Test Coverage Measurement and Analysis on the Basis of Software Traceability Approaches

Muhammad Shahid, Suhaimi Ibrahim, and Harihodin Selamat

***Abstract*—Test Coverage is an important step in software testing and a good indicator of software quality. It helps in evaluating the effectiveness of testing by providing data on different coverage items. It ensures that testing is effectively carried out without missing out all the key functional areas or the features of the software under development. We have analyzed several software traceability approaches to understand the existing test coverage framework. These approaches have been classified in different categories. Moreover, we have compared researchers work based on these approaches. Comparison has been made based on some criteria such as coverage types, data base support, software traceability support, regression testing support, coverage construction and coverage re-construction techniques. We are of the opinion, although much research effort has been put on how to get coverage information by either code based testing or requirement based testing, not much has been paid to measure the coverage by combining both to get integrated coverage.**

***Index Terms*—Test Coverage, Code Coverage, Software Traceability, Software Testing.**

I. Introduction

Software testing is an essential activity in software maintenance life cycle. It is a practice often used to determine and improve software quality. Testing activities include obtaining the test coverage. Coverage is the extent that a structure has been exercised as a percentage of the items being covered. If coverage is not 100%, then more tests may be designed to test those items that were missed and therefore, increase coverage [1]. Test coverage can help in monitoring the quality of testing, and assist in directing the test generators to create tests that cover areas that have not been tested before [2].

There are several test coverage analysis approaches that have been proposed from a very simple way until the latest ones that apply very complex techniques. This paper aims at evaluating several recent test coverage analysis approaches. The initial results obtained from this evaluation can be used to observe to what extent each approach has a capability to support test coverage analysis. Although much research attention has been given to Test Coverage measurement and analysis, but there is a need to cover all three granularity levels of test coverage items, i.e. fine- grain, medium-grain and course-grain in detail.

The remaining part of this paper is organized as follows. Section II describes the test coverage in software testing. It provides definition of test coverage and the usage of its analysis in different fields of software testing. It is followed by section III which offers the survey of recent test coverage approaches. Section IV provides evaluation of these approaches based on a set of criteria and finally section V concludes this paper.

Manuscript received August 25, 2001; revised September 13, 2011.This work was supported by the by the UTM-RU grant under the vote no. 01J58.

Muhammad Shahid is Ph.D. candidate at Universiti Teknologi Malaysia, Jalan Semarak, 54100, Kualalumpur, Malaysia (phone: +6 016 2242627; fax: +60 3 26930933; e-mail: smuhammad4@live.utm.my).

Suhaimi Ibrahim is with Universiti Teknologi Malaysia, Jalan Semarak, 54100, Kualalumpur, Malaysia (e-mail: suhaimiibrahim@utm.my).

Harihodin Selamat is with Universiti Teknologi Malaysia, Jalan Semarak, 54100, Kualalumpur, Malaysia (e-mail: harihodin@ic.utm.my).

II. Test Coverage

Coverage is a quality insurance metric which determines how thoroughly a test suite exercises a given program. The output of coverage measurement can be used in several ways to improve the verification process. First, the coverage tool can detect illegal events that occur and help find bugs in the design. Coverage measurement can also provide the user with information on the status of the verification process. It can help to find holes in the testing, i.e. areas that are not covered [2]. Test coverage also helps in Regression testing, test case prioritization, test suite augmentation and test suite minimization.

A. Test Coverage Items

Based on the research by different researchers [15], there are three granularity levels of test coverage items, i.e. fine-grain, medium-grain and course-grain.

Fine grain level consists of Statement, Basic Block, Branch, Condition, Condition / Decision coverage, Modified Condition MCDC, and LCSAJ (Linear Code Sequence And Jump) items. Method or Function, Class, Package and Design items are categorised in medium-grain while Requirements are considered as course-grain item.

B. Test Coverage Process

The Test/code coverage analysis process is generally divided into three tasks: code instrumentation, data gathering, and coverage analysis. Code instrumentation consists of inserting some additional code to measure coverage results. Instrumentation can be done at the source level in a separate pre-processing phase or at runtime by instrumenting byte code. Data gathering consists of storing coverage data collected during test runtime. Coverage data analysis consists of analysing the collected results and providing test strategy recommendations in order to reduce, to feed or to modify the relevant test suite.

III. Classification of Test Coverage Approaches

This section will present recent state-of-art approaches that have produced solution for test coverage measurement

and analysis. We have identified some significant approaches that can contribute to the further development and enhancement of test coverage measurement and analysis process.

SOFTWARE TRACEABILITY APPROACHES

The researchers [3] divide traceability approaches into seven categories.

- Information retrieval
- Rule based
- Event based
- Hypertext-based
- Feature model-based
- Value-based
- Scenario-based

A. Information Retrieval Approach (IRA)

This approach focuses on automating the generation of traceability link by similarity comparison between two types of artifacts [4]. The two basic IR models which commonly used in traceability generation are probabilistic and vector space models. Numerous variant models have also been applied including the popular model Latent Semantic Indexing which is based on vector space model. In each model, one type of particular artifacts treats as a query and another type of artifacts treats as a document being searched in term of the query. For example, source code treats as a query against requirements specification as a document being searched based on the query.

The general steps include (i) preprocessing, i.e. stop word removal and or stemming, (ii) analyzing and indexing of an incoming document collection, followed by constructing a representation of each document and then archiving them, (iii) analyzing and representing incoming queries, and using a matching or ranking algorithm to determine which document representations are similar to the query representation. The scope of tracing covers almost of artifacts including high-level and low level requirements, manual documents, design elements, test cases, and source code.

B. Rule-Based Approach (RB)

RB covers requirement statement documents, use case documents, and analysis object models as the objects of tracing. Spanoudakis et al. in [5] propose a method to automatically create traceability link using rules. They use two traceability rules, i.e. requirement-to-object-model traceability rule (RTOM rule) and inter-requirement traceability rule (IREQ rule). The rules are deployed into three specific documents types, i.e. (i) requirements statement documents (RSD), (ii) use case documents (UCD), and (iii) analysis object models (AOM). RTOM rules are used to trace the RSD and UCD to an AOM, while IREQ rules are used to trace between RSD and the UCD. The method assumes that all of document types are in XML-based format. The traceability rules are also represented in XML-based markup language. The method consists of four stages, i.e. (i) grammatical tagging of the artifacts, (ii) converting the tagged artifacts into XML representations (iii) generating traceability relations between artifacts, and (iv) generating traceability relations between different parts of the artifacts.

C. Event-Based Approach (EB)

In this approach, Traceability relationships are defined as publisher-subscriber relationships in which dependent artifacts must subscribe to the requirements on which they depend. When a requirement change, the dependent artifacts are notified and subsequently proper action can be taken [6]. The method involves three main components, i.e. (i) the requirement manager which is responsible for managing requirements and for publishing change event messages to the event server, (ii) the event server which responsible for establishing traceability by handling initial subscriptions placed by dependent entities, and also listening for event notifications from the requirement manager(s) and forwarding event messages to relevant subscribers, and (iii) the subscriber manager which responsible for listening on behalf of the subscribers that it manages for event notifications forwarded by the event server.

D. Hypertext-Based Approach (HB)

Maletic et al. in [7, 8] proposed an approach. This approach utilizes XML as the main tool for representing models and created links. The models and their links are converted into XML-based representation. Models are categorized into anchor model and target model. The links are established between anchor and target model with particular link types, i.e. causal, non-causal, and or navigational links. Once the model-to-model traceability links have been established, meta-differencing mechanism is used to indicate if some changes have been occurred in the models. The evolution is supported by a fine-grained versioning technique. The scope of tracing includes all types of artifacts.

E. Feature Model-Based Approach (FB)

Riebisch and Pashov in [9, 10] describe feature model-based method for requirement traceability. They utilize feature modeling which describes a requirements as an overview and models the variability of a product line. A feature model consists of a graph with features as nodes and feature relations as edges. If the number of features is very high, then the representation of features and their relations are displayed by tables. FB is applied for the definition of a product by a customer. Every feature describes a property of a product from the customer's point of view. There are three categories of features, i.e. (i) *functional features* (ii) *interface features*, and (iii) *parameter features.*

F. Value-Based Approach (VB)

VB consists of five processes, i.e. (i) *requirements definition*, that is identifying atomic requirements and assigning an identifier to each of them, (ii) *requirements prioritization*, that is estimating the value, risk, and effort of each requirement, (iii) *requirements packaging*, that is identifying clusters of requirements, (iv) *requirements linking*, that is establishing traceability links between requirements and other artifacts, and (v) *evaluation*, that is utilizing generated traces for certain purposes, e.g. to estimate the impact of change for particular requirements. VB combines a manual and semi-automated way in obtaining the traceability links and in performing a change in the software artifacts. G. Zemont in [11] proposed this

approach.

G. Scenario-Based Approach (SB)

Egyed et al. in [12-14] propose scenario-based approach. SB uses a hypothesized trace information that have to be manually entered. Then, it uses runtime information to creating trace links. Test case scenarios are executed on a running system and execution information is obtained using a monitoring tool. The information is then combined with the hypothesized trace information to form a footprint graph. This graph shows the relationship among artifacts in the system. The traceability links are created automatically, but the hypothesized trace information must be manually entered. In SB, traceability links can only be created once a running system is available.

Researcher's Work

Based on the above approaches, we are explaining here the work of different researchers related to test coverage measurement and analysis.

Faizah and Ibrahim, 2009 [15] proposed the approach that was specifically designed and implemented to analyze and construct the test coverage. Their framework suggests that Software Requirement Specification (SRS), Software Design Document (SDD) and Software Test Document (STD) are to be used as the sources to get the input for test coverage. These sources of an application are to be instrumented with additional codes as markers. Thus when a test script or a test case is run, echoed markers from the executed application can be collected and processed by a trace analyzer. This analysis will generate the traceability matrices data which is kept in the database for test coverage analysis. The analysis of the test coverage can be done by using the collected data kept in the database.

Sakamoto et al. 2010, [16] proposed a framework for consistent and flexible measurement of test coverage, called the Open Code Coverage Framework (OCCF), which supports multiple programming languages. OCCF extracts commonalities from multiple programming languages focusing on only small syntax differences in programming languages using an abstract syntax tree. It consists of three subsystems: the code insertion, code-execution and coverage-display subsystems. OCCF inserts the measurement code into the source code, and coverage is measured by executing the program. OCCF provides guidelines to support several test coverage criteria.

Kapfhammer et al. 2008, [17] presented a test coverage monitoring technique that tracks a program's definition and use of database entities during test suite execution. The instrumentation phase accepts an adequacy criterion, a test suite, and the program under test. They instrumented the program under test by placing probes at key locations within the Control Flow Graph (CFG). The execution of the instrumented program and test suite yields the database-aware coverage results. The coverage report records the definition and use of relational database entities in the context of both the method and the test case that perform the database interaction.

Kessis et al. 2005, [18] used JOnAS J2EE server as the case study to present test and coverage analysis for large scale applications. They addressed the code coverage, which corresponds to a white box approach where the internal mechanisms of the application under test are seen by the tester. They used Clover coverage analyzer for coverage measurement. The Clover analyzer tasks are fully automated via the Ant tool. The integration task of the coverage measure processes with the build project and test processes are relatively easy. First the JOnAS sources are instrumented. Then a binary version of the instrumented code is generated. Third, tests are run. During test runtime, coverage data are gathered and stored. Finally coverage reports are generated based on stored coverage data.

Lingampally et al., 2007 [19] introduced a Java byte code coverage analyzer which provides test coverage information at method, block, branch and predicate level. The overall system has four components as, i.e. Instrumentor, Database, Code Change Analyzer and Graphical User Interface (GUI). Code under test (CUT) is given to the instrumentation engine, i.e., the Instrumentor, which analyzes the byte code using BCEL APIs and inserts probes at appropriate locations depending on the type of coverage desired. The instrumented code is then run with a test case or a test suite. During a test-execution, when a probe is invoked, the relevant data is recorded in the database.

Lormans and Duersen [20] presented a method for generating requirement overage views. They studied to investigate to what extent relevant traceability links can be reverse engineered automatically from available documents and used for generating coverage views. They used Latent Semantic Indexing (LSI) for recovering the traceability links. Their focus is on reconstructing coverage views, i.e., views on the requirements set that can be used to monitor the progress in requirements development, design, and testing.

Genetic algorithms work on the basis of 'genes', which are created randomly. They are then subjected to some task. Genes demonstrating good performance are kept for next phases, while others are discarded. As far as testing is concerned, Genetic algorithm searches for optimal test parameter combinations that satisfy a predefined test criterion. This test criterion is represented through a "coverage function" that measures how many of the automatically generated optimization parameters satisfy the given test criterion. Genes that optimize the coverage function will survive while others will be discarded; the process is repeated with optimized genes being replicated and further random genes replacing of discarded genes. Ultimately one gene (or a small group of genes) will be left in the set and this would logically be the best fit for coverage function.

Rauf et al., 2010, [21] have proposed a genetic algorithm based technique for coverage analysis of GUI testing. The technique has been subjected to extensive testing. Three simple applications were selected to experiment with, which included Notepad, WordPad, and MS WORD. The results have shown enhanced coverage with an increase in the number of generations.

Ciyong Chen et al., 2009, [22] presented a new method called EPDG-GA which utilizes the Edge Partitions Dominator Graph (EPDG) and Genetic Algorithm (GA) for branch coverage measurement. First, a set of Critical Branches (CBs) are obtained by analyzing the EPDG of the tested program, while covering all the CBs implies covering all the branches of the Control Flow Graph (CFG). Then, the

fitness functions are instrumented in the right position by analyzing the Pre-Dominator Tree (PreDT), and two metrics are developed to prioritize the CBs. Coverage-Table is established to record the CBs information and keeps track of whether a branch is executed or not. GA is used to generate test data to cover CBs so as to cover all the branches.

Ricardo D. F. et al. 2010, [23] used UML state machine model to show visually the coverage achieved by a test suite. They presented a novel model coverage analysis tool (MoCAT) that analyses the coverage of a UML state machine model achieved by a given test suite (either manually or automatically generated) and represents it visually. This coverage is represented by colouring the elements in the model, so it is easier and faster to visualize the portions of the model that were not exercised by the test suite.

Matteo et al. 2009, [24] presented an innovative approach to structural coverage based on a virtualized, instrumented, execution environment. Their framework, Couverture, is able to measure structural coverage of object and source code without requiring any form of application instrumentation with a single execution of the cross-compiled application and test suites. The basic idea behind Couverture is to virtualize the approach commonly used to measure object code coverage: while traditional object coverage tools require a physical connection to the target hardware to measure coverage. Couverture uses a virtualized environment producing a rich execution trace containing the address of executed object instructions and the outcomes of conditional branches. Couverture is then able to determine actual object code coverage by processing several execution traces corresponding to the executions of different tests.

IV. Comparative Evaluation

This section provides comparative analysis based on few important criteria such as coverage types, data base support, software traceability support, regression testing support, change management and test case generation support.

A. Coverage Types(CT)

The effectiveness of coverage measurement and analysis depends upon which coverage type has been used by an approach. The common coverage types used by researchers include; Statement, Basic Block, Branch, Path, Conditions, Loop, Condition/Decision, Modified Condition/Decision (MCDC), Method or Function, Class, Package, Design and Requirement. Some other coverage types are Data Flow and Control Flow coverage, c-use and p-use coverage.

B. Data Base(DB) Support

An important aspect of a good approach is its open data base support. Databases are used to record code elements and test coverage information for latter retrieval for further exploration of the facts.

C. Software Traceability (ST) Support

Software traceability is used in order to link software artifacts (code, requirement, test case) in either ways; 'code traceability' (code to test case) and 'requirement traceability' (requirement to test case). Most approaches use code to test case to measure test coverage however the use of both is increasing to measure integrated test coverage.

D. Regression Testing (RT) Support

Regression testing is any type of software testing that seeks to uncover software errors after changes to the program (e.g. bug fixes or new functionality) have been made, by retesting the program. The intent of regression testing is to assure that a change, such as a bug fix, did not introduce new bugs [11]. Regression testing is done so as to execute maximum portion of changed code and achieve maximum coverage.

E. Coverage Construction Technique (CCT)

Coverage construction is another important criteria for evaluation. It depends upon whether coverage is constructed with or without running a test case.

F. Coverage Re-Construction Technique (CRT)

A software change (a change of method, class, package, design, test cases, requirements, etc.) will lead to changing

TABLE I. Comparative Evaluation

Researchers	CT	DB Support	ST Support	RT Support	CCT	CRT
Faizah et al. (2009)	Method, Class, Package, Requirement	Yes	Yes Both	No	With and without running a test case	Traceability matrix
Sakamoto et al.(2010)	Statement, Line, Condition, Decision	No	No	No	No	No
Kapfhammer et al.(2008)	Data Flow	Yes	No	Yes	After running a test case	Test re-run
Kessis et al.(2005)	Line, Condition	No	No	No	After running a test case	Test re-run
Lingampally et al.(2007)	Block, Branch, Method, Class,	Yes	Yes Code Traceability	Yes	With and without running a test case	Database coverage retrieval
Lormans et al. (2005)	Requirement	No	Yes Requirement Traceability	No	Without running a test case	Traceability links
Rauf et al. (2010)	Path	No	No	No	NA	NA
Ciyong Chen et al.(2009)	Branch	Yes	No	No	Without running a test case	Coverage table retrieval
Ricardo D.F. et al.(2010)	MC/DC	No	Yes Code Traceability	No	After running a test case	Test re-run
Matteo et al. (2009)	Code coverage	No	Yes Code Traceability	No	After running a test case	NA

NA – Not Available

code at end. That needs to re-construct coverage after each change. So coverage re-construction plays an important role in coverage analysis. It can be done by test re-run, traceability link presence calculation, and traceability matrix and database coverage percentage retrieval.

Based on the evaluation criteria above, the approaches have been evaluated. The evaluation results are shown in Table I. The table shows that there is no approach that satisfies all the evaluation criteria.

V. Conclusion

This paper aimed to provide general overview and comparison of the current test coverage measurement and analysis approaches. We categorized these approaches into different categories. Then, we have evaluated all these approaches based on some criteria namely, coverage types, data base support, software traceability support, regression testing support, coverage construction and coverage re-construction techniques (test re-run, traceability link presence calculation, and traceability matrix and database coverage percentage retrieval). Finally, we have summarized important factors based on our study.

References

[1] ISTQB, "International Software Testing Qualification Board" version 2.0, 2007, www.istqb.org

[2] Grinwald, R., Harel, E., Orgad, M., Ur S, Ziv, A., "User Defined Coverage – A tool Supported Methodology for Design Verification", IBM Research Lab, Haifa Dac, San Francisco, CA USA, 158-163, 1998

[3] Rochimah, S., Wan Kadir, W. M.N., Abdullah A.H, an Evaluation of Traceability Approaches to Support Software Evolution. International Conference on Software engineering Advances (ICSEA 2007), IEEE

[4] G . Antoniol, G. Canfora, G. Casazza, G. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation,", IEEE Transaction on Software Engineering, vol. 28, no. 10, October, 2002, pp. 970-83.

[5] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," *Journal of Systems and software, 2004, pp. 105-27.*

[6] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Trans. on Software Engineering*, vol. 29, no. 9, Sept, 2003, pp. 796-810.

[7] J. I. Maletic, E. Munson, A. Marcus, and T. Nguyen, "Using a Hypertext Model for Traceability Link Conformance Analysis," in *Proceedings of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03)*, 2003.

[8] J. I. Maletic, M. L. Collard, and B. Simoes, "An XML Based Approach to Support the Evolution of Modelto- Model Traceability Links," in *Proceedings of 4th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*, 2005.

[9] M. Riebisch, "Supporting evolutionary development by feature models and traceability links," in *Proceedings - 11th IEEE International Conference and Workshop on the Engineering of Computer- Based Systems*, 2004.

[10] I. Pashov and M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," in *Proc. of 11th IEEE Int'l Conf. and Workshop on the Engineering of Computer-Based Systems*, 2004.

[11] G. Zemont, "Towards Value-Based Requirements Traceability," PhD Thesis, Department of Computer Science, De Paul University, 2005.

[12] A. Egyed, "A Scenario-Driven Approach to Traceability," in *23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, 2001.

[13] A. Egyed and P. Grunbacher, "Automating Requirements Traceability: Beyond the Record & Replay Paradigm," in *17th IEEE International Conference on Automated Software Engineering (ASE'02)*, 2002.

[14] A. Egyed and P. Grunbacher, "Supporting Software Understanding with Automated Requirements Traceability," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, 2005.

[15] Faizah, Ibrahim, S., "A Software Traceability Approach to Support Test Coverage Analysis", UTM, Malaysia, 2009G.

[16] Kazunori Sakamoto, Hironori Washizaki, Yoshiaki Fukazawa, "Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages", In Procceeding of the 10th International Conference on Quality Software (QSIC), 2010, China

[17] Kapfhammer, G.M., Soffa, m.L. "Database-Aware Test Coverage Monitoring", Proceedings of the Ist India Software engineering conference ISEC'08, Hyderabad, India, ACM.77-86.

[18] M. Kessis, Y. Ledru, G. Vandome. "Experiences in Coverage Testing of a Java Middleware", in Proceedings SEM 2005, Lisbon, Portugal. ACM, pp. 3945, 2005.

[19] R. Lingampally, A. Gupta, P. Jalote. "A Multipurpose Code Coverage Tool for Java," In Proceedings of the 40th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, 261b, 2007.

[20] Lormans, M., Deursen, A.V. "Reconstructing requirements coverage views from design and test using traceability recovery via LSI", TEFSE, Long Beach, California, USA, ACM 2005.

[21] Abdul Rauf, Sajid Anwar, M. Arfan Jaffer, Arshad Ali Shahid, "Automated GUI Test Coverage Analysis using GA", Seventh International Conference on Information Technology, 2010

[22] Ciyong Chen, Xiaofeng Xu, Yan Chen , Xiaochao Li and Donghui Guo, "A new method of test data generation for branch coverage in software testing based on EPDG and Genetic Algorithm", 2009

[23] Ricardo D. F. Ferreira, J. P. F., Ana C. R. Paiva "Test Coverage Analysis of UML State Machines.", 2010

[24] Matteo Bordin, C. C., Tristan Gingold, "Couverture: an Innovative Open Framework for code coverage analysis of safety critical applications.", 2009.

[25] www.en.wikipedia.org/wiki/Regression_testing

Muhammad Shahid was born in Sahiwal, Pakistan in 1962, educated at the University of the Punjab, Lahore, Pakistan (B.Sc in Science and M.Sc in Physics). Currently he is persuing his Ph.D. in Software Engineering at UTM, Kuala Lumpur, Malaysia.

He is also Deputy Chief Manager in National engineering and Scientific Commission (NESCOM), Islamabad, Pakistan. He has published a few papers namely: An Evaluation of Test Coverage Tools in Software Testing (Sydney, Australia, IACSIT 2011), A Study on Test Coverage in Software Testing (Sydney, Australia, IACSIT 2011) and An Evaluation of Requirements Management and Traceability Tools (Amsterdam, The Netherlands, WAET 2011). His field of interest is test coverage measurement and analysis in software testing.

Mr. Shahid is a member of SDIWC since February, 2011.

Suhaimi Bin Ibrahim was born in Kelantan, Malaysia in 1957, educated at the Universiti Technologi Malaysia(M.Sc in Computer Science and Ph.D. in Computer Science).

He is also Associate Professor in the faculty of Advanced Informatics School, Universiti Teknologi Malaysia (UTM). He has published a few papers namely: An Evaluation of Test Coverage Tools in Software Testing (Sydney, Australia, IACSIT 2011), A Study on Test Coverage in Software Testing (Sydney, Australia, IACSIT 2011) and An Evaluation of Requirements Management and Traceability Tools (Amsterdam, The Netherlands, WAET 2011). His field of interest is test coverage measurement and analysis in software testing.

Harihodin bin Selamat was born in Sahiwal, Malaysia in 1962, educated at the University of Cranfield, United Kingdom (M.Sc in Computer Applications) and Ph.D. in Computer Science, University of Bradford, United Kingdom.

He is also Associate Professor in the faculty of Advanced Informatics School, Universiti Teknologi Malaysia (UTM). He has published a few papers namely: An Evaluation of Test Coverage Tools in Software Testing (Sydney, Australia, IACSIT 2011), A Study on Test Coverage in Software Testing (Sydney, Australia, IACSIT 2011) and An Evaluation of Requirements Management and Traceability Tools (Amsterdam, The Netherlands, WAET 2011). His field of interest is test coverage measurement and analysis in software testing. Dr. Harihodin is a member of SDIWC since February, 2011