Software Testing By Using Knowledge Based Approach

Amrita Singh and Er.Anil Pandey

Abstract: Knowledge Management concept for accessing the true and optimal result in the field of software testing. In this paper, we propose knowledge based approach for rectifying the error and give the result as much correct. First we go for testing categories and its approach for testing quality software. Intelligent means optimal result for help of best testing method. Testing is a process centered on the goal of finding defects in a system. There are numbers of method are present to test software. But a rule-based approach, the domain knowledge is represented by a set of production rules and less depends to tester. Data is represented by a set of facts. And predication is based on condition satisfaction.

Keywords: Knowledge, Knowledge management, Rule based, Software testing.

I. INTRODUCTION

Software engineering is deligent knowledge work. Management of knowledge in software engineering has received consideration. Knowledge is one of the organization's most important values, affecting its combativeness. One way to acquire organization's knowledge and make it available to all their associates is through the use of knowledge management system. Knowledge classification survey makes an important contribution to advancing knowledge in both science and engineering.

The knowledge management as a set of activities, Technique and tools are supporting the creation and transfer of software engineering (SE) knowledge management organization.

Effective knowledge management of the testing process is the key to improve the quality of software testing. Knowledge management has different features in software testing. One of the most important research questions is how to effectively integrate the knowledge management with the software testing process so that the knowledge assets can be spread and reused in software testing organizations [1].

In this paper, the current state of knowledge management in rule based was analyzed and the major existing problems were identified; rule based methods was proposed towards a knowledge Management system in rule based was designed and implemented. Knowledge management is the process of capturing and making use of an organization's collective expertise anywhere in the business [2].

Amrita Singh and Er.Anil Pandey are with Computer Science and Engineering, Invertis University Bareilly, Emails: amritas054@gmail.com, anil.p@invertis.org

The importance of knowledge management that has become a hot spot in international manages fields. In order to enhance its competitive power, many enterprises have initiatively taken knowledge management into their core business process, which drives the development of consultation business whose new business scope is knowledge management, and then a lot of software tools and systems about knowledge management have been developed by IT enterprise [3].

Knowledge management is an approach to discovering, capturing, and reusing both tacit (in people's heads) and explicit (digital- or paper-based) knowledge as well as the cultural and technological means of enabling the knowledge management process to be successful [5]. The knowledge management (KM) cornerstone is improving productivity by effective knowledge sharing and transfer [6]. The rule based system, knowledge is represented in the form of (If condition Then conclusion < action >) production rules [7].

Knowledge Management is one of the hottest topics today in both the industry world and information research world. In our daily life, we deal with huge amount of data and information. Data and information is not knowledge until we know how to dig the value out of it. This is the reason we need knowledge management. Unfortunately, there's no universal definition of knowledge management, just as there's no agreement as to what constitutes knowledge in the first place. We chose the following definition for knowledge management for its simplicity and broad context. Knowledge Management (KM) refers to a multi-disciplined approach to achieving organizational objectives by making the best use of knowledge. Knowledge management focuses on processes such as acquiring, creating and sharing knowledge and the cultural and technical foundations that support them.

A knowledge management system facilitates creation, access and reuse of knowledge, and its main goals are to promote knowledge growth, communication, preservation and sharing [8]. The knowledge management approach using software testing involves the process of detecting software discrepancies so that they can be corrected before they are installed into a live environment supporting operational business units [9].

II. KNOWLEDGE BASED APPROACH IN SOFTWARE TESTING

Software testing process normally involved black box and white box of testing. Black Box Testing is conducted on the application by test engineers or by domain experts to check whether the application is working according to customer requirements.

White box testing method is conducted by developer in a technical perception where as black box testing is conducted by test engineers with end-user perception. Programmers will conduct white box testing in a positive perception whereas tester will conduct black box testing with a negative perception where there is a more chance of finding more defects .The more defects you identify results in a quality system.

White box testing will not cover non functional areas. As functional requirements are also very important for production system those are covered in black box testing.

White Box Testing conducted on the source code by developers to check does the source code is working as expected or not is called white box testing.

As the source code is visible, finding and rectifying the problems is easy for developers. The defects that are identified in white box testing are very economical to resolve. To reduce the defects as early as possible white box testing is helpful. Software testing by using knowledge based approach is:

A. Test Cases

Test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly. The process of developing test cases can also help find problems in the requirements or design of an application a test case has components that describes an input, action or event and an expected response, to determine if a feature of an application is working correctly.

Testing an application's or program's working is not black-and-white. A program may work in one situation or condition, but may fail in another. It is up to the software tester, to ensure that a program works correctly in all possible conditions.

Example: Imagine a program which adds two numbers. The program must accept two numerical inputs, perform the addition and display the output. But certain conditions exist, which can hamper the functioning of the program. Like if one input is zero. The program should correctly display the output, which is the number itself. What if one number is negative? Then the program should perform subtraction and correctly assign a positive or negative sign to the answer.

Test cases, at root level, are used to measure how a program handles errors or tricky situations such as if one input is incorrect or if both inputs are incorrect. They are also expected to expose hidden logical errors in the program's code that have gone undetected.

1. Traffic Signal

1. Verify if the traffic lights are having three lights, (Green, Yellow, Red)

2. Verify that the lights turn on in a sequence

3. Verify that lights turn on in a sequence based on time specified,

(Greenlight min, Yellowloght10sec, Redlight1min)

4. Verify that only one light glow at a time

5. Verify if the speed of the traffic light can be accelerated as time specified based on the traffic

2. *Rule Based*

The people cross the road safely when the traffic light is green, and must stop when the traffic light is red.

IF “traffic light” is green

THEN action is go

IF “traffic light” is red

THEN action is stop

These statements represent the IF-THEN form are called production rules.

B. Knowledge Management (KM)

Knowledge management is based on the idea that an organisation’s most valuable resource is the knowledge of its people. Therefore, the extent to which an organisation performs well, will depend, among other things, on how effectively its people can create new knowledge, share knowledge around the organisation, and use that knowledge to best effect.

Knowledge management is a process that emphasises generating, capturing and sharing information know how and integrating these into business practices and decision making for greater organizational benefit.

C. Knowledge Based System (KBS)

Knowledge-based systems also process data and rules to output information and make decisions. In addition, they also process expert knowledge to output answers, recommendations, and expert advice. The Knowledge management system (KMS) can be implemented by using two components which are involved Knowledge management system (KMS) functionality and its related infrastructure through rule based system .The knowledge management system from the information technological point of view [1], knowledge is an entity differentiated from the information object in that there is an element of expert review and distillation where knowledge is concerned.

III. RULE BASED TESTING METHOD

The rule-based test pattern generation system fig1.The reaching input RI (n) is input signals. Test data generation rules are used by a rule interpreter to generate new test cases. A simulator executes these test cases. The simulator records coverage statistics for the test inputs. The rule interpreter analyzes the execution results and generates additional test cases. This cycle continues until the amount of requested coverage is achieved or until a user specified number of test cases have been generated and executed. The coverage metric in the rule-based approach is branch coverage.

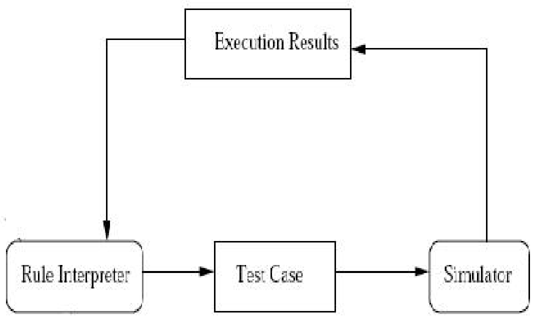

Fig1. Rule based Test Case Generation

A. Test Case Generation Process

The test generation process terms are test pattern, Test case, and branch point.

Test pattern: A test case R = {r_1, r_2… r_z}, each r_x ($1 \le x \le n$) input bit x_y € {0, 1}.

Test Case: test case R = {r_1, r_2… r_n}, each r_x ($1 \le y \le z$) input pattern. For example, R = {r_1, r_2} = {(0, 1), (0, 0)} is a test case.

Branch Point: let N be the number of nodes in the diagram for n €N, n is a branch point if there is than one outgoing arc.

R reaches p. Given a branch point p and a test case

R ={r_1, r_2… r_m}, R reaches p if p is not covered by

$R_{1\dots}$ r_{m-1}, but covered by r_m,

The test generation processes are:

1. Generate an initial test case R_0= {r_{01}, r_{02}….r_{0n}},

($n \ge 1$) by some technique .Set x = 0.

2. Simulate the Rule –based Test case Generation model with R_0 and collect branch coverage statistics, include branch coverage for each r_{0y} ($1 \le y \le n$) .

3. While (x < max and coverage <100%), for a branch Point p_z which is not covered yet:

(i) Determine a subsequence $Q_{m\ \{}r_x1$, r_{x2}… r_{xm}} of R_x = {r_{x1}, r_{x2}…, r_{xm}} (m<n) such that Q_m reaches p_k. Let Qm_{-1} = {r_{x1}, r_{x2}… r_{xm-1}}.
(ii) Usingg test generation rules, generate additional input Patterns Qm’ $_{=\ \{r’xm\dots rxm’\}}$, (m’ > m)
(iii) The new test sequence concatenates patterns in Qm’$_:$ $R_{x+1\ =}\ Q_{m-1}$
(iv) Simulate Rx_{+1}
(v) Increment x

We assume that there is an initial test sequence of input patterns (1) that has been simulated and for which we have step-by-step coverage information (2). As long as the coverage is not complete, and as long as a userdefined number of generation steps have not been exceeded, we do the following: first we determine a branch point we want to cover. Then we determine the subsequence that reaches the branch point (3i). Is more likely to execute the uncovered branch? The simulator executes this new test sequence and records coverage information.

B. Test Data Generation Rules

Test data generation rule consists of two parts: the precondition and the test generation effect. The precondition of a rule describes situations in which the rule could be applied. The test generation effect describes how to generate new test sequences. Three types of rules:
1. Rules with preconditions that apply to any statement
2. Rules for specific types of statements
3. Rules that do not have preconditions and apply to all statements P is an uncovered branch point which we want to cover.p’ refers to a branch point that directly dominates p, p’$>_d$ p. r_m is the last test pattern in R that reaches p.

Test Case Generation Rules 1:

Precondition: │RI (P) │is less than a given number max
Test generation effect: Generate combinations of values for RI (p). Rule 2 to Rule 7 will not be executed.
Rule 1 is applicable when the length of RI (p) is small.
For example, if │RI (p) │ is 2, only four new test patterns (00, 01, 10, 11) will be generated that contain all the possible values. We do not have to consider the statement type and possible relationships among RI (p).Thus it makes generation easy and fast. However this rule is not 1024 test cases will be produced.

For IF/ELSIF and WHILE statements are the new patterns are generated depending on whether or not the condition contains constants.
1 .p contains constants:

Test Case Generation Rules 2:

Precondition: the statement represented by p is IF, ELSIF or WHILE and p contains constants.
Test generation effect: EACH constant c contained in p, do:
EVERY u € RI (p) that │u│=│c│, set the value of u = c, > c and < c in r_m, in two ways:
(i) Set each u one at a time.

(ii) Set all us at once.

The rule, for a given constant contained in p, which reaching input u € RI (p) has the same dimensionality as the constant. Then we set these reaching inputs to the value of the constant, a value greater than the constant and a value smaller than the constant to form r'_m .There can be more than one such u, this should be done in two ways:

(i) Setting one u at a time.

(ii) Setting all us at once, either all equal to c, all greater than c or all smaller than c. For example, in Fig. 2 (I), both e and h have the same dimensionality as constant c =0101. Given r_m = 00011011, we first generate test patterns by changing the value of e without changing h. This produces 01011011 (set e equal to c), 01101011 (e > c), 01001011(e < c) (Fig. 2 (II). Then we change the value of h without changing e. This produces 00010101, 00010110 and 00010100. Finally, we change e and h together which produces 01010101, 01100110 and 01000100. For the other constant"1010" the same steps are executed.

2. P may or may not contain constants:

Test Case Generation Rules 3:

Precondition: the statement represented by P is IF, ELSIF or WHILE and it may or may not contain constants

Test generation effect:

(i) For each u € RI (p), set u to the NOT value of u in r_m to form r'_m in two ways:

(a) Set each u one at a time.

(b) Set all us at once.

(ii) Let $u_1, u_2\ldots, u_n$ be the reaching inputs such that u_x € RI (p), $1 \le x \le n$

(a) Generate combination of reaching inputs of 2, 3 and 4 inputs u_x at a time. The combination has the following format:

= all pairs: $\{u_x, u_y\}$, $1\le x, y\le n$ and $x \ne y$.

= all triplets: $\{u_x, u_y, u_t\}$, $1\le x, y, t \le n$ and $x \ne y \ne t$. = all quadruplets: $\{u_x, u_y, u_t, u_k\}$, $1\le x, y, t, k$€ n and $x \ne y \ne t \ne k$

(b) Set all u in each combination to the value 0 one at a time whiles other u in the same combination to the value1. Same example shown above, with Rule 2 (i), 11101011 is generated by replacing e with $\bar{e}$ in r_m without changing h (Fig. 2 (III). 00010100 is generated by setting h without changing e. 11100100 are generated by changing both e and h. With Rule 2 (ii), a combination of length 2 {e, f} is generated. If │R (p) │>2, different combinations will be generated which produce more test patterns.

The main idea of Rule 2 and Rule 3 is to set those reaching inputs based on the constants contained in the branch point p .We set the reaching inputs with values equal to, less than and greater than the constants repectively. Some reaching inputs cannot be set to the constants because their length does not match that of any constant, or because the branch condition does not contain a constant. In this case, we replace them with their logical NOT value in r_m. We do not know the relationships between reaching inputs. To consider relationships, we change values for combinations of two, three, and four input signals. Combinations of more than four result in too many new patterns and test cases.

Test Case Generation Rule 4 and 5 apply to CASE statements.

Test Case Generation Rules 4:

Precondition: the statement represented by p is WHEN and the constant used in p is c.

Test generation effect:

A.each u € RI (p) such that │u│ =│c│, set u = c to form $r_{m'}$ in two ways:

(i) Set each u one at a time.

(ii) Set all us at once.

B. for each reaching input u € RI (p) that │u│>│c│ set every sequential │c│bits of u to the value one at a time.

The idea of this rule is to set the reaching inputs with the constant used in the WHEN statement. The examples are shown in Fig. 2 (IV) and (V). Given r_m = 101011, in step (1), we determine │sel│= │01│, so the value of is changed to 01 without changing en Then 011011 is generated. In step (2), │en│=4>│01│, so we set the 1st and 2nd bit of en to 01 and then set the 2nd and 3rd to 01 and so on. Thus we generate 100111, 101011 and
101001.

Test Case Generation Rules 5:

Precondition: the statement represented by p WHEN OTHERS.

Test generation effect:

(a) determine all constants c that do NOT occur in any P' € DOM (p) (dominators of p).

(b) For every such c, perform Rule 5.

The idea of Rule 4 is to set the reaching inputs to constants that do not appear in WHEN statements.
Those WHEN statements are must be the dominators of p.

Test Case Generation Rules 6:

Precondition: None.

Test generation effect:

(a) The Generate two special test patterns: r_1 with all '1's, with all '0's.

(b) For each u € RI (p), set u = 0 in r_1 one signal at a time.

(c) For each u € RI (p), set u = 1 r_2 in one signal at a time.

This rule generates new test cases based on two special test cases: all '1's and all '0's. Then we set every reaching input to '0' in the all '1' test case and to '1' in the all '0' test case. An example is shown in Fig. 2 (VII).(s_1 , s_2 ,s_3) is the input vector, but for a given branch point p, assume RI(p) = {s_1,s_2 }.First the rule requires the two special test patterns r_1 = 111111111 and r_2 = 000000000Then we set s_1,s_2 to '0' one at a time in r_1 and 000111111 and 111001111 are generated. Similarly, we set s_1, s_2 to '1' one at a time in r_2. This generated 111000000 and 000110000.

Test Case Generation Rules 7:

Test generation effect:

For each newly generated test pattern r, generate another new test pattern r' = r¯ this rule can help to satisfy the signal criterion that every input bit can make a transition from '0' to '1' and then from '1' to'0'.

Input V = e [3:0], f [3:0]

P_k:

If (e>"0101" and <"1010") then
$R_{m:}$ (00011011)

(I) Simple code for IF statement

```
 e   h
01011011
01101011
01001011
00010101
00010110
00010100
01010101
01100110
01000100
10101011
10111011
10011011
00011010
00011011
00011001
10101010
10011001
10001000
```

(II) Generated Test Case using Rule 2 for example (i)

```
    e    f
(e) 11101011
    00010100
    11100100←
(f) 00000000
```

(III) Generated Test Case using Rule3 for example (vi)

Input v = sel [1:0], en [3:0]
$P_{k:}$
Case sel is
....

When "01"→ .....
……..

(IV) Simple code for CASE statement

```
 sel  en
(1)011011←
(2)100111
   101011
   101001
```

(V) Generated Test Case using Rule 4 for example (iv)

V: a1 [2:0] a2 [1:0] a3 [3:0]

RI: a1, a2

```
111111111
000000000
000111111
111001111
111000000
000110000
```

(VI) Rule 7

Fig.2 Test Case Generation Rules

IV. CONCLUSION

In this paper knowledge based propose a new idea and method to solve the problems for us, but software testing has its features. Knowledge based in software testing is very important to improve the quality of software products and the economic benefit. This paper proposes a new test generation method based on specific rules for test pattern generation. The results show that a rule based test generation will yield higher coverage. We plan to investigate further other types of knowledge that could be used to determine more sophisticated rules. In the future plan to the investigate test Generation heuristics that can explore timing characteristics. Test should applied using a variety of test methods other than random pattern generation and the rule based method.

ACKNOWLEDGEMENT

The author of this paper would like to thank our honorable teacher Professor Mr. Zubair Khan (Head of Department Computer Science and Engineering Invertis University Bareilly, U.P) for his great help and precious suggestion about this work.

REFERENCES

[1] Research and Implementation of Knowledge Management Methods in Software Testing Process Liu Xue-Mei1, 3 GU Guochang1 Liu Yong-Po2 Wu Ji2 1 College of Computer Science and Technology, Harbin Engineering University Heilongjiang 1150001 , China; School of Computer Science and Technology Beijing University Aeronautics and

Astronautics, Beijing 100083, China; 3 Beijing City University Beijing, 100083, China
[2] Toward a Practical Solution for Capturing Knowledge for Software Projects: Seija Komi Sirviö and Annukka Mäntyniemi, *VTT Electronics* Veikko Seppänen, *University of Oulu*
[3] Yongpo Liu Ji Wu School of Computer Science and Technology Beihang University Beijing, China liuypo@sei.buaa.edu.cn Xuemei Liu Guochang Gu College of Computer Science and Technology Harbin Engineering University Heilongjiang, China xuemliu@163.com
[4] Anneliese Andrews, Andrew O'Fallon School of Electrical Engineering and Computer Science Washington State University Pullman, WA 99164 aandrews, aofallon @eecs.wsu.edu Tom Chen Department of Electrical and Computer Engineering Colorado State University Fort Collins, CO 80523 chen@engr.colostate.edu
[5] A three-tier knowledge management scheme for Software engineering support and innovation: Richard D. Corbin a, Christopher B. Dunbar b, Qiuming Zhu c,*
[6] Knowledge Management in Practice: The Case Of Agile Software Development Meira Levy Deutche Telekom Laboratories@ Ben Gurion University and Department of Industrial Engineering & Management Ben-Gurion University of the Negev, Israel Orit Hazzan Department of Education in Technology and Science Technion - Israel Institute of Technology
[7] Louardi Bradji, Mahmoud Boufaida LIRE Laboratory, Mentouri University of Constantine Ain El Bey, Constantine, Algeria E-mail: bradjilouardi@yahoo.fr, mboufaida@umc.edu.dz Received July 23, 2011; revised September 12, 2011; accepted September 22, 2011
[8] Knowledge Management in Software Engineering Environments: Ana Candida Cruz Natali Richard De Almeida Falbo Computer Science Department, Federal University of Espírito Santo, Fernando Ferrari Avenue, CEP 29060 900, Vitória - ES – Brazil {anatali,} falbo}@inf.ufes.br
[9] Ted E. Lee Department of Management Information Systems Fogelman College of Business & Econ, The University of Memphis elee@memphis.edu
[10] B.Lakshmana Rao, G.V Konda Reddy and G.Yedukondalu, "Privacy Preserving in Knowledge Discovery and Data Publishing", International Journal of Emerging Trends in Electrical and Electronics (IJETEE – ISSN: 2320-9569) Vol. 3, Issue. 3, May-2013.

Amrita Singh is a pursuing M.Tech in Computer Science and Engineering from Invertis University Bareilly. Her Research interest area is Software Engineering. Email-amritas054@gmail.com

Er. Anil Pandey is an Assistant Professor of Computer Science and Engineering from Invertis University Bareilly. His Research area is Software Engineering. Email-anil.p@invertis.org