning and Data mining are used as efficient methods for effort estimation and automated testing has been investigated respectively If your software has many outstanding features but does not work properly due to lack of testing, your software is subjected to fail so in order to test them properly, the test results could help the developer to classify them in different categories such as different process models and different types of errors in each developing life cycle phase, then by having these classified results and using data mining methods and Case-Based Reasoning, it would be easy to have the new software's properties and estimate the future test cases in order to reduce the cost of testing phase and eventually the developing cost in similar upcoming projects. In this paper we try to emphasize on testing the similar software with similar test cases. To make it much more efficient, a case with different types of attributes is designed for each software which shows the behavior of it, then we evaluate any upcoming software by fining the most similar case for it from the stored cases and do the performed test cases for it. By estimating the proper set of domains for each attributes, we could increase the efficiency.**

Keywords**:** Cased Based Reasoning, Automated Testing, Effort Estimation, Test Cases, Pattern, Clustering**.**

I. Introduction

The most important factor to evaluate the software performance is how well it works and it becomes an essential activity in Software Engineering field these days. Testing is widely used in industry for quality assurance. Indeed, with the complexity, variety of software growing continuously, ensuring that it behaves according to the desired levels of quality and dependability becomes more crucial, and increasingly difficult and expensive. One of the important and crucial tasks in software engineering process is estimating the cost of the future software and tries to make it as realistic as possible and reduce the cost of development as well. Each and every research article on software testing start from claiming that testing is a very expensive activity. A recent study by the National Institute of Standards & Technology [1] found that "the national annual cost of inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion". One of the newly emerging challenges is how to minimize the amount of retesting after the software is modified. Some people believe that the program correctness can never be demonstrated through software testing [2].

There are so many ways that make the software fails:

- Due to wrong requirements
- Due to missing requirements
- Implementing the requirements are impossible
- Due to faulty design
- Due to faulty code
- …

Testing goals could be, faults identifications, we should find out, what fault caused the failure? Then we should search for the remedy, how we can fix the error or how we can delete the fault.

For this purpose documenting the test cases and the bugs and problems that occur in each tested product and also try to use them in future developing cases can help us to reduce the cost of testing, retesting during maintenance, also can help in estimating how much effort is needed for similar projects to be tested; Test documentation has four disciplines:

1.Test Plan :

Describes system and plan for exercising all functions and characteristics.

2.Test Specifications and Evaluations :

Details each test and defines criteria for evaluating each features.

3.Test Descriptions :

Test data and procedure for each test.

4.Test Analysis Report :

Results of each test.

Documentation of the test process records both the test to be performed and the result and analysis of those tests. Computer testing is too complex to formalize the process. Documentation is an integral part of a formalization of testing. Test documentation is important for conducting the test and during the maintenance. Test documentation should be continually updated. The more structured the documentation, the easier it is to be updated and reuse.

Some people think that testing means only code testing but testing should commence in the requirements phase and continue through the life of the project, we mostly call it as review. The test process that has been outlined in each phase of the life cycle should be documented.

On the other hand, Data mining and working on different types of its techniques to retrieve reliable predictions becomes a very interesting and popular topic recently in any aspects of industry and as it mentioned in the above paragraph, trying to reduce the cost of testing in order to reduce the developing cost by documenting the test results could be the effective solution to this aim.

There are various techniques as well as computational tools for data mining such as Linear Regression (LR), Neural Networks (NNs), Genetic Algorithms (GAs), Support Vector Machines (SVMs), Case-Based Reasoning (CBR) and so on.

This paper gives an overview on testing and data mining in the Introduction part, in the second and third part there is a review on Case-Based Reasoning as one of the main techniques in data mining and some methods for efforts estimation respectively, after that there is an overview on test documentation tool followed by the implementation of the proposed method

II. Cased-Based Reasoning

According to Aamodt and Plaza [3], CBR is able to utilize the specific knowledge of the previously experienced, concrete problem situation (cases). A new problem is solved by finding a similar past cases and reusing it in the new problem situation.

Case-Based Reasoning is to solve a new problem by remembering a previous similar situations and by reusing information and knowledge of that situation.

Case-Based Reasoning has been formalized for purposes of computer reasoning as a four-step process:

1. **Retrieve:** Given a target problem, retrieve cases from memory that is relevant to solving it.
2. **Reuse:** Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation.
3. **Revise:** Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise.
4. **Retain:** After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory.

In CBR, the primary knowledge source is not generalized rules but a memory of stored cases recording specific prior episodes. In CBR, new solutions are generated not by chaining, but by retrieving the most relevant cases from memory and adapting them to fit new situations. Thus in CBR, reasoning is based on remembering.

III. Effort Estimation

For a successful project, both good project management and good engineering are essential. Lack of either one can cause the project to fail. Project management activities can be viewed as having three major phases: project planning, project monitoring and control and project termination [4]. Effort estimation is one of the major issues in project planning along with process planning, schedule and resource estimation, Quality plans, Configuration management plans, risk management and project monitoring plans.

In effort or cost estimation, for a set of requirements it is desirable to know how much it costs to develop software and how much time it takes to finish it. We can perform estimation at any point in the software life cycle from feasibility to installation and maintenance phase. The main point is the accuracy of the estimate and it depends on the amount of reliable information someone gathers about the future software. From the experienced, it showed that, the estimation in the early stages of the project life cycle is not that much accurate and reliable due to the lack of information and unclear functionality of the system; But once the requirements are completely specified in for example Requirements and Analysis phases, more accurate effort estimates can be made compared to the estimate after the feasibility study.

In the last three decades, many quantities software cost estimation models have been developed. They ranged from empirical models such as Boehm's COCOMO models [5] to analytical models such as those in [6, 7, 8, 9]. An empirical model uses data from previous projects to evaluate the current project and derive the basic formulas from analysis of the particular database available [10, 11]. The analytical model, on the other hand uses formulas based on global assumptions, such as a rate at which developer solves problems and the number of problems available.

One common approach for estimating effort is to make it function of the project size and the equation of the effort is considered as

$$EFFORT = a * SIZE^{b}$$

Where a and b are constants [11] and project size is generally in KLOC or function points. Watson and Felix [12] analyze the idea of more than 60 projects done at IBM Federal Systems Division, ranging from 4000 to 467000 lines of delivered source code, and found that if the SIZE estimate is in thousands of delivered lines of code (KLOC), the total effort, E, in person-months (PM) can be given by the following equation:

$$E = 5.2 * (SIZE)^{.91}$$

There are other approaches as well like bottom-up approaches which the project divided in to tasks. Then first we estimate the divided different tasks and then the overall estimation is obtained. The first step is modules identification and classifies them into simple, medium and complex groups then we calculate the average coding efforts for each module and from that we find out the overall effort. We use these results for similar projects and estimate the efforts for them. As we see here we get the first idea of CBR in data mining which is using a judicious mixture of past data and experience for estimating the future effort cost for similar projects. This method is mostly suitable for small projects.

The other famous approach here is a COCOMO model developed by Boehm [6, 13] which implies that size is the primary factor cost; other factors have lesser effects. Here first the initial estimate of the developing effort from its line of code (KLOC) is obtained, then a set of 15 multiplying factors from different attributed of the project classified as Product Attributes like CPLX stated as product activity, Computer Attributes like TURN stated as computer turnaround time, Personnel Attributes like ACAP states as analyst capability and Project Attributes like TOOL states as use of software tools and range of them from Very Low to Very High are determined and finally estimate the effort by multiplying the initial estimate with all these multiplying factors. For initial estimation variable a and b can be replaced 3.2 and 1.05 accordingly if the system is Organic, 3.0 and 1.12 if the system is Semidetached and 2.8 and 1.20 if the system is Embedded.

As it is mentioned earlier, we can store our calculated programs efforts estimation based on the results we get from the mentioned methods and classify them based on a type of project. We can categorize them in various types such as:

- Types of Process Models like Water fall, Incremental, Spiral, …
- Types of Methodologies like Object Oriented, Agile, SSADM, …
- Types of Applications like Student Projects, Stock Market Projects, Health Applications
- Application Risks

IV. Why Automated Testing?

Testing is one of the most important phases in software life cycle especially in terms of money and time. Experts say whenever a developer ran out of time and money, he could stop testing. If someone wants to reduce the cost of developing the new project, he/she have to reduce the cost of testing; but the most important question is how?

A large part of current testing research aim at improving the degree of attainable automation, either by developing the advanced techniques for generating the test inputs, or, beyond test generation, by finding innovative support procedures to automate the testing process [14].

Software developers talks a bout using an automated testing tools continuously but how these tools can reduce the cost of testing phase; besides testing is not supposed to be conducted for coding phase only but also can use as a review during the rest of phases.

According to Ben-Gurion, Software testing activities are usually planned by human experts, while test automation tools are limited to execution of pre-planned tests only [15]. If for example the types of errors are defined and classified in each phase of the developing life cycle, they could be used in future predictions in similar types of projects, hence they could be avoided in different similar projects.

The common way to test a system is to design different test scenarios and test the system against them. As the system changes and new functionalities added, new test scenarios should be designed and run. The previously designed scenarios also should be run again to ensure that the new changes have not adversely affected any previous functionality. Thus the number of scenarios will increase as the system becomes bigger and bigger. Eventually the testing time will increase release by release

A scenario must be tested more than once and it could be tedious for any tester as it would be much more mechanical than intellectual task. This also increased the mistake ratio of the tester and might reduce the test accuracy, quality. A clear benefit of automation is the ability to run more tests in less time and to run them more often.

V. Test Cases in Web Applications

We can categorize the test cases based on different categories; some of them are presented in table 1 as follows:

Name	Items to Check

Content	*Semantic Errors:* in text based document
	Syntactic Errors : error in accuracy or completeness of information
Transactions	The work is done properly from the beginning till the end
User Interface	Compatible with UI standards, Navigation through the application properly reflects business functions and requirements
	Detect faults due to invalid assumptions about interface
	Check in different environment (browsers) and user categories
	Check all links, forms, client/server scripting, dynamic HTML, pop-up windows, …
Stress	When an invalid or out of band data is given
	Exercise the system beyond its maximum design load
	Checks for unacceptable loss of service and data
Database	Database data access method works properly without corruption
	Test Communication between web server and remote database
	The dynamic content of object must be transmitted to the client in a form that can be displayed to the end user
Compatibility	Different OSs, connection speeds, client/server processing speed
Component Level	Functional Testing
	Boundary Value Analysis
	Path Testing: each path through the program executed at least once.
Navigation	Links, frames and framesets, site maps and internal search engines
Configuration	*Client Side:* Hardware's, OSs, Browsers, Connectivity's, UI components
	Server Side: security (firewalls, encryption), Different versions of database, distributed servers configurations
Workload	Check response time under varying workload conditions
Volume	When Hard is full or when user uses the most of RAM
Usability	Understandability, User comfort ability, Display Characteristics, Personalization and accessibility for people who has disability
	System: Verify that only those users

Security	with access to the system and application(s) are permitted to access them
	Function/Data: Verify that user can access only those functions / data for which their user type is permitted
Special Conditions	Test reaction of system in critical conditions such as power cut in server and client side

Table 1: Web application test cases

Apart from the above classifications, faults can be categorized as below:

1. Algorithmic Fault
2. Syntax Fault
3. Computation and Precision Fault
4. Documentation Fault
5. Stress and Overload Fault
6. Capacity and Boundary Fault
7. Timing and Coordination Fault
8. Throughput and Performance Fault
9. Recovery Fault
10. Hardware and System Software Fault
11. Standards and Procedures Fault

VI. Using Clustering

Cluster analysis is the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity (Figure 1) depicts a typical sequencing of the first three of clustering steps [16], including a feedback path where the grouping process output could affect subsequent feature extraction and similarity computations.

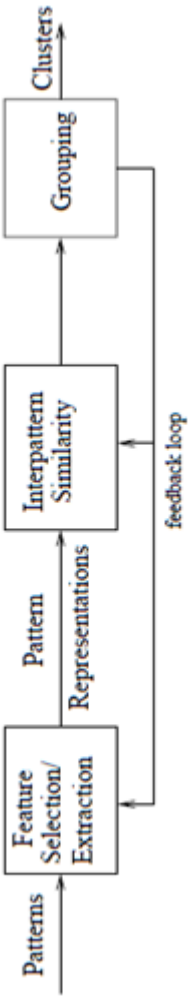

Figure 1: Stages in clustering

Pattern representation refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Feature selection is the process of identifying the most effective subset of the original features to use in clustering. Feature extraction is the use of one or more transformations of the input features to produce new salient features. Either or both of these techniques can be used to obtain an appropriate set of features to use in clustering. Pattern proximity is usually measured by a distance function defined on pairs of patterns. A variety of distance measures are in use in the various communities. The grouping step can be performed in a number of ways. The output clustering (or clustering) can be hard (a partition of the data into groups) or fuzzy (where each pattern has a variable degree of membership in each of the output clusters)[17].

VII. Convert software to case

The following terms and notations are used throughout this paper.

- A case (or feature vector) X is a single data item used by clustering algorithm. It typically consists of a vector of d measurements: X = ($x_1, \dots x_n$)

- The individual scalar components x_i of a pattern x are called features (or attributes).

- d is the number of features of each cases and we use as a dimension of the pattern in clustering issue.

- A distance measure (a specialization of a proximity measure) is a metric (or quasi_metric) on the feature space used to quantify the similarity of patterns.

VIII. Pattern and Similarity Measures

The most popular metric for continuous features is the Euclidean distance [18]:

$$d_2(\mathbf{x}_i, \mathbf{x}_j) = (\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2)^{1/2}$$

$$= \|\mathbf{x}_i - \mathbf{x}_j\|_2,$$

Which is a special case (p=2) of the Murkowski metric

$$d_p(\mathbf{x}_i, \mathbf{x}_j) = (\sum_{k=1}^{d} |x_{i,k} - x_{j,k}|^p)^{1/p}$$

$$= \|\mathbf{x}_i - \mathbf{x}_j\|_p.$$

IX. Define Case Attributes and Domains

Attribute	Domain
Contents that are shown in UI	D: {10, 20, 30}
Data Entry Page	D: {Y, N}
Scale of Transactions	D: {0, 1, 2}
Scale of Data Base	D: {10, 20, 30}
Process Model	D: {Linear, Waterfall, Prototype, RAD, Incremental, Spiral, Agile}

Figure 2: Case attributes and there domains

X. Using CBR to Find Appropriate Test Cases

The Figure 3 in the following page shows, how the best solution for problems is founded in the in Case-Based process. As it is shown, any software cases (as mentioned in section VII) has been defined in the first part and all performed test cases in the solution part.

After preparing a comprehensive set of cases for software, we will be able to use CBR life cycle to find a useful set. Since each software system has been converted to a case with some sufficient attributes, we can find some suitable cases for any upcoming software that must be tested. To make it more efficient, the cases should be extracted in a descending order.

To find more sufficient domain of attributes we could use previous test case that has been used for similar software.

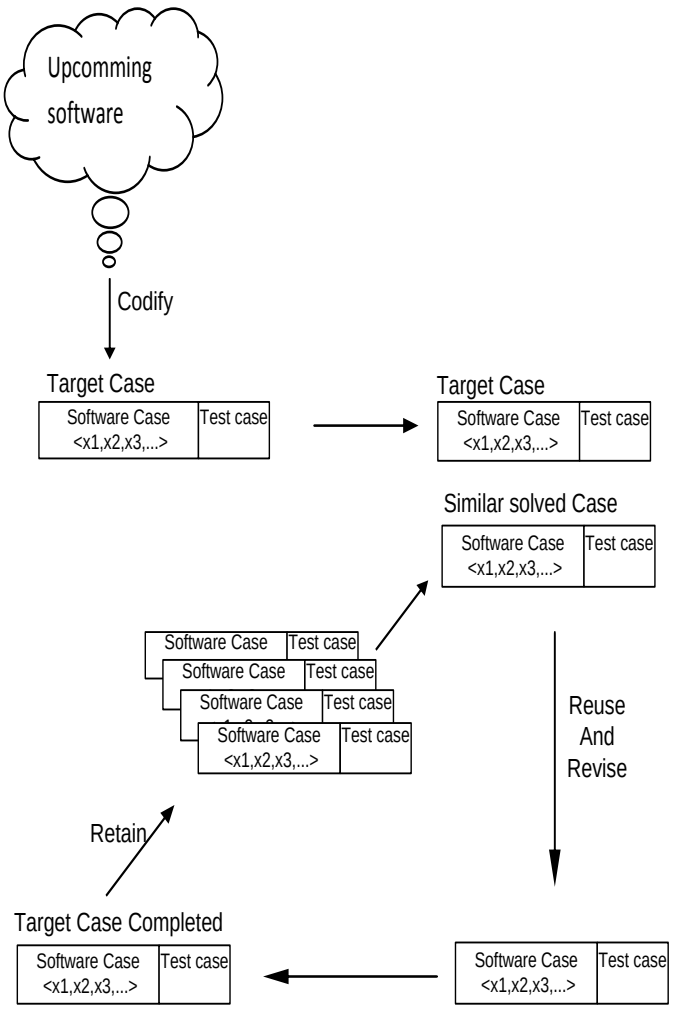

Figure 3: Finding best solution in Case-Based process

XI. Conclusion

Today, almost no model can estimate the cost of software with a high degree of accuracy; some studies estimate that testing can consume fifty percent, or even more, of the development cost so by trying to estimate the cost of testing, the cost of developing the new software will be reduced eventually.

Evaluating the derived results from the previous selected similar test cases will increase the chance of finding any cases based on the most nearest similar attributes.

On the other hand in CBR a new problem is solved by finding a similar past cases and reusing it in the new problem situation hence by documenting the test analysis results and findings and classified them in to different groups based on their process models, application types, error types, different developing life cycle phases, it could be possible to have a concrete set of data which can be used as a data pool for CBR.

Testing could be done automatically and manually, it doesn't matter which technique has been considered, the important factor is how to extract the attributes and gather the most accurate set of data which can be used as a new testing case for similar future software.

XII. Future Works

Considering derived results from various times we use this system, help us to have a better option in choosing domains and attributes. Using the pervious test cases will give the better performance in testing any upcoming software. In order to do this it seems, whatever the number of attributes is more, the range of domain in closer to reality.

XIII. References

[1] National Institute of Standards & Technology. "The Economic Impacts of Inadequate Infrastructure for Software Testing", Planning Report 02-3, May 2002.

[2] DeMillo R.A., and Offlut. A. J. "Constraint-Based Automatic Test Data Generation", IEEE Transactions on Software Engineering, 1991

[3] Agnar Aamodt and Enric Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," *Artificial Intelligence Communications* 7 (1994): 1, 39-52.

[4] Pankaj Jalote, "An Integrated Approach to Software Engineering", Third Edition

[5] B. W. Boehm. "Software engineering economics". Prentice Hall, Englewood cliffs, NJ, 1981

[6] G. Cantone, A. Cimitile and U. De Carlini, " A comparison of models for software cost estimation and management of software projects", in Computer Systems: Performance and Simulation, Elsevier Science Publishers B.V.,1986

[7] N.A.Parr, "An alternative to the Raleigh Curve Model for Software development effort", IEEE on Software Eng.May 1980

[8] L. H. Putnam, "A general empirical solution to the macro software sizing and estimating problem" IEEE Trans. Soft. Eng., July 1978, pp.345-361.

[9] *J.S.Pahariya, V. Ravia, M. Carra and M. Vasua*. "Computational Intelligence Hybrids Applied to Software Cost Estimation". International Journal of Computer Information Systems and Industrial Management Applications(IJCISIM), ISSN: 2150 -7988 Vol 2 (2010), pp104-112

[10] Roger S. Pressman, "Software Engineering, A Practitioner's Approach", Sixth Edition

[11] V.R.Basili. "Tutorial on Models and metrics for software management and engineering". IEEE Press, 1980.

[12] C.Warson and C.Felix . "A method of programming measurements and estimation". IBM system Journals, 16(1), Jan 1997

[13] B. W. Boehm. "Software engineering economics. IEEE Transaction on software engineering", 10(1): 135-152, Jan. 1984

[14] Antonia Bertolino"Software Testing Research: Achievements, Challenges, Dreams." IEEE, Future Of Software Engineering (FOSE'07). 0-7695-2829-5/07-2007.

[15] Oded Maimon, Lior Rokach, "Data Mining and Knowledge Discovery Handbook", Springer (2006)

[16] Jain A. K. and Dubes R. C. "Algorithms for Clustering Data", Prentice-Hall advanced,reference series. Prentice-Hall, Inc., 1996

[17] Jain A.K. and Murty M.N. and Flynn, P.J. "Data Clustering" ACM Computing Surveys, Vol. 31, No. 3, 1999.

[18] Mao, J. and Jain, A. K. "A self-organizing network for hyperellipsoidal clustering (HEC)." IEEE Trans. Neural Netw. 7, 16–29. 1996

Biographies:

Ali Ilkhani. Received the Bachelor degree in Computer Science from University of Kerman, Iran the Master degree in computer science from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran. He is presently a lecturer in Shahab e Danesh institute of higher education, member of software modeling group at Tehran Securities Exchange Technology Management Co. (TSETMC). His research interests include data mining and knowledge discovery.

Golnoosh Abaee. Received her B.E. in Software engineering from Islamic Azad University, Central Branch, Tehran, Iran and the master's degree, MCA (Master of Computer Application) from University of Mysore, Mysore, Karnataka state, India. She had worked in several software companies as a programmer, Project Manager and System Analyst for almost 4 years before she persuade her master's degree . She is presently works as a faculty in Shahab e Danesh institute of higher education and also is a lecturer in Islamic Azad University, Roodehen Branch. Her research interests are Software Testing, Data Mining, Mobile Cloud Computing.