Approaches to Regression Testing

J. Hartmann and D.J. Robson

Centre for Software Maintenance
Computer Science, University of Durham
Durham DH1 3LE, United Kingdom.

Abstract

Software maintenance involves changing programs as a result of errors, or a change in the user requirements. It is common knowledge that new errors resulting in undesirable side effects are easily introduced into a software system that is undergoing maintenance modifications. Regression testing is the name given to the process of retesting software after such modifications. This paper examines current testing methodologies and their relationship to regression testing. Included is a survey and critical assessment of current regression testing tools, with a discussion of a number of important aspects which these tools have omitted and that still require to be developed further. Proposals for a new regression testing tool incorporating all of these stated concepts is suggested.

Introduction

It is generally accepted that every computer program requires some form of testing, in order to identify any existing errors. This fact became apparent almost as soon as users began to write their first programs. Thus references to software testing may be traced back to as early as 1950. Miller's paper presents a citation to Turing indicating that 'testing is the empirical form of software quality assurance, while proving is the theoretical way'[1].

Whilst testing a program, one needs to identify two categories of errors, namely functional errors, caused by departures from the program requirements, and logical errors, introduced during the implementation of the program design.

Over the past twenty years, the need for software testing has steadily gained significance, in terms of both experience and economics. As computer systems and the associated software have rapidly developed in complexity, the time and resources devoted to testing have increased. It has been said that 'approximately 50% of the elapsed time and over 50% of the total cost are expended in testing a program or system being developed'[2]. Literally, millions of programs and systems have been tested since then with widely disparate results. Although practitioners have witnessed numerous calamities and failures resulting from inadequately tested software, it is clear that there have also been many successes and, that a systematic discipline and testing methodology is now slowly emerging in the software industry.

This paper will attempt to provide a general overview of both static analysis and dynamic testing techniques. Furthermore, there will be a detailed discussion of regression testing, including an accurate definition and description. The survey of selected regression testing tools attempts to provide both a historical background to the subject of testing in the maintenance phase, and highlight the advantages and problems encountered in the use of such tools. It concludes with a discussion of desirable features for a new regression testing tool, based on topics such as machine independence, hybrid testing, a retesting strategy for software modifications, and the possible use of test case languages or grammars for the automatic generation of both input test cases and their output verification.

General Testing Strategies

The section will describe both dynamic testing and static analysis strategies, as well as overviews of related techniques such as black-box and white-box testing, that are later discussed in conjunction with proposals for a new regression testing tool.

Static analysis and dynamic testing are two testing strategies that mutually complement each other in their approaches to software verification. The main difference between the two techniques is in the analysis of the software under test. When using static analysis, the structure of the code is analysed, but the code itself is not executed, whereas in dynamic testing a test plan is derived, test cases are executed, and the results evaluated[3]. Thus, the former method provides the tester with global information concerning the program structure, whilst dynamic testing investigates the run-time behaviour of the program. As static analysis is concerned with the structure of the code, it is particularly well suited to finding logical errors and displaying questionable coding practices. Dynamic testing, however, has the responsibility of verifying other aspects, such as the program function and the more specific, detailed issues of program logic.

Static testing techniques can be performed either manually or with the help of an automated tool. It includes features such as error and anomaly detection aids, and structural analysis, which each have a particular strategy and approach for structure and organisation. Their usage allows the following information to be obtained by static analysis:-

- syntactic error messages;
- number of occurrences of source statements by type;
- cross-references of global identifier usage;
- analysis of identifier usage for each statement;
- subroutines and functions called by each routine;
- uninitialised variables;
- variables set but not used;
- isolated code segments that cannot be stimulated by input test data;
- departures from the given coding standards, such as the format;

CH2615-3/88/0000/0368$01.00 © 1988 IEEE

- misuses of global variables, local variables, parameter lists.

In general, dynamic testing methods are preferable to static testing strategies, as they have been suggested to be one of the more productive and cost-effective techniques. Several advantages accrue from their usage. They may be utilised 'to evaluate and confirm changes, individually or collectively, to code, as well as to assess its impact on the software requirements'. Dynamic testing techniques include functional (black-box) testing, structural (white-box) testing, performance monitoring and interface testing.

Black-box testing, which is also known as data-driven, or input/output testing, is based on the concept that one may regard the software under test as a black box and does not need to be concerned with the internal structure and behaviour of the program. The only interest is in determining whether there are instances in which the program does not conform to its specifications. All required input test data is derived directly from the given specifications.

The other relevant testing strategy is known as white-box or logic-driven testing. This method allows one to observe the internal structure and logic of the software under test. With this technique, the appropriate input test data may be derived from an examination of both the program code and the specifications. In white-box testing the target program is annotated with probes at strategic points in its control flow, which are triggered during the execution of the software, causing an appropriate entry to be made in a specified file or database. These entries are later analysed to produce statistical measures of the achieved test coverage.

Testing in the Maintenance Phase

Regression testing can be classified as a preventive maintenance technique, which occurs in the midst of the change activity. Of course, the most preferred form of maintenance would be none at all, but even in the unlikely event that the software requires no corrections to be made to it, one might still need to undertake some adaptive and perfective maintenance changes at a later stage. However, 'preventive maintenance is more than just the elimination of the need for maintenance'[4]. The use of techniques, such as regression testing, enable both maintenance and integration (system) testing, to be more flexible, effective and efficient in the use of time and computational resources. Thus, it could improve the potential capabilities of a software product to be easily adaptable to any changes in the requirements without a loss in reliability, and still provide management with confidence in their products as they enter the last phase of the softwares' life cycle.

The term regression testing was first mentioned in a paper by Scherr, at the first formal conference devoted to software testing, held at the University of North Carolina in June 1972 [5]. The IEEE Standard Glossary of Software Engineering Terminology [6] states the following as its definition:-

'Regression Testing - Selective testing to verify that modifications have not caused unintended adverse side effects or to verify that a modified system still meets requirements.'

Regression testing involves the selection of an appropriate subset of the initial input test data, which was constructed during the development of the software and the minimisation of 'an objective function subject to a series of constraints. These constraints are built up, by relating each test run to the segments executed by the run'[7]. Furthermore, it allows the retention of all developed input test cases in a form which makes it possible for them to be reused and understood by someone other than the original test designer. Its major flaw, however, is that the current regression testing tools require the entire set of test cases to be rerun, when modifications are made to the relevant source code. In the case of large software systems, the cost of such re-executions would be extremely high in terms of time and resources.

There is wide agreement to the suggestion that many of the errors appearing in production software have not arisen from the original implementation, but have accidently been incorporated during the post-release modifications and alterations, producing unintended side effects. In order to combat such problems, the original implementation of the software product should include a thorough set of test cases, or procedures that exercise and verify all the functional aspects of the program, together with the capabilities of retaining and extending these test procedures during the software's life cycle[8]. Thus in subsequent modifications or alterations all, or a specific subset of the previous test procedures, may be rerun in order to verify that the changes had only the local effects intended.

However, under the present state of software testing technology, effective regression testing is seldom possible. The input test cases are usually in a variety of different file and data formats, and storage media. Furthermore, complicated procedures are needed to load and execute selected test cases and verify the different responses from the software under test. Generally, the test designer is the only person who can determine whether the results of the executing test cases are, in fact, correct.

Survey of Existing Regression Testing Tools

The following survey will discuss a selection of regression testing tools in order to illustrate the work that has been undertaken in this area of software testing, since the first of these tools was introduced in 1972. It will attempt to highlight some of the advantages and problems associated with each tool.

The period from 1972 until 1978 was a time when a number of new software testing tools were being introduced. Several large companies, notably IBM, General Electric and TRW, decided to investigate the uses of such tools for program testing. However, over the years, many of these research and development prototypes perished, due to inadequate maintenance support. The few that did survive the test of time generally had 'some limited self-application. An example of such a tool was the JAVS system, built to analyse JOVIAL/J3 programs and coded in itself'[9].

The first software testing tool, which was commercially available in 1972, was the Automatic Unit Test system (AUT) developed by IBM. Originally intended as a regression testing tool, it was mainly used in unit testing[10].

Panzl describes this first regression testing tool as having failed in gaining wide acceptance due to two important reasons:-

(i) The test language used in AUT, the Module Interface Language-Specific (MIL-S), was a 'low-level language reminiscent of assembler languages' and 'the test procedures tended to be lengthy';

(ii) The MIL-S test language 'contained no facilities for the modelling of I/O devices and files'.

Although AUT lacked a comprehensible and efficient test procedure language, it formed, together with a similar system known as TESTMANAGER, the first attempts at developing such black-box testing tools [11].

The next generation of regression testing tools were implemented in the form of software test drivers. In 1975, the General Electric Research and Development Centre developed such a driver, which became known as the Test Procedure Language (TPL/F) testing system. This TPL/F system extended the ideas used in IBM's AUT tester, as it incorporated a new higher level test procedure language based on FORTRAN.

Thus, test cases developed for a target program could be defined in a far more compact notation, and the use of the integrated macroprocessor allowed each test case represented by a single macro call.

Apart from enabling the test team to describe test cases or procedures in such a concise manner, a major disadvantage accrued from the fact that the system was restricted for use with FORTRAN programs. Furthermore, Heninger [12] suggested in an ACM review of the Panzl article that 'effective testing is based on comparing results derived by two independent means'. However in this case, the program execution and the programmer looking at its results are not independent, possibly resulting in a higher probability of error.

Since those early days, the interest in software testing tools, in particular regression testing tools, appeared to have dwindled. However in 1983 this interest was rekindled with the introduction of the AUTOTESTER system by Wang, which allowed the evaluation of the software under test using an interactive, distributed processing environment based on intelligent work-stations[13].

AUTOTESTER represented the start of a new generation of regression testing tools, that are designed to allow the viewing and reviewing of the external effects due to a series of interactive responses between a user and screen menus. Furthermore, the tool automatically compares the expected and actual interactive responses that result from the execution of the software under test and provides a mechanism for organising and structuring the test cases, thus allowing the regeneration of the user interactions during future regression tests and the monitoring of the test results.

More recently, two other regression testing tools have been introduced, notably the Hewlett-Packard Virtual Terminal/Scaffold Test Package Automation Tool and Test Package Standard[14], and the DEC/Test Manager[15].

All three tools have very similar problems. They are designed to be installed on hardware specific to each company, and able to operate only under the particular company's operating system. Furthermore, they do not adequately resolve the question of halting and recovery from a failure of the software under test. The sole advantage of the DEC tool is that the company's hardware products are widely available, and that it has been integrated into a loosely coupled set of general purpose programming tools[16]. Thus, other parts of the set, such as the Performance Coverage Analyser (PCA), can complement the DEC/Test Manager tool, improving the overall performance of the regression testing phase.

Methodologies Useful for Future Regression Testing

In the past, a number of interesting concepts have been developed, which are significant and could be adapted to provide further research opportunities with respect to the improvement and enhancement of existing regression testing tools.

The majority of current regression testing tools are based on the concept of black-box testing. However, in order to improve the test coverage and completeness of the software under test there exist no reasons as to why this should not be extended to include other testing strategies such as structural testing. A combination of both functional and structural testing, known as hybrid testing, would improve software reliability and could successfully be used to form the basis of a new testing strategy in future regression testing tools.

The concept of hybrid or gray-box testing was first discussed by Woodfield *et al.*[17] in 1983. Their aim was to combine both black-box and white-box testing strategies to produce a method which could verify software at a minimal cost. It was suggested that the success of the combination of any two methods relied upon their ease of use, and the way in which they complemented each other with respect to uncovering different types of errors in the software under test. By the development and use of an automated software tool known as the Pascal Automated Verification System (PAVS), they were able to perform structural analyses of the source code and obtain feedback from it in the form of a report, which could later be used to generate the required input test cases.

As a result of their research, they were convinced that this hybrid testing technique could produce acceptable levels of reliability at a cost lower than that associated with current techniques. Thus, incorporating such a hybrid methodology into a regression testing tool, could yield the answer to obtaining improved software reliability.

Apart from the ideas of combining both functional and structural testing strategies for improving the reliability and confidence in the software under test, one should also seek to develop new regression testing tools, which incorporate any of the latest ideas from either of the two testing methodologies.

This includes the path prefix testing technique, which could be described as a variation of the structural testing methodology and has an adaptive approach. The reason why its authors, Prather *et al.*[18], believe their strategy to be adaptive, is that the method utilises previous test input paths as a guide for the selection of its subsequent paths. The method also ensures a reduction in the number of branch coverage tests and advantages over existing strategies, in terms of its reduced computational requirements.

Path prefix testing could be termed an adaptive branch coverage methodology, that uses the previously traversed paths at each subsequent selection of a new input. For each available input path, the corresponding, shortest reversible prefix is determined, that 'constitutes the minimal initial portion of the input path to a decision node whose branches have not yet been fully covered'. The paper continues with a description of the path prefix algorithm, an example of its use, and the associated mathematical background. As the major advantage of this technique, Prather *et al.* have shown their method to have less computational requirements, because fewer decision nodes are considered in the interpretation of the path constraints on the input than in the case of the complete paths in the more test traditional methodologies.

In papers, published in 1977 and 1981 by Fischer [19,20], and in 1987 by Yau *et al.* [21], useful suggestions were presented for retesting methodologies. Their aim was to solve the problem of determining exactly which subset of the previously executed test cases was required to be rerun, in order to show that there had been no deterioration of reliability.

Current regression testing tools do not provide any mechanisms for automatically determining which subset of the stored test cases require to be rerun after code modifications have been made. Thus, to ensure the verification of the modified segments of code, the user is advised simply to rerun the entire set of input test cases, or intuitively/randomly select test cases which will exercise all main program features to provide a degree of confidence in the correct operation of the modified software. This approach, of course, can be very wasteful of both time and resources and may frequently be totally impractical for software systems with a large set of input test cases.

The two publications, by Fischer, describe various retest criteria, including those based on:-

•the reachability and connectivity of the modified code segment with respect to the other segments, which together constitute the program;
•the retesting of all code segments which branch to/from the modified code;
•the cost of each test case that requires to be rerun.

The solution to a suitable retesting strategy takes the form of a set covering problem. It uses the concept of 0-1 integer programming models to implement one of these criteria. These models consist of a minimising function:-

$$Z = c_1x_1 + c_2x_2 + ... + c_nx_n$$

subject to the following constraints:-

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n >= b_1$$
$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n >= b_2$$
$$\cdot$$
$$\cdot$$
$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n >= b_m$$

where **Z** is commonly referred to as the objective function, c_j is a cost element of the objective function, a_{ij} is an element of the constraint coefficient matrix, b_i is the lower bound on the constraint row **i**, and x_j is the variable for solution which can only take on the value 0 or 1.

In addition, algorithms and frameworks have been developed using this concept, which may prove to be useful in the automation of the retesting process as part of a new regression testing tool.

Another interesting concept, from the point of view of regression testing tools, is the automatic generation of the input test cases. If one could develop a new test case language, or adapt an established grammar to automatically create both the test case inputs, and verify the outputs from the software under test, then these would provide a powerful means of automatically establishing the initial baseline set of test cases.

The use of test case languages or grammars has already been proposed in publications by Duncan *et al.*[22], in which an attribute grammar generated test cases, either randomly or systematically. Furthermore, it drove not only the generation process, but also served as 'a concise documentation of the test plan'. Apart from illustrating the use of this attribute grammar, by means of a number of examples, the paper provides a detailed description of the capabilities of their developed test case generator, making comparisons of their technique with related methods, and discussing how it could be utilised in connection with various testing heuristics. As a conclusion to their work they believe that further research was required in discovering 'good testing heuristics based on the specifications and to find good ways of encoding these heuristics either in the test grammars themselves or as commands to the run time system'.

Future Research Directions

The aim of any future work here at Durham, is to extend the capabilities of current regression testing tools, by the development of a new software tool, which will incorporate a variety of new features and concepts including:-

a) An extension of the testing strategies involved by using hybrid techniques, that could improve both the reliability and confidence in the software product. The use of such techniques is based on the assumption that the source code of the target program is available. This gray-box testing concept could improve the test coverage of the software, and decrease the number of test cases which exercise the same code segments. The new structural testing techniques which are to employed, such as the path prefix strategy, may further reduce the computational requirements for this phase. The idea is to use the tool in conjunction with a test coverage analyser, whereby the regression testing tool initially supplies a given test case with the analyser monitoring the program execution, recording all relevant data. This information is subsequently fed back to the tool, allowing it to determine the next test case.

b) Automating the currently manual task of entering the baseline of initial test cases and generally improving the user interaction within the regression testing process. The user interface must be simplified further, so that the user can focus on testing of the software rather than on attempting to understand the proceedings of tool use.
Suggestions include:-

i) The development of a system such as the TESTDOC environment [23], in which the text of the functional specifications is produced and maintained by the computer, and functional strings are extracted automatically from the text and updated when they change within the functional specifications. Such ideas would prove ideal for a new regression testing tool, which could then extract the relevant functional data, in order to generate a test case based on this information and also verify the subsequent responses from the software under test. Furthermore, the TESTDOC system allows the user to control exactly what output he requires from the test tool. This aspect has fortunately been adequately addressed in recent developments, and is a great improvement over earlier tools in which voluminous amounts of output were produced.

ii) The development of new test case language directed towards describing interactive applications, so that the user could develop all required test cases, their parameters and the expected responses in a more general form. Furthermore, the associated command language would have to be designed for flexibility, to allow a large and ever increasing number of testing heuristics to be encoded.

c) The automation and subsequent reduction of the computational requirements and user time involved in the retest phase of a software product. Current tools depend on the user's intuition or simply sheer computational powers, to determine which test cases require to be rerun after modifications have been made to the source code. Thus, it is proposed that a new tool should analyse the modified code and provide the user with some guidance as to which test cases require to be executed again. Eventually, it may be possible to improve the process further, to a stage where minimal user intervention or assistance is needed.

d) Provisions for a more flexible and versatile regression testing tool, that runs under an operating system capable of being ported to a variety of different host machines. Current tools are restricted to use with specific hardware and operating systems.

e) The sensible recovery of the tool from the problems of hang-up of the test terminal and the actions taken when the testing process is suddenly halted and requires to be restarted at a particular point in the proceedings. Useful ideas in this matter were presented by the Wang Laboratories team who developed the AUTOTESTER system, but a satisfactory solution to the problem still needs to be found.

Acknowledgements

Jean Hartmann is sponsored by British Telecom Research Laboratories, Martlesham Heath, Ipswich, IP5 7RE, U.K. and would like to thank them for their support.

References

[1] Miller E.F. Jr., 'Program Testing Technology in the 1980s', The Oregon Report: Procs. of the Conf. on Computing in the 1980's (IEEE Press), pp. 72-9, 1979.

[2] Myers G.J., The Art of Software Testing, A Wiley-Interscience Publication, 1979.

[3] Fairley R.E., 'Tutorial: Static Analysis and Dynamic Testing of Computer Software', IEEE Computer, Vol. 11, No. 4, pp. 14-23, April 1978.

[4] Glass R.L., 'Software Maintenance Techniques', 12th Hawaii Int. Conf. Procs. System Sciences, Part I, pp.91-103, 1979.

[5] Hetzel W.C. (Ed.), Program Test Methods, Prentice-Hall, 1973.

[6] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1983 (IEEE Press), 1983.

[7] Ince D., Automatic Generation of Test Data, Open University Report, Dec. 1984.

[8] Panzl D.J., 'Automatic Software Test Drivers', IEEE Computer, Vol. 11, No. 4, pp. 44-50, April 1978.

[9] Miller E.F. Jr., 'Program Testing Technology in the 1980s ', The Oregon Report: Procs. of the Conf. on Computing in the 1980's (IEEE Press), pp. 72-9, 1979.

[10] Software Testing: Infotech State of the Art Report, Infotech State of the Art Report, Vol. 1, pp. 218-9, 1979.

[11] TESTMANAGER:Training Manual (TMR-008), Management Systems and Programming Ltd., March 1976.

[12] Heninger K., 'Comment to Panzl Article 'Automatic Software Test Drivers'', Computing Reviews Journal of the ACM, Vol. 20, p. 69, Feb. 1979.

[13] Leach D.M., Paige M.R. & Satko J.E., 'AUTOTESTER: A Testing Methodology for Interactive User Environments', 2nd Annual IEEE Phoenix Conf. Computers and Communications, pp. 143-7, March 1983.

[14] Fuget C.D. & Scott B.J., 'Tools for Automating Software Test Package Execution ', Hewlett-Packard Journal, Vol. 37, No. 3, pp. 24-8, March 1986.

[15] VAX DEC/Test Manager (Version 2.2), Software Product Description (26.68.04), April 1987.

[16] Connell C., 'DEC's VAXset Tools', DEC Professional, pp.72-7, March 1987.

[17] Woodfield S.N., Gibbs N.E. & Collofello J.S., 'Improved Software Reliability Through the Use of Functional and Structural Testing', 2nd Annual IEEE Phoenix Conf. Computers and Communications, pp. 154-7, March 1983.

[18] Prather R.E. & Meyers J.P. Jr. , 'The Path Prefix Software Testing Strategy', IEEE Trans. on Software Engineering, Vol. SE-13, No. 7, pp. 761-5, July 1987.

[19] Fischer K.F., 'A Test Case Selection Method for the Validation of Software Maintenance Modifications', IEEE COMPSAC 77 Int. Conf. Procs., pp.421-6, Nov. 1977.

[20] Fischer K.F., Raji F. & Chruscicki A., 'A Methodology for Re-Testing Modified Software', National Telecomms. Conf. Procs., pp. B6.3.1-5, Nov. 1981.

[21] Yau S.S. & Kishimoto Z., 'A Method for Revalidating Modified Programs in the Maintenance Phase', IEEE COMPSAC 87 Int. Conf. Procs., pp. 272-7, Oct. 1987.

[22] Duncan A.G. & Hutchison J.S. , 'Using Attributed Grammars to Test Designs and Implementations ', 5th Int. Conf. Procs. Software Engineering, pp. 170-7, March 1981.

[23] Ceriani M, Cicu A. & Maiocchi M., 'A Methodology for Accurate Software Test Specification and Auditing', Article from 'Computer Program Testing', pp. 301-25, June 1981.