Advanced Approach for Effective Verification of Component Based Software Systems

Irena Pavlova and Aleksandar Dimov

Dept. of Software Engineering, Faculty of Mathematics and Informatics,
Sofia University
5 James Bourchier blvd. 1164 Sofia, Bulgaria
{irena_pavlova, aldi}@ fmi.uni-sofia.bg

Abstract. The development of complex systems based on reusable components has many advantages such as lower costs and shortened development lifecycles. At the same time this innovative approach continues to place significant challenges towards integration and testing of such systems. The paper analyses the difficulties of components testing and proposes a combination of Built-In-Testing, Aspect Oriented Software Development, Test Driven Development and Test Governance to realize a method for effective verification in component based systems.

Keywords: component, component based development, built-in-testing, aspect oriented development, test driven development, test governance

1 Introduction

Increased requirements for quality, reduced costs and shortened "Time–to-Market" are the main aspects that determine the shift from traditional software development to modern advanced approaches. The process is focusing more and more on reusing preexisting software units and developing reusable entities and Component-based Software Engineering (CBSE) is becoming dominant [1].

The main principles of CBSE inherited from Object Oriented Development, such as encapsulation and information hiding, as well as the separate development of components and systems appeared to be quite a challenge when testing and verifying components during their integration into systems [2]. The lengthy and costly verification activities directly undermine the benefits of reusing the design and implementation efforts put into a component [3].

Although very promising, CBSE shows weaknesses in testing and verification aspects, which are also a consequence of lack of methodology, process and tools support. In this paper we analyze and propose to adapt and combine four advanced approaches in order to realize an effective method that will encompass the whole component and system development lifecycle and will decrease significantly the verification and integration efforts.

The rest of the paper is organized as follows. Section 2 introduces the basic principles of CBSE. Section 3 analyses the main challenges of components verification. Section 4 describes the advances and limitations of several technologies, in respect with component verification. In Section 5 a proposal is made on how the technologies could be combined in a really effective solution. In section 6 a conclusion is made and future research directions are outlined.

2 Component-Based Software Engineering Main Principles

2.1 Components and COTS

CBSE emerged as a result of the increased interest in reuse of pre-existing entities. Introduction of the notion of component [4] is the key innovation in this discipline. Here, we adopt the definition given in [5] and refer to the component as to a unit of composition with contractually specified interfaces and context dependencies only, that can be deployed independently and is subject to composition by third parties.

One of the main specifics of the component is its black box nature, i.e. the internal implementation is hidden. The only interaction point with the rest of the system is called interface. It should be very clearly specified, because it defines the services that components provide to and require from the deployment environment, as well as the configuration requirements for their execution [6]. When provided, components may have specific individual requirements that can be violated when composed and deployed with other components [7]. Hence, techniques are needed to guarantee that these requirements do not interfere with each other during components' integration.

2.2 The Asynchronous Development Process

In CBSE the development of components is separated from development of systems [8]. Two sub-processes exist: component development and development with components. There is a slight difference in the requirements and business ideas in this two cases and different approaches are needed accordingly.

During the development of component based systems, the emphasis is not on the implementation of the system designed, but on the reuse of pre-existing components [6]. Hence, the most critical part is namely the selection, evaluation (including testing) and integration. A number of open issues exist and one of them is related to efficient testing and verification of components in order to guarantee that they will meet the requirements specified, will be compliant with the system design and will work as specified in the target system.

Components are developed as reusable entities to be used in many products, many of them still not existing. For this reason they must be abstract and general enough but also sufficiently specific to be easily identified, understood, adapted, delivered, deployed, and replaced if needed. The above requirements raise some important questions towards components testability, as stated in [9], since testing is still the main approach used for technical evaluation of components to be integrated into systems.

3 Problems of Testing Component Based Systems

Practice has shown that standard testing techniques are inefficient for component based systems [10]. The main principles of CBSE, reusability, encapsulation and independent development are controversial to the basic elements determining testability, as defined in [11]: observability, traceability, controllability and understandability. Further analysis on component testability is made in [2].

As already mentioned, components are described to users, by component's interface specifications. Testing is equivalent to black-box testing where specifications are available but no design or code is. Unfortunately, the component specification does not contain sufficient information for the system engineer to test adequately and fully the available components. The limited options for describing comprehensively the component functionality affect strongly an important element of testability, namely understandability. Verification is additionally hampered by the independent development. Components are provided to be reused in different systems; hence they are developed more general and described at a higher level of abstraction, which sometimes leads to misunderstanding of their functionality.

In this context it is clear that due to the limited testability of components, new methods and techniques should be developed, in order to overcome the testing and verification problems and performance overheads.

4 Advanced Approaches for CBD

4.1 Built-in-Testing (BIT)

It is possible that a component possess facilities capable of generating test cases which can be accessed by the user or which the component can use to test itself and its own methods. These capabilities are called built-in testing (BIT) capabilities [12].

By developing a methodology for integrating BIT into COTS components, COMPONENT + FP5 project has widened the application of self-test techniques for component-based software [13]. In order to avoid the shortcomings of storing the test cases in the component, an architecture consisting of three types of components - BIT components, testers, and handlers is defined in [14]. BIT component interfaces are extended with one or more "testing interfaces" that provide information for the internal states of the component during testing. Testers are components that contain the test cases accordingly and invoke the built-in testing methods and handlers log and manage the testing information. BIT interfaces provide access to methods, builtinto the component, without revealing its internals i.e. the component encapsulation is preserved during testing and at the same time – testability is increased.

BIT is a Design-for-Testability (DFT) technique, because it encourages developing comprehensive tests synchronously with developing the functionality of the component. Here some important questions appear, such as separating testing concerns from functionality and ensuring more seamless integration of tests without influencing the normal component operation. These issues could be addressed by combination of BIT with Aspect Oriented techniques.

Unfortunately the BIT approach still lacks a clear methodology on how to analyze the testing requirements and formalize the testing considerations. That is why the adoption of approaches such as Test Driven Development (TDD) that fosters test development upfront could be very beneficial.

4.2 Aspect Oriented Software Development

Aspect-oriented software development (AOSD) seeks modularizations of software systems in order to isolate secondary or supporting functions from the main program's business logic. AOSD focuses on the identification, specification and representation of crosscutting concerns and their encapsulation into separate functional units, as well as their automated composition into a working system.

In the recent years, research activities are focused on combining aspect orientation with CBD. A good example is Aspect-Oriented Component Requirements Engineering (AOCRE) [15] that focuses on identifying and specifying the functional and non-functional requirements related to key "aspects" of a system each component provides or requires.

4.3 Test Driven Development

The last testing approaches have proved to be ineffective for complex systems. A new way to develop systems is required where testing is performed much earlier in the system lifecycle and the development evolves in an iterative and incremental manner [16]. Test Driven Development (TDD) emerged from Agile Development of the 80's to combat the above mentioned issues and completely turns traditional development around. Instead of writing functional code first and then testing it, the test code is written before the functional one. In other words - the most risky phase, testing, is moved from the end to the front of the life cycle.

4.4 Test Governance

To ensure that integration of independently developed pieces of software is successful, component producers and consumers should make an effort towards introduction of standardized notations, interfaces, data coding and semantics. Further, in order to ensure interoperability, it is necessary to put in place some organization to govern the interaction between the participating parties by some agreed rules. This is what Governance is conceived for. Here, we will adopt the definition given in [18]: Test Governance (TG) specifically concerns the establishment and enforcement of polices, procedure, notations and tools that are required to enable distributed testing of component integrations. Though [17] specifically targets Service Oriented Architecture (SOA), the governance principles are fully applicable in CBSE as well, as SOA evolved from component based approach.

5 An Approach for Effective Verification

Apart from the clear advantages, BIT has some limitations such as lack of formal methodology to cover the overall development process of components and systems, no clear roles and responsibilities, mixing testing with core functionality and others. We propose a new approach for realizing BIT, by advancing it trough other approaches that will lead to a more effective solution for verification of components.

From the viewpoint of aspect-oriented techniques for testing components it may be assumed that testing is a crosscutting concern, hence all testing functionality should be encapsulated in an aspect [18]. Further, following the identified testing aspects, the BIT methods support will be realized through aspect programming language as well. For example, BIT technology relies on the state based nature of components, as well as on the contract based interactions between the different components. BIT methods for putting the component under test (CUT) into a specific state before test and checking the state after execution might be realized through aspects. They have privileged access to the adapted implementation. The original source code is not modified and aspects can be easily removed.

The "test first approach" advocated by TDD will be very useful in the process of identification of testing aspects early in the development process and clearly separating the testing from functional concerns. In contrast to TDD, where testing is considered at a very low level i.e. unit testing, here it should be considered at component level. It is also important to note that both functional (Contract) and nonfunctional (Quality of Service) testing should be taken into account.

And last but not least, in order to ensure smooth implementation, the relevant polices, procedure, notations and tools should be in place to govern the overall process of testable component development and seamless integration into software systems and here is the place of the Test Governance.

6 Conclusions and Future Work

Although promising, reuse of independently developed software code hides some problems, especially in the area of component testing and verification. BIT technology promises to resolve these problems, by providing an architecture that makes it possible to reuse test code and develop self-testable software components. Apart from the clear advantages, BIT has some limitations, but we believe that they can be efficiently diminished by combining BIT with AOSD, TDD and TG. In order to validate the effectiveness of our approach, a case study will be realized: the proposed technologies will be combined and applied in the development of several components and their integration and testing into a system and the process will measured on the time, effort and the number of defects found basis. Further, a specification and formalization of the methodology and tool support is needed. They should cover the entire component and system lifecycle including technological, organizational, legal, and other aspects.

Acknowledgements. The work presented in this paper was partially supported by grants from the National Science Fund in Bulgaria under the MU-01-143 (ADEESS) project and the SISTER project, funded by the European Commission in FP7-SP4 Capacities.

References

1. I. Crnkovic, Component-Based Software Engineering-New Challenges in Software Development, Journal of Computing and Information Technology, Vol 11, No 3 2003
2. I. Pavlova, Testability of Component Based and Service Oriented Systems, Proceedings of 12 International Conference Automatics and Informatics, 2008

3. I. Pavlova, M. Akerholm, J. Fredriksson, Application of Built-In-Testing in Component-Based Embedded Systems, ISSTA 2006, p. 51-52, ISBN:1-59593-459-6
4. D. D'Souza, A. Wills, Objects, Components and Frameworks: The Catalysis Approach, Reading, MA: Addison-Wesley, 1998
5. C. Szyperski, D. Gruntz, S. Murer, Compnent Software – Beyond Object-Oriented Programming, Component Software, Addison-Wesley, 2nd ed., 1999.
6. Crnkovic, M. Larsson, Building Reliable Component Based Systems, Norwood, MA, USA: Artech House, Inc, ISBN 1-58053-327-2
7. D. Garlan, A. Robert, Architectural Mismatch or why it's Hard to Build Systems of Existing Parts; 17th ICSE p.179-185, 1995
8. M. Morisio, C.B. Seaman, COTS-Based Software Development: Processes and Open Issues, Journal of Systems and Software Volume 61, Pages: 189 – 189, 2002
9. S. Beydeda, V. Gruhn, State of the art in testing components, Proceedings of Third International Conference On Quality Software, ISBN: 0-7695-2015-4, 2007
10. St. De Panfilis, A. Berre, Open issues and concerns on Component Based Software Engineering, Deliverable of results of CBSEnet IST-2001-35485, 2005
11. J. Gao, Component Testability and Component Testing Challenge. Ph.D. Thesis, San Jose State University, USA, 2000
12. S. Beydeda, V. Gruhn, Merging components and testing tools: The Self-Testing Components (STECC) Strategy, 29th Euromicro Conf. , ISBN: 0-7695-1996-2, 2003
13. Y. Wang, G. King, A European COTS Architecture with BIT, Lecture Notes Computer Science, Springer Vol. 2425, p. 69-74, ISBN 978-3-540-44087-1, 2002
14. Y. Wang, G. King, H. Wickburg. A method for built-in tests in component-based software maintenance. European Conference on Software Maintenance and Reengineering, p. 186–189. 1999.
15. J. Grundy, Aspect-oriented Requirements Engineering for Component-based Software Systems, 4th IEEE International Symposium on Requirements Engineering, ISBN: 0-7695-0188-5, 1999
16. B. S. Mattu, R. Shankar, Test Driven Design Methodology for Component Based Systems, 1st Annual IEEE Systems Conference, 2007
17. A. Bertolino, A. Polini, SOA Test Governance: enabling service integration testing across organization and technology borders, IEEE International Conference on Software Testing, Verification and Validation Workshop, 2008
18. D. Sokenou, S. Herrmann. Aspects for Testing Aspects. Workshop on Testing Aspect-Oriented Programs, AOSD 2005, Chicago, USA, March 2005