Regression Test Selection for Black-box Dynamic Link Library Components

Jiang Zheng[1], Laurie Williams[1], Brian Robinson[2], Karen Smiley[2]
[1] *Department of Computer Science, North Carolina State University, Raleigh, NC 27695*
{jzheng4, lawilli3}@ncsu.edu
[2] *ABB Inc., US Corporate Research*
{brian.p.robinson, karen.smiley}@us.abb.com

Abstract

Software products are often configured with commercial-off-the-shelf (COTS) components. When new releases of these components are made available for integration and testing, source code is usually not provided. Various regression test selection processes have been developed and have been shown to be cost effective. However, the majority of these test selection techniques rely on access to source code for change identification. Based on our prior work, we are studying the solution to regression testing COTS-based applications that incorporate components of dynamic link library (DLL) files. We evolved the Integrated - Black-box Approach for Component Change Identification (I-BACCI) process that selects regression tests for applications based upon static binary code analysis to Version 4 to support DLL components. A feasibility case study was conducted at ABB on products written in C/C++ to determine the effectiveness of the I-BACCI process. The results of the case study indicate this process can reduce the required number of regression tests by as much as 100% if our analysis indicates the changes to the component are not called by the glue code of the application using the COTS component. Similar to other regression test selection techniques, when there are many changes in the new component I-BACCI suggests a retest-all regression test strategy.

1. Introduction

Companies increasingly incorporate a variety of commercial-off-the-shelf (COTS) components in their products. Upon receiving a new release of a COTS component, users of the component often conduct regression testing to determine if a new version of a component will cause problems with their existing software and/or hardware system. Regression testing involves selective re-testing of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements [4]. A variety of regression test selection (RTS) processes have been developed (for example, [1, 3, 10]) to reduce the number of tests that need to be executed without significant risk of excluding important failure-revealing test cases. However, most existing RTS processes rely on source code, and therefore are not suitable when source code is not available for analysis, such as when an application incorporates COTS components.

Due to the lack of information, the most straightforward RTS strategy for COTS-based applications would be to rerun all of the test cases for the application involving the glue code after the new COTS component(s) have been integrated. *Glue code* is application code that interfaces with the COTS components, integrating the component with the application. The retest-all strategy can be prohibitively expensive in both time and resources [3]. In our prior research, we have evolved an effective multi-step RTS process called *Integrated - Black-box Approach for Component Change Identification* (I-BACCI) for COTS-based applications by static binary change identification and the firewall analysis [12] RTS technique [16-18]. *Black-box testing*, also called *functional testing* or *behavioral testing*, is testing that ignores the internal mechanisms of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions [4]. Tool support for library (LIB) components has been developed [18]. We are continuing this research to *reduce the regression testing required for COTS-based applications that incorporate DLL components when components change and source code is not available*.

The rest of this paper is organized as follows. Section 2 outlines the background and related work. Section 3 describes the I-BACCI Version 4 process and its limitations. Section 4 presents the feasibility case study. Finally, Section 5 presents the conclusions and future work.

2. Background and Related Work

2.1. Testing of Software Components

Poor testability, due to the lack of access to the component's source code and other artifacts, is one of the challenges in user-oriented component testing [2, 11]. Generally, black-box tests can be run on COTS software even though users do not have access to the source code to analyze the internal implementation. Black-box test cases of COTS component functionality can be based upon the specification documentation provided by the vendor. Alternately, the behavior could be determined by studying the inputs and the related outputs of the component. When only binary code is available, binary reverse engineering is a technically-feasible approach for deriving information that can inform the RTS, e.g. call graphs describing the design structure of a component [9].

2.2. Firewall Analysis

Leung and White [7, 8, 14] developed firewall analysis for regression testing with integration test cases (tests that evaluate interactions among components [4]) in the presence of small changes in functionally-designed software. Firewall analysis restricts regression testing to potentially-affected system elements directly dependent upon changed system elements [14, 15]. Affected system elements include modified functions and data structures, and their calling functions. Our approach extends the traditional concept and scope of firewall analysis for use with binary code.

Module dependencies, control-flow dependencies, and data dependencies are considered in firewall analysis [14]. Dependencies are modeled as call graphs and a "firewall" is set up around the changed functions on the call graph. All modules inside the firewall are unit and integration tested, and are integration tested with all modules not enclosed by the firewall [14]. Test cases that need to be re-run over these modules are identified and/or new test cases to exercise new code or functionality are generated. The firewall concept has been utilized on object-oriented systems [5, 6, 12] and for regression testing of graphical user interfaces [13].

Firewall methods can only be guaranteed to select all modification-revealing tests and to be safe if all unit and integration tests initially used to test system components are reliable. *Modification-revealing test cases* are those test cases, when executed before and after the modification, for which the program will generate different output [10]. A *safe* RTS process guarantees that the subset of tests selected contains all test cases in the original test suite that can reveal faults based upon the modified program [1, 7, 10]. Tests are *reliable* if the correctness of modules exercised by those tests for the tested inputs implies correctness of those modules for all inputs [10]. Since test suites are typically not reliable in practice [15], the firewall technique may omit modification-revealing tests and/or may admit some non-modification-traversing tests. However, via empirical studies of industrial real-time systems, firewall was shown to be effective despite these theoretical limitations [15]. Firewall is the RTS technique embodied in the I-BACCI Version 4. Based upon these empirical study results for firewall, these theoretical limitations of firewall should not impair the effectiveness of the I-BACCI process in practice.

2.3. Legal Issues

Twenty-eight software license agreements[1] were gathered to investigate the legality of analyzing binary code of purchased COTS components. Relevant sentences in the license agreements were reviewed by lawyers of North Carolina State University (NCSU). Many of these license agreements of commercial components prohibit the users of components from reverse engineering, decompiling, disassembling, or otherwise attempting to discover the source code of the software, except to the extent that this restriction is expressly prohibited by law. Copyright law does not prohibit analysis on the code, only prohibits reproducing the components, making derivative works, or distributing copies of the products. As a result, the NCSU lawyers deemed that the approaches and algorithms used in the I-BACCI process are legal given the purpose of the analysis.

We have also consulted a professor of Software Engineering who is also a lawyer. The definition of "reverse engineering" he provided is: "to study or analyze (a device, as a microchip for computers) in order to learn details of design, construction, and operation, perhaps to produce a copy or an improved version." The professor deems that we are reverse engineering, and if a license indicates "no reverse engineering", then use of the tool could constitute a breach of contract. However, the interaction of patent law and mass market license terms, as it affects interoperability, is being actively debated within the legal profession[2]; furthermore, many software components may not have this license clause.

---

[1] The agreements we analyzed are listed at http://www4.ncsu.edu/~jzheng4/895/legalissues.htm

[2] e.g. Daniel Laster, http://law.bepress.com/expresso/eps/975

3. The I-BACCI Version 4 Process

The I-BACCI process is an integration of a binary code change identification process and a code-based RTS process. Our uniqueness is the combination of these two parts to identify and localize the changes, to reduce the regression test suite. The I-BACCI process and supporting tools have been generalized to Version 4 through the application of the process on both LIB and Dynamic Link Library (DLL) components written in C/C++. The I-BACCI Version 4 involves seven steps, as shown in Figure 1. The inputs to the I-BACCI process are shown in gray blocks. The first four steps are completed via the binary change identification process (in dash-dotted line frame), which produces a report on affected exported component functions. *Affected exported component functions* are functions within the COTS component that interface with the application, and are either changed or affected by other changed functions. The remaining three RTS steps are currently completed via firewall analysis (in dashed line frame) and ultimately produce the reduced test suite that needs to be rerun.

3.1. Process

The first step of the I-BACCI process is to decompose[2] the binary files of the component. Prior to distribution, component source code is compiled into binary code, such as `.lib`, `.dll`, or `.class` files. Information on the data structures, functions, and function calling relationships of the source code is stored in the binary files according to pre-defined formats, such as Common Object File Format (COFF) and Portable Executable (PE) format[3], so that an external system is able to find and call the functions in the corresponding code sections. Often the first sub-step can be accomplished by parsing tools available for the language/architecture. For example, COFF and PE binary files can be examined by the Microsoft COFF Binary File Dumper (DUMPBIN)[4].

The second step, filtering trivial information, is frequently necessary because the output from the first sub-step may contain information such as timestamps and file pointers, which are irrelevant to the change identification. Generally, the second step cannot be completed via existing tools. The Decomposer and Trivial Information Zapper (D-TIZ)[4] tool was created to perform the decomposition and remove trivial information. The output of Step 2 is the raw code section of each function/data, and function/data calling relationships for the new version of the component.

The main goal of **the third step** of the I-BACCI process is to identify true positive changes in the raw binary code of functions and data. The Trivial Identifier of Differences in BInary-analysis Text Zapper (TID-BITZ)[4] tool removes most of the false positives caused by trivial differences such as shifted addresses and register reallocations. Also, this step generates call graphs for the new version of the component. We created Call-graph Analyzer - Affected Function Identifier (CAAFI)[5] applications to represent and analyze the call graphs of components of both LIB and DLL types automatically in the I-BACCI Version 4. For LIB components, Step 2 is executed before Step 3. However, Step 3 must be executed before Step 2 for DLL components because only the names of exported component functions can be obtained in the binary code. The main goal of Step 2 and Step 3 is to facilitate comparisons and the identification of affected exported component functions.

In **the fourth step**, we identify changed and new component functions according to the results of prior steps, and then identify affected exported component functions by tracing along the call graphs within the component using directed graph theory algorithms. Analysis starts from each component function identified as changed, and that change is propagated along the call graphs from Step 3 until the exported functions are reached. The output of Step 4 is a list of all affected exported component functions. CAAFI identifies the affected component functions as well.

Using the source code of glue code functions, **the fifth step** is to generate function call graphs for glue code functions that call exported component functions. The call graphs generated from Step 3 and Step 5 can be integrated to learn how glue code functions are affected by changed and new component functions. The call graphs can be drawn using existing open source tools such as GraphViz[5].

Similar to Step 4, the affected glue code functions are identified in **the sixth step**. *Affected glue code functions* are functions within the glue code that directly call affected exported component functions and therefore need to be re-tested.

In **the seventh step**, the set of test cases which are mapped to the glue code functions they cover are used to select only test cases that cover the affected glue code functions, as identified by the steps above.

The I-BACCI process has the potential to reduce the set of regression test cases because it focuses on the affected glue code functions and ignores the unaffected areas in the application.

---

[3] We use the term *decomposing* to refer to breaking up the binary code down into constituent elements, such as code sections and relocation tables.

[4] MSDN Library - Visual Studio .NET 2003

[5] http://www4.ncsu.edu/~jzheng4/895/tools.htm

[6] An open source tool, http://www.graphviz.org/

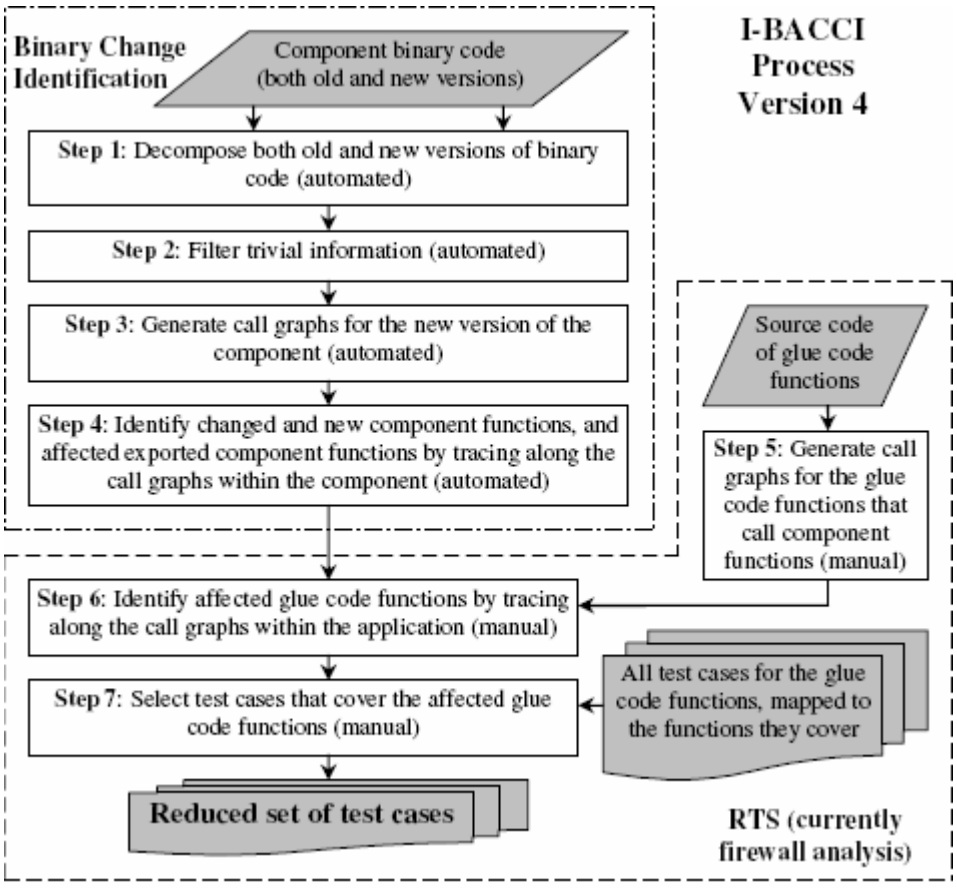

Figure 1: I-BACCI Version 4 regression test selection process

3.2. Limitations of the I-BACCI Process

The I-BACCI process shares an acknowledged technical limitation with all existing firewall methods: the potential for reporting false positives and false negatives when binary differences are due to factors other than changes in source code (e.g. build tools, environment, or target platform). Although such differences are potentially detectable from the binary file comparisons used in the I-BACCI process, the current method of analysis precludes identification of such differences.

The second limitation of the I-BACCI process is its potential for identifying false positives by conservatively assuming, in tracing the call graphs, that any uses of called functions with changed binaries will be affected by the change. However, an actual use of a changed function might never exercise the changed logic or data. With further development of the I-BACCI process, these unneeded tests may be eliminated from the regression suite.

As noted above, license agreement considerations may constrain the breadth of applicability of this tool and method.

Finally, the I-BACCI process requires (as input) test suites which are traceable to the glue code functions they cover, in order to perform RTS.

4. Feasibility Case Study for DLL Components

Two case studies had previously been conducted on ABB applications that incorporate LIB components using the prior I-BACCI versions [16-18]. These two case studies were re-analyzed with I-BACCI Version 4 and the results are identical to those presented in [18].

A new case study (henceforth called Case 3) was conducted with on a 757 thousand lines of code (KLOC) ABB application (henceforth called Application A) written in C/C++ using the I-BACCI Version 4 process. For Case 3, Application A uses a 3 KLOC internal ABB software component (henceforth called Component C) of a DLL file written in C. Four incremental releases of Component C were analyzed and compared (henceforth referred to as Release C1 through Release C4, respectively). Each Component C release contains a DLL file with size of about 110 kilobytes. We strived to maintain as much objectivity

as possible in the work. The first author was the analyzer and the third author was the verifier. The analyzer conducted the first six steps of the I-BACCI Version 4. The results of the identified changes for all comparisons and all call graphs for the components were preliminarily verified by the analyzer, using source code for the component to determine the accuracy of the analysis post hoc. Then, the verifier determined the numbers and percent reduction of the regression test cases needed, based on the list of all the affected glue code functions and the original test suite. The verifier also confirmed the efficacy of the RTS process by examining the failure records of the retest-all black-box testing that had been conducted.

4.1. Results

The results of applying the I-BACCI Version 4 on Case 3 are shown in Table 1. To establish a baseline of affected functions in the application, the interface between Application A's glue code functions and Component C was examined. Only two functions in Application A call four functions of Component C.

Table 1: Case 3 Results by the I-BACCI V4

Metrics	Comparisons		
	1 vs 2	2 vs 3	3 vs 4
Same linker?	No	Yes	Yes
Affected exported component functions	45	9	44
True positive ratio	100%	100%	100%
False positive ratio	60%	0%	0%
% of affected exported component functions	91.8%	18.4%	84.6%
Affected glue code functions	2	0	2
% of affected glue code functions	100%	0%	100%
Total test cases needed	31	0	31
% of test cases reduction	0%	100%	0%
Actual regression failures found	1	0	0
Regression failures detected by reduced test suite	1	0	0

The first analysis was conducted between Release C1 and Release C2. Of the 49 exported component functions in Release C2, 45 were changed. Both of the glue code functions in Application A that called Component C were affected. As a result, there was no regression test case reduction. The current tools identified all changes but had significant false positives when two versions were not built by the same linker. *The second analysis* comparing Release C2 and Release C3 correctly identified nine affected exported component functions, but no function in Application A called any affected functions in the components. Therefore, we achieved 100% regression test case reduction for this comparison. The result of *the third analysis* between Release C3 and Release C4 was similar to that of the first analysis, but no false positives were identified because the current tools worked well when comparing two releases built by the same linker. The verifier examined the failure records of retest-all black-box testing. One regression test failure was found in the first comparison. No false negatives were found in any analyses.

4.2. Case Study Limitations

A limitation of the case studies is that all of the applications and components used were software developed by ABB Inc. involving .lib and .dll library files. Additionally, the current tools work well only when the releases of components are built by the same linker. If two compared releases are built by different compilers or linkers, the current tools used in the I-BACCI process will yield a significant number of false positives. Also, we do not yet have data on the saving of regression testing time for the case studies.

5. Conclusions and Future Work

In this paper, we present a feasibility case study of the I-BACCI Version 4 process for regression test selection for applications that incorporate DLL components for which source code is not available. The I-BACCI process can reduce the required number of regression tests by as much as 100% if our analysis indicates the changes to the COTS component are not called by the glue code. Similar to other RTS techniques, when there are a large number of changes in the new component, I-BACCI suggests a retest-all regression testing strategy. The results have been verified by examining the failure records of retest-all black-box testing. Current tools identified all changes; no failures would have escaped the reduced test suites. However, current tools reported false positives when two versions were not built by the same linker.

We plan to pursue several directions in our future work. Besides expanding the I-BACCI process to adapt to more programming languages and more of the COTS types, such as Component Object Model (COM)[4] type, we plan to address the limitations of the I-BACCI process which are discussed in Section 3.2. First a theoretical analysis of the safeness [10] of the I-BACCI process will be conducted to complement our empirical studies. We will evaluate replacing the theoretically unsafe firewall analysis with other

existing safe code-based RTS processes (for example, [1, 3, 10]). Second, we will consider changes caused by factors other than source code (e.g. build tools, environment, and target platforms) to remove as many false negatives as possible. Additionally, extensive validation of both the tool support and RTS process will require more industrial case studies, data collection, and further RTS analysis. Based on the results of the case studies and research on limitations removal, we will continuously refine and integrate the supporting tools. Eventually, an open source end-to-end automation will be implemented to facilitate the efficiency and convenience of the whole process.

6. Acknowledgments

This research was supported by a research grant from ABB Corporate Research.

7. References

[1] Bible, J., Rothermel, G., and Rosenblum, D., "A Comparative Study of Course- and Fine-Grained Safe Regression Test-Selection Techniques," *ACM Transactions on Software Engineering and Methodology*, 10(2), 2001, pp. 149-183.

[2] Gao, J. Z., Tsao, H.-S. J., and Wu, Y., Testing and Quality Assurance for Component-Based Software. Boston: Artech House, 2003.

[3] Graves, T. L., Harrold, M. J., Kim, Y. M., Porter, A., and Rothermel, G., "An Empirical Study of Regression Test Selection Techniques," *ACM Transactions on Software Engineering and Methodology*, 10(2), 2001, pp. 184-208.

[4] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Standard 610.12*, 1990.

[5] Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., and Chen, C., "Change Impact Identification in Object-Oriented Software Maintenance," International Conference on Software Maintenance, Victoria, B.C., Canada, 1994, pp. 202-211.

[6] Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., and Chen, C., "Class Firewall, Test Order and Regression Testing of Object-Oriented Programs," *Journal of Object-Oriented Programming*, 8(2), 1995, pp. 51-65.

[7] Leung, H. and White, L., "A Study of Integration Testing and Software Regression at the Integration Level," International Conference on Software Maintenance, San Diego, 1990, pp. 290-301.

[8] Leung, H. and White, L., "Insights into Testing and Regression Testing Global Variables," *Journal of Software Maintenance*, 2(4), 1991, pp. 209-222.

[9] Memon, A. M., "A process and role-based taxonomy of techniques to make testable COTS components," in Testing Commercial-off-the-shelf Components and Systems, S. Beydeda and V. Gruhn, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 109-140.

[10] Rothermel, G. and Harrold, M., "Analyzing regression test selection techniques," *IEEE Trans. on Software Engineering*, 22(8), 1996, pp. 529-551.

[11] Weyuker, E. J., "Testing Component-Based Software: A Cautionary Tale," *IEEE Software*, 15(5), 1998, pp. 54-59.

[12] White, L. and Abdullah, K., "A Firewall Approach for the Regression Testing of Object-Oriented Software," in *Software Quality Week*, San Francisco, 1997.

[13] White, L., Almezen, H., and Sastry, S., "Firewall Regression Testing of GUI Sequences and Their Interactions," International Conference on Software Maintenance, Amsterdam, The Netherlands, 2003, pp. 398-409.

[14] White, L. and Leung, H., "A Firewall Concept for both Control-Flow and Data Flow in Regression Integration Testing," International Conference on Software Maintenance, Orlando, 1992, pp. 262-271.

[15] White, L. and Robinson, B., "Industrial Real-Time Regression Testing and Analysis Using Firewall," International Conference on Software Maintenance, Chicago, 2004, pp. 18-27.

[16] Zheng, J., Robinson, B., Williams, L., and Smiley, K., "A Lightweight Process for Change Identification and Regression Test Selection in Using COTS Components," 5th International Conference on COTS-Based Software Systems, Orlando, FL, USA, 2006, pp. 137-143.

[17] Zheng, J., Robinson, B., Williams, L., and Smiley, K., "An Initial Study of a Lightweight Process for Change Identification and Regression Test Selection When Source Code is Not Available," 16th IEEE International Symposium on Software Reliability Engineering, Chicago, IL, USA, 2005, pp. 225-234.

[18] Zheng, J., Robinson, B., Williams, L., and Smiley, K., "Applying Regression Test Selection for COTS-based Applications," 28th IEEE International Conference on Software Engineering (ICSE'06), Shanghai, P. R. China, May 2006, pp. 512-521.