Enhancing Test Driven Development with Model Based Testing and Performance Analysis

Sebastian Wieczorek,
Alin Stefanescu
SAP Research CEC Darmstadt
Bleichstr. 8, 64283 Darmstadt, Germany
sebastian.wieczorek@sap.com
alin.stefanescu@sap.com

Mathias Fritzsche
SAP Research CEC Belfast
Shore Road
BT370QB, Newtownabbey
United Kingdom
mathias.fritzsche@sap.com

Joachim Schnitter
SAP AG
Dietmar-Hopp-Allee 16,
69190 Walldorf,
Germany
j.schnitter@sap.com

Abstract

This paper suggests a holistic design and development method combining test-driven and model-driven development for SOA architectures. It uses test-driven development on component level and model-based testing on system level. Moreover, monitored performance parameters during test execution serve as input for a model-driven performance analysis of the business application, providing early indication of possible performance issues.

1. Introduction

Enterprise Resource Planning (ERP) software [15] supports business processes for whole companies. Such software systems are typically very large. SAP R/3 software for instance consists of more than 250 million lines of code [16]. ERP software integrates many organizational parts and functions into one logical software system, posing unique challenges to software development and testing [22].

Enterprise Service Oriented Architecture (Enterprise SOA) [23],[18] is regarded as the next evolutionary step to cope with the software complexity of ERP systems where monolithic approaches are not applicable anymore. In Enterprise SOA, an application platform consists of independent business components exhibiting enterprise services that can be composed individually to form customized business processes. SAP recently presented a new on-demand solution named *SAP Business ByDesign*[1] which is fully implemented on the principles of Enterprise SOA. Since the quality of the delivered software platform is crucial for a successful application to volume business, intensive quality assurance methods comprising several types of testing like unit, component, integration and performance are applied at SAP during the software development lifecycle.

As observed by [5], the increased business process agility promoted by SOA demands a test-driven approach to develop and adhere to governance standards and hence to provide high-quality services. SAP therefore provides tools for SOA governance in its technological platform to support reusability and business alignment of SOA services developed by customers.

According to SOA principles, complex application requirements are fulfilled by combining services into composite applications. This integration takes place on a higher level of abstraction than system development requiring specialized testing techniques. Test criteria like code coverage are not sufficient for service integration testing, as they do not take global communication into account but focus on local correctness. Therefore they are enhanced with more suitable criteria at higher levels of abstraction like state or transition coverage for the communication protocol between service providers and consumers [2],[13].

Model-based testing (MBT) [20] is a form of black-box testing that uses structural and behavioral models, described for instance in UML diagrams, to automatically generate test cases, thus automating the test case design process. MBT is gaining its momentum together with the model driven software development (MDD) methodology [2]. MBT test case generators aim to cover model related features, e.g. all states in an UML state machine, or data features, e.g. all boundary values. Therefore they are highly suitable for integration testing of SOA components [21].

Apart from functional correctness, software complexity also requires early identification of possible performance issues in order to avoid

[1] *http://www.sap.com/solutions/sme/businessbydesign*

significant additional refactoring effort. To enable early performance feedback a Model-Driven Performance Engineering (MDPE) methodology has been proposed in [8] and [9].

Agile development practices like Scrum [19], Test-Driven Development (TDD) [1], and Extreme Programming [6] are more and more regarded as key practices of software engineering for systems of extreme size and complexity. Waterfall-like development process models bear immense risks, as no subsequent process phase can correct all errors from the phases executed before. Iterative development models with close feedback loops help mitigate these risks. Having developers and stakeholders plan and review development efforts frequently creates transparency and allows for correcting implementation mistakes early in the process. Since 2004, SAP has been using Scrum and TDD was recently introduced to SAP´s technology development area. The about 80 Scrum projects finished before end of 2007 showed significantly better results compared to those using processes inspired by the waterfall and V models. First pilots for TDD delivered a much smaller defect rate, more comprehensible and thus more maintainable code.

The key contribution of this paper is the introduction of a holistic design and development method incorporating the advantages of agile and model-driven approaches. It further aims to combine the enforcement of functional correctness with the novel MDPE approach throughout the development process. The core idea is to use TDD for the business component development while MBT supports the inter-component service integration. Furthermore, performance parameters derived by the test executions are utilized to drive the MDPE performance analyzer. Figure 1 gives an overview of the envisaged process whose details will be discussed in the paper.

The paper is structured as follows. Our approach starts with the modeling of the targeted business process in Section 2 and an early analysis of the estimated performance in Section 3. It then continues with the implementation of additional service components, using TDD and model based performance monitoring in Section 4, and concludes with MBT generated integration test in Section 5, that also provide information for an automatic performance evaluation.

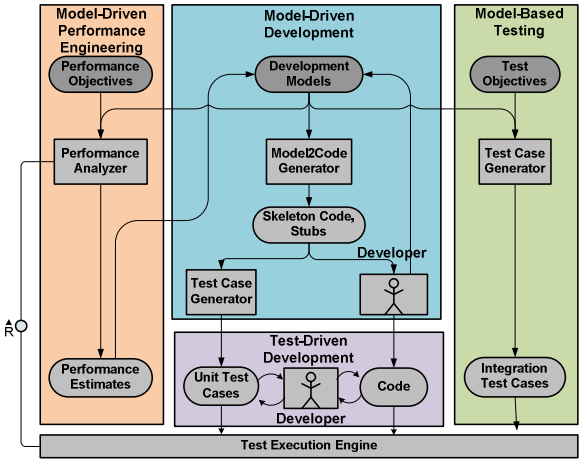

Figure 1: An overview of the proposed process

2. Modeling Process

As in every customer oriented scenario, the development process should start with the definition of user and market requirements. For Enterprise SOA applications especially functional requirements describe business processes which have to be supported. This top-down approach of software development, starting with a high level description of needs, can be combined easily with the concepts of MDD (see Figure 1), where general specifications of a system are stepwise refined by adding relevant domain specific information.

In the platform independent modeling (PIM) phase of MDD, the requirements are used to create a structural model identifying and connecting the business components, and a behavioral model of the business process flow. The business components are the building blocks of Enterprise SOA. They are defined as “modular, context-independent, reusable pieces of software that expose their functionalities as services” [18]. The inter-component communication in Enterprise SOA is message-based.

As illustrated in Figure 1 the proposed *Development Models* are used as input for a *Model2Code Generator* enabling automated generation of *Skeleton Code* and *Stubs*. The generated *Code* as well as the *Development Models* are refined by *Developers*. The different Development Models in use are described below.

The structural model has to identify existing business components that are already offering interfaces to support the envisioned process as well as those, which were not yet implemented in the Enterprise SOA platform. Interfaces defining the services of each component have to be part of the model as well as their associated interaction channels.

Amongst others, UML 2 component or composite structure diagrams can be used to describe this information in a transparent manner. The SAP modeling process however utilizes proprietary Process Component Interaction Models supported by the ARIS platform.

The behavioral model should supplement the structural model by describing the message choreography of the component interactions. It defines the communication protocol by restricting the message exchange depending on the overall state of the communication. An example of a standard regarding service choreographies is WS-CDL [24].

The message choreography modeling can be carried out in various ways [7]. For example UML sequence diagrams could be used. Also state machines, representing the protocol-compliant communication are suitable. The trigger of a state transition in such a model is the message which is sent from one component to the other while the states are aggregated states of the participating components. One advantage of such state machine representation is that the local behavior of components may be also derived [25]. On the contrary the advantage of activity diagrams is the directly visible link between messages, their sender and receiver. As each of the two model types can be transformed into the other it is up to personal taste which one to use.

For SAP the modeling of constraints on the communication of services components has been identified as a vital prerequisite to increase software quality in various fields, e.g. by providing the possibility of early performance evaluation, means to measure test coverage on the global process modeling level and even opening up the opportunity for automatic test generation. All the mentioned benefits will become clear in the following sections, while others like having a formal guidance for the software developers during implementation are out of focus for this paper.

3. Performance Estimation

In parallel to the MDD process, performance analysis can be carried out (see Figure 1). This is important because many performance relevant issues like inacceptable end-to-end processing times can be predicted and investigated at this early stage. Being able to identify such issues enables refactoring before critical architectural decisions are manifesting themselves in the implementation. Late identification of performance issues increases the effort and cost for fixing while possibly decreasing the software quality due to workarounds.

Performance objectives (see Figure 1), as well as requirements and modification constraints [9] are usually assigned to global use case scenarios at the application level and then broken down recursively. Performance predictions on the other side are derived bottom up. While reused components have known parameters, estimations have to be made for the new components. These have to be annotated to the previously defined models. In order to check the conformance of global performance objectives and local performance parameters *Performance Analysis* (see Figure 1), such as simulations of models annotated with performance parameters can help to predict future bottlenecks and performance issues. As a consequence, many problems that usually cause late code refactoring can already be spotted based on *Development Models*, and therefore support is offered to the *Developers* based on the models they understand.

4. Test Driven Development

At this point in the process, models have been produced that describe both structural and behavioral aspects of the envisioned process components and their interaction. Further it has been verified that they are compliant with the performance requirements.

Model-driven development would now lead to the next phase where the local behavior is further refined with the ultimate goal to automatically derive code. Such concrete modeling however often demands modeling languages that are equally powerful as the targeted programming language itself. Furthermore model debugging and verification is not state of the practice yet.

Test driven development offers a more promising approach at that level of abstraction. After automatically generating code stubs from the structural models, developers are starting to create unit tests for the functions they are going to implement (see Figure 1). Afterwards they are able to validate their own code automatically by running these tests. After test success refactoring takes place to increase code readability, followed by another round of testing to make sure the changes did not affect the behavior. These process steps are repeated recursively for each bit of added functionality.

TDD breaks with a widespread practice, which is to develop a significant amount of the desired functionality and only then perform tests for the first time. In complex systems errors are not only hard to detect and localize, but often require corrective actions that may affect many other parts of the system which are known to work properly. Test-driven development is a preventive method that provides tests and testbeds before a particular system part or function is

programmed. Software developers create unit test cases before even the first line of product code is written, and no line of code is written before a suitable test case exists. If this practice is strictly applied throughout programming the result is not only a system with a significantly reduced defect rate but also a complete unit test suite with code coverage close to 100 percent. This test suite helps to prevent introduction of new defects in working code when other defects are fixed, the system is extended, or system components are replaced.

As mentioned before, TDD provides code with measurably higher quality leading to decreased development and maintenance costs. Additionally automatic execution of unit tests can be used to monitor and compare the performance estimates, made for the components. Thus the performance annotated design models are updated with the measured performance parameters. Continuous recalculation and simulation for performance evaluation can again indicate possible issues much earlier and more effective.

5. Integration Testing

After development of the software components, integration testing is carried out. It has the aim to show that the application as a whole behaves correctly. Especially for applications whose components are loosely coupled, as it clearly is the case for SOA, tests of the communication and interaction are as vital as the functional correctness of the communicating parts.

To determine the success of integration testing, specific coverage criteria have to be applied. Local test coverage criteria are unfortunately not sufficient in determining whether two components are able to operate with each other under the agreed circumstances. Only the application of a global test concept can provide that.

For example, a global behavioral model incorporating all communicating components as parts of one application can be used to evaluate the integration test coverage. Such a model however easily exceeds the border of manageable complexity because it incorporates much detail that is not related to the correct communication. Integration testing should therefore focus on model content that is directly connected to the communication protocol and abstracts from the inner behavior of the components.

MBT approaches are able to effectively support automatic test generation for component interaction models as well as to execute and evaluate their success. Nevertheless manual work cannot be eliminated totally making it necessary to restrict the test case generation to a meaningful minimum.

As illustrated in Figure 1 after investigating integration test objectives [21] state of the art MBT test case generation can be used [20]. By applying MBT in the integration phase, not only the effort for test case generation can be decreased, but also test coverage and overall test effort can be controlled in an easy and transparent way.

6. Conclusion

In this paper we presented some general information on the implementation of applications based on service oriented architectures. We further described a holistic development process for such SOA applications that especially addresses modeling, testing and performance predictions. By using different concepts like MDD, MBT, MDPE and TDD we showed at which stage of development they can be most effective and how they can be combined. It has to be mention though that at the first sight there seems to be a philosophical mismatch between agile and model-driven approaches since the former wants to avoid heavy and complex formal specification during the development, whereas the latter essentially needs a precise and formal system specification to derive code and test suites.

Our aim was to show, that the approaches can form a symbiosis by joining their advantages in a global development process. First it has been described how the shortcoming of MBT in fine grained local modeling can be balanced by applying TDD, second how global MDD artifacts can not only be used for code generation but also for model-based integration testing. Third the application of MDPE along the development process has been promoted by showing its possible value.

The following related work also tries to combine agile and model-driven approaches. [20] presents some reconciliation of the two approaches where MBT is used to derive the unit tests as well as acceptance tests for TDD. In contrast, our approach proposes MBT for integration testing. [9] shows another possibility of using a model-driven approach in a more agile way for GUI testing based on use-cases while our scope is the application layer. [17] looks at the combination of approaches from a different perspective, by trying to bring in agile methods into the test modeling for MBT in the lines of agile modeling [1].

For our future work we anticipate measuring the effectiveness of this combination of methods in terms of costs and large scale deployment. To realize a smooth integration of the modules in Figure 1, a mature testing environment supporting tests at different levels like unit, integration and system tests is needed.

SAP possesses such a rich tooling integrated framework based on eCATT [11].

Acknowledgements: This work was partially supported by the EU-funded project MODELPLEX [14].

7. References

[1] S.W. Ambler. *Effective practices for Extreme Programming and the Unified Process*, Wiley&Sons, 2002. See also *http://www.agilemodeling.com*

[2] P. Ammann, J. Offutt, W. Xu, Coverage Criteria for State Based Specifications. In Formal Methods and Testing 2008, LNCS 4949, Springer, 2008, pp. 118-156.

[3] P. Baker, Z.R. Dai, J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams. *Model-driven testing – using the UML testing profile.* Springer, 2007.

[4] K. Beck, *Test-driven development by example*, Adisson-Wesley Professional, 2003.

[5] C. Benedetto, SOA and integration testing: The end-to-end view, *SOA World Magazine* 6(8), 2006. Online at: *http://webservices.sys-con.com/read/275057.htm*

[6] A. Cockburn, *Agile Software Development*. Addison-Wesley Professional, 2001.

[7] G. Decker and M. von Riegen, Scenarios and techniques for choreography design. In Proc. of *Business Information Systems (BIS'07)*, LNCS 4439, Springer, 2007, pp. 121-132.

[8] M. Fritzsche, W. Gilani, C. Fritzsche, I. Spence, P. Kilpatrick, and J. Brown, Model driven performance engineering for composite applications on top of Enterprise SOA. In Proc. of *ECMDA'08*, LNCS 5095, Springer, 2008.

[9] M. Fritzsche, W. Gilani, I. Spence, T.J. Brown, P. Kilpatrick, R. Bashroush, Towards performance related decision support for model driven engineering of Enterprise SOA applications. In Proc. of *15th ECBS'08*, IEEE, pp 57-65.

[10] M. Fritzsche, J. Johannes, S. Zschaler, A. Zherebtsov, A. Terekhov, Application of tracing techniques in model-driven performance engineering. In Proc. of 4th *ECMDA-FA workshop on traceability,* 2008.

[11] M. Helfen, M. Lauer, and H. M. Trautwein, *Testing SAP solutions*, SAP Press, 2007.

[12] M. Katara and A. Kervinen, Making model-based testing more agile: a use case driven approach, Proc. of Haifa Verification Conference, LNCS 4383, Springer, 2007, pp. 219-234.

[13] R. Lai, *A survey of communication protocol testing*, Journal of Systems and Software 62(1), 2002, pp. 21-46.

[14] MODELPLEX, MODELling solution for comPLEX software systems. 6th Framework Programme EU Project. Reference: IST 34081 (IP). http://www.modelplex.org.

[15] D.E. O'Leary, *Enterprise resource planning systems-systems, life cycle, electronic commerce and risks.* Cambridge University Press, 2000.

[16] G. Pike, Supporting business innovation while reducing technology risk, *SAP Insight*, June 2006. Online at: *http://www.sap.com/services/programs/pdf/BWP_Supporting_Business_Innovation.pdf*

[17] B. Rumpe, Agile test-based modeling, Proc. of *Software Engineering Research and Practice,* CSREA Press, 2006, pp. 10-15.

[18] SAP AG, *Enterprise SOA in a Nutshell*, 2007.Online at: *http://help.sap.com/redirect_sdn_esoa/redirect_esoainanutshell.htm*

[19] K. Schwaber and M. Beedle, *Agile software development with SCRUM,* Prentice-Hall, 2001.

[20] M. Utting and B. Legeard, *Practical model-based testing, a tools approach*. Morgan Kaufmann Publishers, 2007.

[21] S. Wieczorek, A. Stefanescu, and J. Großmann, Enabling model-based testing for SOA integration testing, In Proc. of *Model-based testing in practice (MOTIP'08)*, satellite workshop of ECMDA'08, 2008, pp 77-82.

[22] S. Wieczorek, A. Stefanescu, and I. Schieferdecker, Test data provision for ERP systems, In Proc. of *Int. Conf. on Software Testing, Verification and Validation (ICST'08)*, IEEE Computer Society, 2008, pp 396-403.

[23] D. Woods, and T. Mattern*, Enterprise SOA - Designing IT for Business Innovation*, O'Reilly, 2006.

[24] World Wide Web Consortium (W3C), *Web Services Choreography Description Language (WS-CDL) Version 1.0* Online at: *http://www.w3.org/TR/ws-cdl-10*

[25] J.M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, G. Decker, Service interaction modeling: bridging global and local views. In Proc. of *Enterprise Distributed Object Computing Conf. (EDOC'06)*, IEEE Computer Society, pp. 45-55.