



International Journal of Engineering, Business and Enterprise Applications (IJEBA)

www.iasir.net

Model-Based Testing of Enterprise System Architecture via Graphical User Interfaces

¹Sanjeev Dhawan, ²Nirmal Kumar, ³Divya Sethi

¹Assistant Professor, ²Research Scholar

^{1,2} Department of Computer Science and Engineering, University Institute of Engineering & Technology (U.I.E.T), Kurukshetra University, Kurukshetra- 136 119 (K.U.K), Haryana, INDIA.

E-mail (s): rsdhawan@rediffmail.com, nirmal.sirohi@gmail.com

³ GM Conferencing & VSAT Solutions, Enterprise Services, Bharti Airtel, Gurgaon, Haryana 122 001, INDIA.

E-mail: divya.sethi@airtel.in

Abstract: Software development and testing of Enterprise Resource Planning (ERP) systems need dedicated methods to tackle its special features. As manual testing is not capable to systematically test ERP systems due to the concerned complexity. So, an efficient testing approach should be required to automate such type of secured systems. Since the original business processes of enterprise systems are realized at the level of user interface (UI), therefore, it is quite obvious to use system level testing as one of the leading testing approaches. On the other side, the recent architectural shifts to service-based enterprise systems demand to apply black box testing techniques. But, Model-based testing is a process that enables a high degree of computerization for black box testing. The present research and practice usually does not address the proper functionality of UI. In this paper, an attempt has been made to systematically explain the state of the art and their practices in order to motivate further research activities in this area.

Keywords - Model-based testing, Enterprise systems, Service oriented architecture, UI testing, System-level testing.

I. Introduction

ERP Testing is a procedure that usually occurs before a company fully implements an ERP software package and the software goes live. It is done in order to identify certain situations (such as determining operational responsibility, training needs, and problem management procedures) which may arise after the implementation has been completed. Testing (or unit testing) is a very significant part of software development to deliver an efficient and cost-effective solution to a client. Primarily, it is required to develop a test script, which can test full end to end performance to get the desired results. In order to perform this, one should require specific business specialist that can provide crystal clear business scenarios. Thereafter, the business scenario can be converted into business steps. To carry such tasks, it is recommended to create a process flow diagram of software process cycle of that enterprise system. One thing should be remember that this procedure is subject to change throughout the development of the test script and unit testing. Forrester studies and surveys [5], [6] of more than 2200 IT decision makers across North-America and Europe show that 2/3 of the companies expect to be using SOA by the end of 2009 while 60% of those currently using it are expanding their usage. Leading firms now use Service oriented architecture (SOA) on more than 50% of their solution delivery projects. SAP, a leading provider of business software delivers SOA-enabled software, SOA methodology guidelines, and professional services [4]. The difficulties to be overcome are due to the heterogeneity, high distributive, dynamicity, loose coupling, and less granularity level of the service based systems. These difficult properties present enormous hindrance and cost challenges in the testing process. Therefore, a model-driven approach to SOA integration helps to address such challenges as it allows for a general solution, which can be applied to state-of-the-art tools and techniques for formal reasoning about SOA [10] and Model-based testing (MBT) [14].

II. SOA Testing

SOA enables IT departments to make the transition from an application-centric view of the world to a process-centric view. Today, IT departments have the liberty to combine business services from multiple applications to deliver true end-to-end support for business processes. An integration mechanism of SOA (usually Web-based services) enables coupling among various enterprise systems to improve or change their applications without making impact on other applications. However, SOA is being implemented both as green field (top down) and legacy modernization (bottom up). Thus, there is a clear lack of testing methodologies designed specifically for SOA applications. SOA testing requires testing of interfaces and services that might carry together diverse systems and platforms, along with other performance (latency) and security related aspects. One of the other

challenges to be tackled in SOA testing is the accessibility of the environment with the dependent fundamental services and/or applications. For instance, an SOA Implementation might bring together two or more autonomous internal application when composing a business process of an ERP based system. The availability of these internal applications becomes highly important during integration testing in parts as well as during end-to-end testing of the business process. Starting with a concise introduction to SOA, this paper outlines an approach to test SOA implementation in an efficient and reliable manner. The paper further describes the various testing categories, suggested test strategy, and brief overview to the available tools in the market that can be used to complement the overall testing approach for SOA and ERP based systems.

III. SOA Challenges

SOA brings new testing challenges that will require changes to your existing company test strategy if the key benefits of SOA are to be realized. Here some of the examples of the new SOA challenges [26]: -

- a) Many services will not have a user interface that will need a new breed of tools to assist with the testing.
- b) Infinite service consumers and users are possible.
- c) Interoperability.
- d) SOA can be boundless! Services may be used by applications yet to be developed or by consumers outside the enterprise.
- e) Loosely-coupled connection that allows unforeseen applications to take benefit of ever-expanding capabilities.
- f) SOA is driven by business processes that not only cross technologies but span organizations. Test teams will need a broader set of domain and technical knowledge.

Therefore, it can be assumed that MBT will have a much greater impact in the testing process than in traditional industrial development setups, where modeling is carried out rather intermittently. The lack of control over parts of the system implies that tests will be carried out not only in the development of a service-based application, but also regularly after deployment. Thus, to have automatic regression tests in place is a natural conclusion [17]. Moreover, the GUI testing in many cases is the only possibility of testing the functional and non-functional properties of a service-based system.

IV. Testing Stack of Enterprise SOA

After having discussed the specific challenges of SOA testing, a closer look will be taken on the particular testing activities, in order to see which of them is affected in what way. Utting and Legeard [13] discussed the commonly used layered testing approach for component-based systems (CBS). As the general idea of partitioning applications into logical units is somehow similar to the SOA approach of encapsulating related functional units in a service, the definition of SOA testing layers can be done analogous to the one for CBSs. Consequently, four distinct testing layers are illustrated in figure 1 [18]. In the following section, each layer is shortly introduced.

- A. *Unit Testing:* Unit testing is the best understood testing layer in research and practice. In contrast to all other testing layers, unit testing focuses on getting confidence in the functional correctness and hence in the correct implementation of the algorithms. As mentioned above, it deals with single software units in isolation. During unit testing the execution context of the software unit under test is mocked. Therefore, it can be carried out in SOA systems just like in any CBS implementation or any other software architecture, using all available tools and techniques.
- B. *Service Testing:* Also service testing for SOA is analogous to the testing of components in the CBS world to some extent. The general focus of service testing is less on the correct implementation of algorithms but on the integration of the functional units inside the (service) component and on the fulfilment of the contractual obligations of the component's interfaces. This is also conforming to the definition of testing layers [19], where it is argued that everything apart from unit testing is a form of integration testing.
- C. *Integration Testing:* As mentioned before, the loose coupling of service components is one of the distinguishing factors of SOA. In contrast to the CBS approach, integration testing cannot rely on homogeneous components with tightly connected interfaces. Instead, the adaptability and distribution of SOA demands additional considerations for integration testing. Especially the effects of message racing and its implications have to be considered during system development and should be tested thoroughly [18]. Message racing in this context refers to situations where messages are not received in the same order as they are sent.
- D. *System Testing:* In the SOA world, system testing can be defined similar to the classical definition for CBS. It comprises the fully integrated application, which usually uses its externally exposed interfaces. As the faultless interplay of the services can be assured on the integration testing level, in practice system testing is based on high-level usage scenarios and business requirements that have been defined by business analysts or customers. UI-based testing is therefore most appropriate to carry out the tests, as the system should be validated as a whole and only using access points that are available to the prospect user.

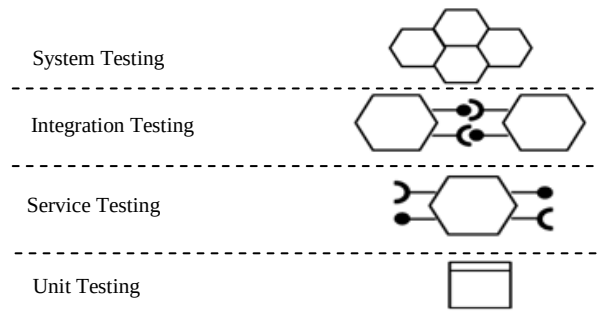


Figure 1: SOA Testing Layers [18]

V. GUI Test Automation

In most cases MBT has been used for testing through various kinds of APIs. However, this kind of approach is a limited one. An equally important area of application is system testing through a GUI. Software developers are usually unwilling to design system-level APIs solely for testing purposes. In addition, the general-purpose testing tools must be tailored in order to adapt and use such APIs in an effective way [27]. There are lots of general-purpose GUI testing tools available that can be simply utilised. However, GUI testing tools are usually not meeting the expectations among the test automation community. This scepticism is often a result of bad experiences in utilising capture/replay tools that capture mouse movements and key presses, and replay those in the regression tests. On the other side, capture or replay tools were the first generation test automation tools which suffered from many limitations. For example, defects, if any, were found during the capture phase. Replaying did not offer extra benefit. There were also high maintenance costs with such tools because the GUI is usually changed more frequently than the other system and changes to it create a need for change in the GUI test automation scripts. The times have distorted since the introduction of capture or replay tools. However, the key problem of scripted testing, the laborious maintenance, still remained. No real ease was found until the introduction of keywords and action words [16].

VI. Testing with Action Words and Keywords

Most recommended practice in the test design process is that the test designers concentrate on high-level concepts, i.e. business process modelling. This way they can pick the chains of events that are interesting for discovering potential defects. These concepts (high-level events) are called “action words”. Action words, of course, require concrete implementations in order to be useful in the test automation. These simpler implementing events are called keywords. Action words reflect the actions of users at a high level of abstraction. In the case of a mobile phone, for example, action words can be such as “add a new contact”, “send an SMS”, “answer a call”, “browse the calendar” etc. Keywords sequences map every action word to a sequence of key strokes, e.g. menu navigation, text inputting etc. Action words may involve several alternative keyword sequences that implement them. An example of a keyword in Symbian operating system could be *kwPressKey* which models a key press. This keyword could be used, for example, in an action word that starts the calendar application, *awStartCalendar*. The keywords may include parameters that specify their functionality. Pressing the key “5” in the keypad could be described as *kwPressKey* <5>. Sometimes the difference between action words and keywords is not clear. The most complicated or the most general keywords could be described as action words as well. Therefore the conceptual difference must be kept in mind when keywords and action words are defined. The level of abstraction and the purpose of use are the key factors that define which events are keywords and which are action words. Keywords are usually derived from a GUI while action words correspond to the operations of applications. [15]

VII. Three-Tier Test Model Architecture

In order to conduct MBT successfully in GUI test automation, a proper test model Architecture should be designed. Kervinen *et al.* [28] have developed three-tier test model architecture with the intention of performing a case study with it in Symbian OS environment. This architecture (*Figure 2*) is utilised in TEMA Tool and it is also the conceptual basis for the event capturing tool development. The architecture consists of three tiers that distinguish the utilised concepts in GUI testing:

- a) *Keywords* for navigating in the GUI.
- b) *Action words* to describe the high-level functional concepts.
- c) *Control words* to define the test control related matters.

As seen in figure 2, the lowest-level tier is the Keyword Tier which defines how to navigate in the GUI. In this tier the LTSs are called refinement machines. They describe the means of execution for GUI actions.

VIII. Brief View of State of the Practices

In the past, a significant effort has been put by the software industry to increase the efficiency of testing. Test automation in this context has been regarded as the most promising way. State of the practice of test automation for enterprise software is the automation of the test execution process, while the test design (the definition of

abstract test cases and their levels) is in general still a manual task. At SAP, the transformation from abstract test cases to executable test scripts usually follows the keyword-driven testing principles. Keyword-driven testing (or action-word testing) uses action keywords in the test cases, in addition to data. Each action keyword corresponds to a fragment of a test script (the adapter code), which allows the test execution tool to translate a sequence of keywords and data values into executable tests [13]. The introduced keywords are for example realized on top of SAP's eCATT test script language [20]. To allow the highest possible reuse, they are oriented on the structure of the enterprise system itself, which is organized in so-called transactions. Therefore, the current practice is to leverage the experience of the testers, who are asked to provide appropriate test data. SAP further provides a tool called Test Data Migration Server (TDMS), which is able to derive consistent reference data from existing systems. It is also quite common that reference test data is provided by customers or internal departments, as additional information to the requirement specification. On the basis of these data samples, testers are able to choose the correct input for each generated test case from that source. At SAP the test execution is controlled by the Test Workbench, where test plans (consisting of multiple test suites) are executed automatically and periodically in the case of regression tests. The results of the test runs are centrally reported, including different path coverage criteria based on source code, model elements or other requirements.

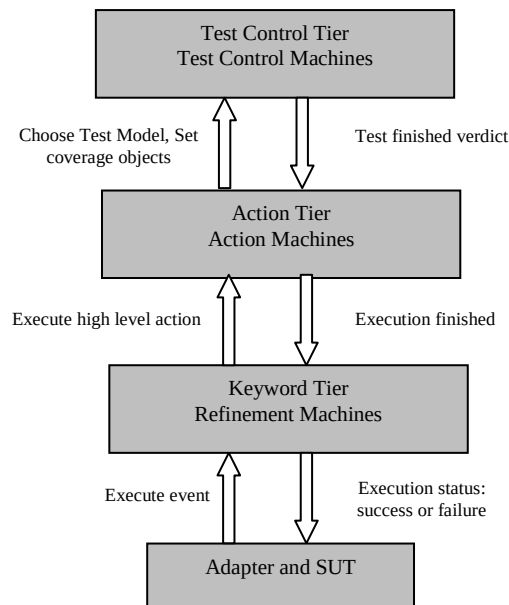


Figure 2: Three Tier Test Model Architecture

IX. Conclusions and Future Scopes

In this paper we tried to motivate and draw some guidelines of research in the area of system testing for enterprise systems, with a focus on GUI testing. The research will aim at improving the state of the art and state of the practice by using MBT techniques. This future work will leverage the existing work in GUI testing and MBT and try to find solutions that overcome the new challenges deriving from the new SOA paradigm. One of the key success elements in this endeavour will be the modelling part that will have to address the complex test data, existing models for business processes and UI composition, and dynamic aspects of UI at runtime. This research will contribute to more reliable and automatically testable SOA enterprise applications.

X. References

- [1] D. E. O'Leary, Enterprise Resource Planning systems, life cycle, electronic commerce and risk. Cambridge University Press, 2000.
- [2] G. Pike, "Supporting business innovation while reducing technology risk," SAP AG, Tech Rep., 2006.
- [3] S. Wiczorek, A. Stefanescu, and I. Schieferdecker, "Test data provision for ERP systems," in Proc. of Int. Conf. on Software Testing (ICST'08). IEEE Computer Society, 2008, pp. 396–403.
- [4] D. Woods and T. Mattern, Enterprise SOA - Designing IT for Business Innovation. O'Reilly 2006.
- [5] Forrester, "Enterprise and SMB software survey, North America and Europe, Q4 2008," Forrester Research, Business Data Service Survey, 2008.
- [6] R. Heffner, "Across all vertical industry groups, the majority of SOA users are expanding its use," Forrester Research, Research Report, May 2009.
- [7] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "Towards automated WSDL-based testing of web services," in Intern. Conf. on Service-oriented Computing (ICSOC'08), ser.
- [8] W.-T. Tsai, Y. Chen, R. A. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," in 29th Int. Computer Software and Applications Conference (COMPSAC'05). IEEE Computer Society, 2005.

- [9] J. Offutt and W. Xu, "Generating test cases for web services using data perturbation," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 5, pp. 1–10, 2004.
- [10] L. Baresi and E. Di Nitto, *Test and Analysis of Web Services*. Springer, 2007.
- [11] G. Canfora and M. D. Penta, "Service-oriented architectures testing: A survey," in *Software Engineering: International Summer Schools, ISSSE 2006-2008, Revised Tutorial Lectures*. Springer-Verlag, 2009, pp. 78–105.
- [12] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, "Challenges of testing web services and security in SOA implementations," in *Test and Analysis of Web Services*.
- [13] M. Utting and B. Legeard, *Practical model-based testing, a tools approach*. Morgan Kaufmann, 2007.
- [14] P. Baker, Z. R. Dai, J. J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams, *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2008.
- [15] A. Milanova, A. Rountev, and B. Ryder, "Parameterized object sensitivity for points-to analysis for Java," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 14, no. 1, pp. 1–41, 2005.
- [16] M. Greiler, H.-G. Gross, and K. A. Nasr, "Runtime integration and testing for highly dynamic service oriented ict solutions," in *Proc. of Testing: Academic & Industrial Conference - Practice and research techniques (TAICPART'09)*. IEEE Computer Society, 2009.
- [17] M. Acharya, A. Kulkarni, R. Kuppili, R. Mani, N. More, S. Narayanan, P. Patel, K. Schuelke, and S. Subramanian, "SOA in the real world - experiences," in *Service-Oriented Computing (ICSOC)*, vol. 3826, 2005, pp. 437–449.
- [18] S. Wiczorek and A. Stefanescu, "Service integration: A soft spot in the SOA testing stack," in *Proceedings of the 5th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR'09)*. To appear in IEEE Computer Society, 2009.
- [19] S. Ali, L. C. Briand, M. J.-U. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, "A based approach to integration testing based on UML models," *Information & Software*.
- [20] M. Helfen, M. Lauer, and H. M. Trauthwein, *Testing SAP Solutions*. SAP Press, 2007.
- [21] J. Jacky, M. Veanes, C. Campbell, and W. Schulte, *Modelbased Software Testing and Analysis with C#*. Cambridge University Press, 2008.
- [22] A. M. Memon, "An event-flow model of GUI-based applications for testing," *Softw. Test., Verif. Reliab.*, vol. 17, no. 3, pp. 137–157, 2007.
- [23] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, and J. Kazmeier, "Automation of GUI testing using a model driven approach," in *AST '06: Proceedings of the 2006 International workshop on Automation of software test*. New York, NY, USA: ACM, 2006, pp. 9–14.
- [24] I. Craggs, M. Sardis, and T. Heuillard, "AGEDIS case studies: Model-based testing in industry," in *proceedings of the 1st European Conference on Model Driven Software Engineering*. Imbus AG, 2003.
- [25] A. Paiva, J.C.P. Faria, N. Tillmann, and R. F. A. M. Vidal, "A model-to-implementation mapping tool for automated Model-based GUI testing," in *ICFEM'05, ser. Lecture Notes in Computer Science*, vol. 3785. Springer, 2005, pp. 450–464.
- [26] SOA Testing, Available online at <http://www.crestechglobal.com/sts-soa-testing.html>.
- [27] Mikko Satama, "Event Capturing Tool for Model-Based GUI Test Automation", Available online at: practise.cs.tut.fi/files/publications/TEMA/Satama_final.
- [28] Kervinen, A., Maunumaa, M., Katara, M., "Controlling Testing Using Three-Tier Model Architecture", In *Proc. of the 2nd International Workshop on Model-Based Testing, MBT'06*, Vienna, Austria, March 25-26, 2006.