Volume 2, Issue 4, April 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Knowledge Management: through the Classification of Unit Testing Techniques

M.Sravani*
M. Tech Student
IT Department, SVEC, Tirupati
sravani999it@gmail.com

S. Munwar
Assistant Professor
IT Department, SVEC, Tirupati

Abstract— **Now a day's Knowledge classification analyze the classification makes a significant contribution to advancing knowledge in both science and engineering. It is a way of investigating the relationships between the objects to be classified and identifies gaps in knowledge. Classification in engineering also has a practical application; it supports object selection.**
Classifications have advanced knowledge in three ways as the following

- **By providing a set of unifying constructs.**
- **By understanding interrelationships**
- **By identifying knowledge gaps.**

They can help mature Software engineering knowledge, as classifications constitute an organized structure of knowledge items. Till date, there have been few attempts at classifying in Software Engineering. In this research, we examine how useful classifications in Software Engineering are for advancing knowledge by trying to classify testing techniques. The paper presents a preliminary classification of a set of unit testing techniques. To obtain this classification, we enacted a generic process for developing useful Software Engineering classifications. The proposed classification has been proven useful for maturing knowledge about testing techniques, and therefore, SE, as it helps to:
1) Provide a systematic description of the techniques,
2) Understand testing techniques by studying the relationships among techniques (measured in terms of differences and similarities),
3) Identify potentially useful techniques that do not yet exist by analyzing gaps in the classification, and
4) Support practitioners in testing technique selection by matching technique characteristics to project characteristics.

Keywords— **Classification, Software engineering, Software testing, test design techniques, unit testing techniques.**

I. Introduction

Software Engineering (SE) has aspects that disqualify it as a genuine engineering discipline. A prominent point is the immaturity of the theoretical knowledge in some areas of SE. In science and engineering, knowledge matures as the investigated objects are classified. Mature knowledge is not a sequential heap of pieces of knowledge, but an organized structure of knowledge items, where each piece smoothly and elegantly fits into place, as in a puzzle. Classification groups similar objects to form an organization. Examples are the classification of living beings in the natural sciences, diseases in medicine, elements in chemistry, architectural styles in architecture, materials in civil engineering, etc. Classifications have advanced knowledge in three ways as the following:

By providing a set of unifying constructs: Such constructs systematically characterize the area of research. To facilitate knowledge sharing, disciplines typically develop classifications. These classifications then provide a common terminology for communication.

By understanding interrelationships: For example, the periodic table of elements that Mendeleyev built in the 1860s had a profound impact on the understanding of the structure of the atom. On the contrary, it is hard to pigeonhole bacteria within the classification of living beings because relatively little is known about them.

By identifying knowledge gaps: For instance, the gaps in the classification of chemical elements prompted a search for further knowledge. Properties of elements like gallium and germanium were predicted before they were discovered years later. However, classifications can serve other purposes apart from providing a useful organization of knowledge. In medicine, for example, the classification of diseases has two main aims: prediction (separating diseases that require different treatments) and provision of a basis for research into the causes of different types of disease. In the case of engineering, this other purpose is usually decision making support.

II. BACKGROUND AND RELATED WORK

Experience has shown that as software is fixed, emergence of new and/or reemergence of old faults is quite common. Sometimes reemergence occurs because a fix gets lost through poor revision control practices (or simple human error in revision control). Often, a fix for a problem will be "fragile" in that it fixes the problem in the narrow case where it was first observed but not in more general cases which may arise over the lifetime of the software. Frequently, a fix for a problem in one area inadvertently causes a software bug in another area. Finally, it has often been the case that when some feature is redesigned, the same mistakes that were made in the original implementation of the feature were made in the redesign.Therefore, in most software development situations it is considered good practice that when a bug is located and fixed, a test that exposes the bug is recorded and regularly retested after subsequent changes to the program. Although this may be done through manual testing procedures using programming techniques, it is often done using automated testing tools. Such a test suite contains software tools that allow the testing environment to execute all the regression test cases automatically; some projects even set up automated systems to automatically re-run all regression tests at specified intervals and report any failures (which could imply a regression or an out-of-date test). Common strategies are to run such a system after every successful compile (for small projects), every night, or once a week. Those strategies can be automated by an external tool, such as BuildBot.Regression testing is an integral part of the extreme programming software development method. In this method, design documents are replaced by extensive, repeatable, and automated testing of the entire software package at every stage in the software development cycle. Traditionally, in the corporate world, regression testing has been performed by a software quality assurance team after the development team has completed work. However, defects found at this stage are the most costly to fix. This problem is being addressed by the rise of developer testing. Although developers have always written test cases as part of the development cycle, these test cases have generally been either functional tests or unit tests that verify only intended outcomes. Developer testing compels a developer to focus on unit testing and to include both positive and negative test cases.

III. Knowledge Management in ATPG

ATPG (acronym for both Automatic Test Pattern Generation and Automatic Test Pattern Generator) is an electronic design automation method/technology used to find an input (or test) sequence that, when applied to a digital circuit, enables testers to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects. The generated patterns are used to test semiconductor devices after manufacture, and in some cases to assist with determining the cause of failure (failure analysis.[1]) the effectiveness of ATPG is measured by the amount of modeled defects, or fault models, that are detected and the number of generated patterns. These metrics generally indicate test quality (higher with more fault detections) and test application time (higher with more patterns). ATPG efficiency is another important consideration. It is influenced by the fault model under consideration, the type of circuit under test (full scan, synchronous sequential, or asynchronous sequential), the level of abstraction used to represent the circuit under test.

Fig. 4. Application Window

Table 1
Four parameter values on application

Test C.No.	Input	Expected Behavior	Observed behavior	Status P = Passed F = Failed

1. Basics of ATPG

A defect is an error introduced into a device during the manufacturing process. A fault model is a mathematical description of how a defect alters design behavior. A fault is said to be detected by a test pattern if, when applying the pattern to the design, any logic value observed at one or more

of the circuit's primary outputs differs between the original design and the design with the fault. The ATPG process for a targeted fault consists of two phases: fault activation and fault propagation. Fault activation establishes a signal value at the fault model site that is opposite of the value produced by the fault model. Fault propagation moves the resulting signal value, or fault effect, forward by sensitizing a path from the fault site to a primary output.

ATPG can fail to find a test for a particular fault in at least two cases. First, the fault may be intrinsically undetectable, such that no patterns exist that can detect that particular fault. The classic example of this is a redundant circuit, designed so that no single fault causes the output to change. In such a circuit, any single fault will be inherently undetectable.

Second, it is possible that a pattern(s) exist, but the algorithm cannot find it. Since the ATPG problem is NP-complete (by reduction from the Boolean satisfiability problem) there will be cases where patterns exist, but ATPG gives up since it will take an incredibly long time to find them (assuming P≠NP, of course).

2. The Stuck-at fault model

In the past several decades, the most popular fault model used in practice is the single stuck-at fault model. In this model, one of the signal lines in a circuit is assumed to be stuck at a fixed logic value, regardless of what inputs are supplied to the circuit. Hence, if a circuit has n signal lines, there are potentially 2n stuck-at faults defined on the circuit, of which some can be viewed as being equivalent to others. The stuck-at fault model is a logical fault model because no delay information is associated with the fault definition. It is also called a permanent fault model because the faulty effect is assumed to be permanent, in contrast to intermittent faults which occur (seemingly) at random and transient faults which occur sporadically, perhaps depending on operating conditions (e.g. temperature, power supply voltage) or on the data values (high or low voltage states) on surrounding signal lines. The single stuck-at fault model is structural because it is defined based on a structural gate-level circuit model.

A pattern set with 100% stuck-at fault coverage consists of tests to detect every possible stuck-at fault in a circuit. 100% stuck-at fault coverage does not necessarily guarantee high quality, since faults of many other kinds -- such as bridging faults, opens faults, and transition (aka delay) faults -- often occur.

3. Sequential ATPG

Sequential-circuit ATPG searches for a sequence of vectors to detect a particular fault through the space of all possible vector sequences. Various search strategies and heuristics have been devised to find a shorter sequence and/or to find a sequence faster. However, according to reported results, no single strategy/heuristic out-performs others for all applications/circuits. This observation implies that a test generator should include a comprehensive set of heuristics .Even a simple stuck-at fault requires a sequence of vectors for detection in a sequential circuit. Also, due to the presence of memory elements, the controllability and observability of the internal signals in a sequential circuit are in general much more difficult than those in a combinational circuit. These factors make the complexity of sequential ATPG much higher than that of combinational ATPG.

Due to the high complexity of the sequential ATPG, it remains a challenging task for large, highly sequential circuits that do not incorporate any Design For Testability (DFT) scheme. However, these test generators, combined with low-overhead DFT techniques such as partial scan, have shown a certain degree of success in testing large designs. For designs that are sensitive to area and/or performance overhead, the solution of using sequential-circuit ATPG and partial scan offers an attractive alternative to the popular full-scan solution, which is based on combinational-circuit ATPG.

4. ATPG and nanometer technologies

Historically, ATPG has focused on a set of faults derived from a gate-level fault model. As design trends move toward nanometer technology, new manufacture testing problems are emerging. During design validation, engineers can no longer ignore the effects of crosstalk and power supply noise on reliability and performance. Current fault modeling and vector-generation techniques are giving way to new models and techniques that consider timing information during test generation, that are scalable to larger designs, and that can capture extreme design conditions. For nanometer technology, many current design validation problems are becoming manufacturing test problems as well, so new fault-modeling and ATPG techniques will be needed.

Algorithmic methods

Testing very-large-scale integrated circuits with a high fault coverage is a difficult task because of complexity. Therefore many different ATPG methods have been developed to address combinatorial and sequential circuits. Early test generation algorithms such as Boolean difference and literal proposition were not practical to implement on a computer.

•The D Algorithm was the first practical test generation algorithm in terms of memory requirements. The D Algorithm introduced D Notation which continues to be used in most ATPG algorithms.

•Path-Oriented Decision Making (PODEM) is an improvement over the D Algorithm. PODEM was created in

1981 when shortcomings in D Algorithm became evident when design innovations resulted in circuits that D Algorithm could not realize.

• Fan-Out Oriented (FAN Algorithm) is an improvement over PODEM. It limits the ATPG search space to reduce computation time and accelerates backtracking.

• Methods based on Boolean satisfiability are sometimes used to generate test vectors.

• Pseudorandom test generation is the simplest method of creating tests. It uses a pseudorandom number generator to generate test vectors, and relies on logic simulation to compute good machine results, and fault simulation to calculate the fault coverage of the generated vectors.

Test call generators {TCGs) are revenue assurance solutions that replicate events on a telecoms network [1] to identify potential revenue leakage and to help achieve regulatory compliance. Both cellular and fixed-line telecom operators utilize test call generators to independently test their networks for call detail record (CDR) reconciliation and validate call start-time/duration metering and telecommunications rating. TCGs are mission-critical tools utilized by telecom operator revenue assurance departments to improve revenue capture and to validate network integrity.

Revenue assurance and test call generators

Network integrity test call generators are too often seen solely as an engineering tool to gauge the quality of network functions. Automated test call generators can also enhance revenue assurance by providing measures of completeness, accuracy, and timeliness.

Cellular and fixed line network testing Test call generators have the capability to test both GSM and fixed-line networks through utilizing various hardware components. The components of a TCG system consists of both hardware and software, the key components are defined as:

• Network hardware tester units - These can be either GSM units for 2G/3G testing, and fixed-line units for analogue testing.

• System software - Generally consists of the following software modules; (1) System controller to manage the automated call execution process, (2) CDR importer to import the corresponding Operator CDRs, (3) Matching algorithm to match the operator CDRs and (4) Rating module to independently rate the CDRs.

Test call generation – revenue assurance call testing

Revenue assurance via testing

while various processes make a contribution to Revenue Assurance, it is - service usage from the subscriber's views that is receiving the most attention among Wireless Operators. Test Systems are available that emulate the subscriber's service usage within the operational network, and check the corresponding billing records. These are completely automated and under the direct control of the groups who validate the billing process and measure the quality of service. Though these techniques are comparatively new to Operators, they are becoming a priority across the industry for completely understanding and managing Revenue Assurance and QoS (service quality).

TCG's are utilized by telecom operators to consolidate revenue assurance strategies. They provide automated testing by executing live calls on the operators network to identify potential network performance issues and revenue leakage. TCGs produce independent rated CDRs that are reconciled against the operators CDRs to validate CDR integrity and to ultimately uncover lost revenue. Some of the services that TCGs provide are:

1. Real-time testing for multiple call and data services e.g. voice, SMS, MMS, HTTP, mobile TV, video calling, content download (games, ringtones..)
2. End to end call detail record reconciliation (from switch to billing)
3. Verification testing of new tariffs
4. CDR matching reconciliation
5. Call rating validation for interconnect and retail billing
6. Regulatory compliance testing (Ofcom/Sarbanes-Oxley)
7. Network performance testing to validate new network components

Compare with manual testing.

Test automation is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions [1]. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.

There are two general approaches to test automation:

• Code-driven testing. The public (usually) interfaces to classes, modules, or libraries are tested with a variety of input arguments to validate that the results that are returned are correct.

• Graphical user interface testing. A testing framework generates user interface events such as keystrokes and mouse clicks, and observes the changes that result in the user interface, to validate that the observable behavior of the program is correct.

Test automation tools can be expensive, and it is usually employed in combination with manual testing. It can be made

cost-effective in the longer term, especially when used repeatedly in regression testing.

One way to generate test cases automatically is model-based testing through use of a model of the system for test case generation, but research continues into a variety of alternative methodologies for doing so.

What to automate, when to automate, or even whether one really needs automation are crucial decisions which the testing (or development) team must make. Selecting the correct features of the product for automation largely determines the success of the automation. Automating unstable features or features that are undergoing changes should be avoided.

Code-driven testing

A growing trend in software development is the use of testing frameworks such as the unit frameworks (for example, JUnit and NUnit) that allow the execution of unit tests to determine whether various sections of the code are acting as expected under various circumstances. Test cases describe tests that need to be run on the program to verify that the program runs as expected.

Code driven test automation is a key feature of Agile software development, where it is known as Test-driven development (TDD). Unit tests are written to define the functionality before the code is written. Only when all tests pass is the code considered complete. Proponents argue that it produces software that is both more reliable and less costly than code that is tested by manual exploration. It is considered more reliable because the code coverage is better, and because it is run constantly during development rather than once at the end of a waterfall development cycle. Because the developer discovers defects immediately upon making a change, when it is least expensive to fix. Also, since the only code that is written is what is required to make the tests pass, the tendency to write too much code is removed. Finally, rework is safer. When code is made faster or is cleaned-up, all of the tests that passed must continue to pass or the reworked code is not working as it should.

Graphical User Interface (GUI) testing

Many test automation tools provide record and playback features that allow users to interactively record user actions and replay it back any number of times, comparing actual results to those expected. The advantage of this approach is that it requires little or no software development. This approach can be applied to any application that has a graphical user interface. However, reliance on these features poses major reliability and maintainability problems. Relabeling a button or moving it to another part of the window may require the test to be re-recorded. Record and playback also often adds irrelevant activities or incorrectly records some activities.

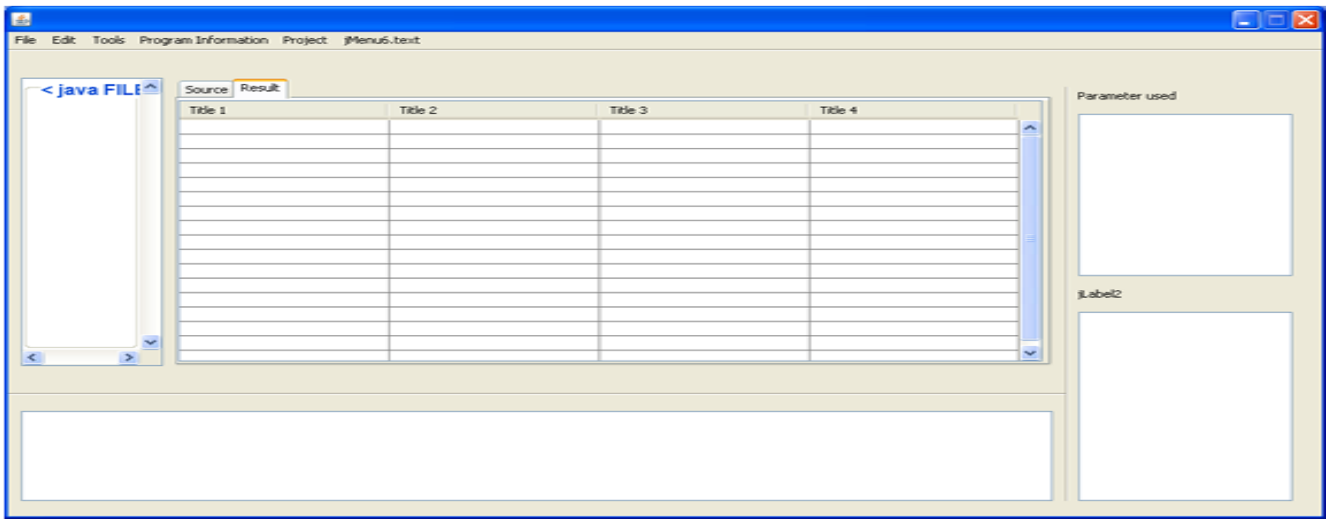
A variation on this type of tool is for testing of web sites. Here, the "interface" is the web page. This type of tool also requires little or no software development. However, such a framework utilizes entirely different techniques because it is reading html instead of observing window events.

Another variation is script less test automation that does not use record and playback, but instead builds a model of the application under test and then enables the tester to create test cases by simply editing in test parameters and conditions. This requires no scripting skills, but has all the power and flexibility of a scripted approach. Test-case maintenance is easy, as there is no code to maintain and as the application under test changes the software objects can simply be re-learned or added. It can be applied to any GUI-based software application

IV. RESULTS

Table 2 shows the model output of ATPG

Table 2: Output of ATPG



V. CONCLUSION AND FUTURE WORK

Although knowledge management in unit testing manual tests may find many defects in a software application, it is a laborious and time consuming process. In addition it may not be effective in finding certain classes of defects. Test automation is a process of writing a computer program to do testing that would otherwise need to be done manually. Once tests have been automated, they can be run quickly. This is often the most cost effective method for software products that have a long maintenance life, because even minor patches over the lifetime of the application can cause features to break which were working at an earlier point in time.

REFERENCES

[1] V.R. Basili, F. Shull, and F. Lanubile, "Using Experiments to Build a Body of Knowledge," Proc. Third Int'l Performance Studies Int'l Conf., pp. 265-282, July 1999.
[2] L. Bass, P. Clements, R. Kazman, and K. Bass, Software Architecture in Practice. Addison-Wesley, 1998.
[3] M J. Baxter, Exploratory Multivariate Analysis in Archaeology. Edinburgh Univ. Press, 1994.
[4] A. Bertolino, SWEBOK: Guide to the Software Engineering Body of Knowledge, Guide to the Knowledge Area of Software Testing, 2004 version, chapter 5. IEEE CS, 2004.
[5] R. Chillarege, "Orthogonal Defect Classification," Handbook of Software Reliability Eng., chapter 9, Mc Graw-Hill, 1996.[1]

AUTHOR'S BIOGRAPHY

M.Sravani received the B.Tech Information Technology from JNTUA, Anantapur, India in 2010 and pursuing his Master's degree in Software Engineering from the Sree Vidyanikethan Engineering College, Tirupathi, India. Her research areas are Software Engineering, Reuse & Reengineering, Quality Assurance and Testing.

Sk. Munwar received the B.Tech degree in Information Technology from JNT University; He is currently working as Assistant Professor in the Department of Information Technology at Sree Vidyanikethan Engineering College, Tirupati, India since 2004. His current research interests include Computer Networks, Wireless Networks and Information Security and Algorithms. He is a member of ISTE.