Towards Automated Test Practice Detection and Governance

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Abstract—**The selection, monitoring, and adjustment of quality measures are fundamental to software engineering, and testing is a key quality assurance activity. In Small and Medium Enterprises (SMEs), it is often difficult and time consuming to manually ascertain the degree and type of test practice usage and related process compliance, thus such data collection may be omitted. Moreover, any manual data collection may not be objective, comprehensive, and dependable, since manual collection cannot typically be done transparently with software engineers. Considering test-driven development, the intention and the order of programming are important, and few clues are left ex post that can be objectively verified. This paper presents an approach that enables an automatic test practice detection capability using the SEEEK (Software Engineering Environment Event-driven frameworK) to support adaptable processes while ensuring process compliance and supporting governance. The results show the feasibility of this approach for automatically detecting test practices and adjusting developer task management accordingly.**

Keywords- automated test practice detection; software engineering environments; complex event processing; process-aware information systems; test-driven development

I. INTRODUCTION

Many software engineering (SE) processes today rely on human-triggered quality assurance activities. Various testing practices are often touted as "best practice" by proponents, yet the degree to which they are or should be actually followed under which circumstances is left nebulous. The clichés "practice what you preach" and "the exception proves the rule" come to mind. Specifically, test-driven development (TDD) practices [2] are acclaimed to be gaining momentum; yet they may not always be practicable in certain circumstances, and comparative analysis of empirical studies show mixed results [13].

Testing practices are an integral part of SE processes (such as OpenUP [24], RUP [25], XP [26], and VM-XT [27]), and compliance and governance of SE processes is essential for process maturity and process effectiveness. Many SE organizations are required to define and establish processes or meet compliance requirements (e.g., CMMI, ISO 15504, or ISO 9001), yet are faced with difficult decisions about certain practices without reliable data. To what degree are the practices being followed (in the organization and in others), and when and under what circumstances are exceptions occurring, and do the exceptions make sense and can they be categorized? Additionally, manual process compliance and exception checks are time consuming and, at least for SMEs, perhaps too costly. The nature of SE projects is such that any processes must allow for a reasonable degree of adaptability within the guidelines of the organization. Process-Aware Information Systems (PAIS) have shown promise for addressing process adaptation in organizations [4]. Yet due to the rapid change in SE, discrepancy analysis, adaptation of processes and practices, and governance will continue to be challenges for SE organizations.

Additionally, SE environments (SEEs) face technical challenges for automated practice detection. Typical SEEs lack the capability of systematic and automatic process or practice data collection, especially to a fine granularity. One reason is that few tools are equipped to provide the necessary data in an interoperable format. Another reason is due to the typically heterogeneous and highly tailored project-specific tool environments in SEEs. Currently no practical solution has established itself that automatically detects and correlates test (or more generally SE) practice occurrence and additionally adjusts the process enactment in accordance with governing policy.

Since the typical testing effort for software projects is said to lie in the range of 40-60%, ensuring proper utilization of limited test resources is crucial. Automated, non-intrusive, and transparent data collection and analysis in SEEs could yield comprehensive, objective, and dependable awareness and answers to the questions about the effectiveness, efficiency, and usage of test techniques by software engineers.

Yet the usage of test techniques are rarely accurately, consistently, dependably, transparently, and affordably measured in the specific organizations within the SE project settings in which they occur. To alleviate this situation, support for the automatic detection of SE test practices should address the following requirements:

1) SE tools utilized in testing should provide automated indications of relevant changes in their state, e.g., via events, in a manner transparent to the software engineers. This enables automated data collection, analysis, and practice detection.

2) Test practices detected during workflows should be archived for later analysis. Exception analysis and the promotion/demotion of practices require a history.

3) Automated current and next task guidance to software engineers within SE processes should be provided for

process compliance, yet adaptation within guidelines supported within given compliance restrictions.

4) Correctness and validity of the workflows and process instances should be automatically verified.

Machine-based detection of test practices is dependent, among other things, on correlating low-level events (e.g., from tools) with some expected pattern match to a known activity. Compared to other practices that can be detected based on source code presence or tool usage events, TDD presents specific challenges since few if any hints remain ex post. Note that while TDD may be viewed exclusively as a development practice, for purposes of this paper it is considered a test practice. Reasons for automatically assessing TDD usage (or other test practices) are economical evaluation for practice promotion, demotion, or constraint determination, process compliance exception analysis, etc. And while manual data collection and analysis is possible, eliciting the data (e.g., via any explicit questionnaires) may cause the Hawthorne or observer effect to a greater extent (as would reminder checklists) than other more transparent forms of data collection.

This paper presents an approach using SEEEK (Software Engineering Environment Event-driven frameworK) for addressing the aforementioned challenges and requirements with regard to automated test (or SE) detection, adaptation, and governance. A TDD scenario was chosen for evaluation of the approach due to the challenges it presents. The paper is organized as follows: section II discusses related work. Section III describes the solution approach followed by a description of the realization. Section V evaluates the solution which is then followed by the conclusion and future work.

II. Related Work

With regard to automatic detection of TDD, [22] investigated automated TDD assessment using only CVS logs, which may encumber developers who do not wish to commit until they have completed coding and testing. TestFirstGauge [21] uses an Excel spreadsheet to aggregate and mine log data collected by Hackystat and determine and visualize TDD process conformance. Ratios of test vs. production for code and effort and cycle distribution are used as indicators of TDD process conformance. TDD-Guide [16] is an Eclipse rule-based tool whose intent is to provide guidance for proper TDD usage. It instantly detects TDD non-conformance. [15] studied TDD compliance with an Eclipse plugin that records low-level developer activity. It no longer appears to be active. Zorro [12][11] is a fully automated TDD detection system that utilizes Hackystat, but relies on the JESS rules engine for monitoring process compliance and SDSA for event stream processing.

While detecting TDD, the above related work does not address the larger issue of the intertwining of test practices and SE lifecycle processes with regard to the aspects of organizational SE process compliance and governance as a whole.

III. Solution Approach

To address these requirements, the general idea of the solution approach is to extract and store events transparently during SE tool usage, detect higher level activities via lower level event pattern matching, and appropriately adjust process management which is used to guide software engineers.

The Software Engineering Environment Event-driven frameworK (SEEEK) employs a mixture of architectural components (the upper colored modules shown in Fig. 1) and event processing (shown in Fig. 2) to deal with the various facets involved.

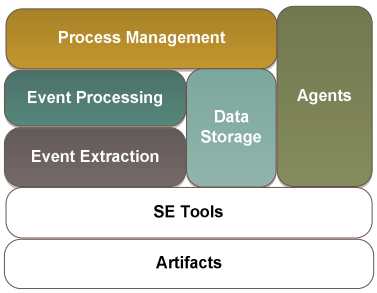

Figure 1. SEEEK Conceptual Architecture

Artifacts is a placeholder for the artifacts that are produced or used in a software project. These are directly or indirectly accessed on a file system usually via the SE tools. *SE Tools* is a placeholder for the independent SE tools used for development and testing. *Agents* provides reactive management agents for each SE tool for application control and integration in the architecture. The agents are employed in the style of the blackboard architecture pattern [10]. *Event Extraction* consists primarily of event sensors and data collection for SE tools. These sensors may already exist, e.g., embedded as a plugin in an IDE, or alternatively event generation for a tool can be done by a tool agent. *Data Storage* provides event and data storage as an XML Space implementation similar in ways to the tuple space concept [8]. *Event Processing* applies CEP and any contextual annotation to events, although other agents may annotate as well. *Process Management* is responsible for SE workflow conformance of activities and supports reactive and proactive task management.

Specifically for SEEs, if a high degree of process rigidity is required, the utility and chances of adoption are reduced, especially for SMEs. PAIS separate process logic from functional and data-centric application code, whereby a key criteria is effectively handling process change [20][17][19]. For governance considerations, the correctness and validity/compliance checking of workflows is important, and PAIS research in this area includes [14].

The processing of SEE events involves multiple steps as shown in Fig. 2. Events from SE tools are acquired and then stored, optionally annotated with any relevant contextual information. This is followed by complex event processing (CEP), which enables the detection of complex patterns of multiple events, event correlation, and event abstraction.

Once complex events are detected, workflow adjustments are made in a PAIS, and software engineers are informed about changes via Task Management.

Figure 2. Event Process Flow

The combination of these various structural components in the architecture put the mechanisms in place for fulfilling the requirements of section II

IV. TDD Scenario Realization

While there are many possible test practices, TDD detection was chosen due to the challenges involved, including temporal aspects and changes to both test code and source code and utilization of various tools (unit test, code coverage, version control, etc.).

The concrete solution realization for the TDD scenario consisted of the following implementations as shown in Fig. 3 and described below:

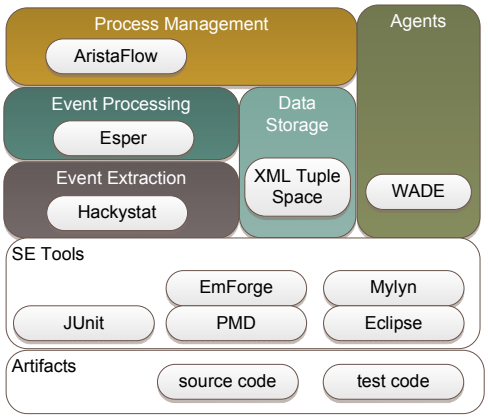

Figure 3. SEEEK Implementation Architecture

SE Tools: Eclipse was used due to its proliferation, Java support, and open plugin architecture; JUnit was used due to its proliferation and the availability of a Hackystat [9] sensor; PMD [23] was used for static analysis due to its rule extensibility and Hackystat sensor availability; and EmForge [5] and Mylyn [18] were used for task management due to their good integration in eclipse. Agents: contains agents implemented using WADE [3], which was chosen due to its Java support and agent workflow capability. Event Extraction: Hackystat was used due to the availability of multiple sensors and a REST Web Service-based collection mechanism. Data Storage: a Java Web Service-based XML Space service was implemented using Apache CXF and the eXist [7] XML native database with "round tripping" backend in order to store and retrieve the events in XML. Event Processing: the Java-based Esper [6] was used for CEP. Process Management: ADEPT2 [1][4] was used as a PAIS due to its Java compatibility and its adaptability and correctness features.

A. Process Governance with ADEPT2

For an archetype SE process, OpenUP was chosen. A process based on the OpenUP "Develop Solution Increment" workflow is shown in shown in Fig. 4, whereby a modification was made to allow test-first as well as implement-first test practices.

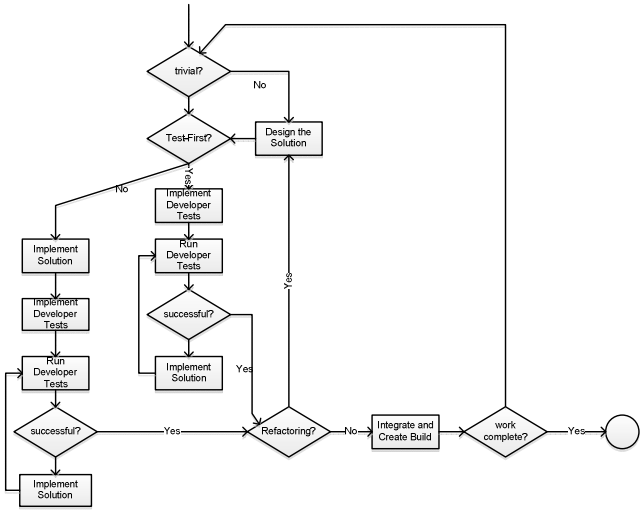

Figure 4. Modified OpenUP Workflow

A corresponding template in ADEPT2 was created. Once testFirst is detected, it is saved as a fact for use in the next iteration. The Fig. 5 shows this option.

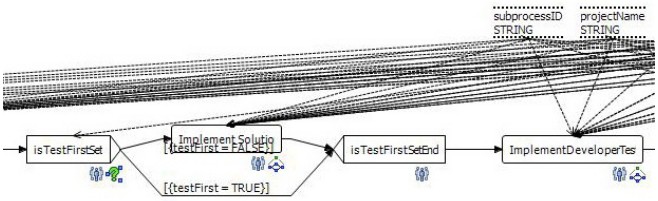

Figure 5. Develop Solution Increment Template Screenshot

An Activity Template was created for each activity, wherein the activity name, parameters necessary for the Start Task Template, and a stopTask were defined. Each activity is realized as a subprocess instantiated from the Develop Solution Increment Template. As shown in Fig. 6 for the Implement Solution Increment, Test-First detection is included for this activity. Note that the overlapping in the layout was not adjustable for the screenshot.

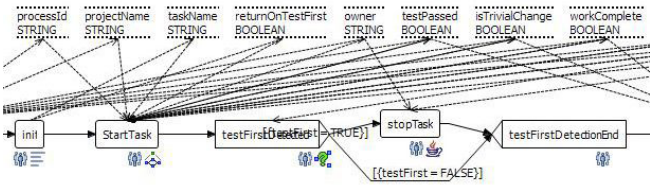

Figure 6. Implement Solution Increment Template Screenshot

The Start Task Template is a helper template that is used by every Activity Template to invoke runTask (see Fig. 7), which waits until either notified by the EmForge agent that a task is completed or until a Test-First event was detected. It then sets control values via small scripts.

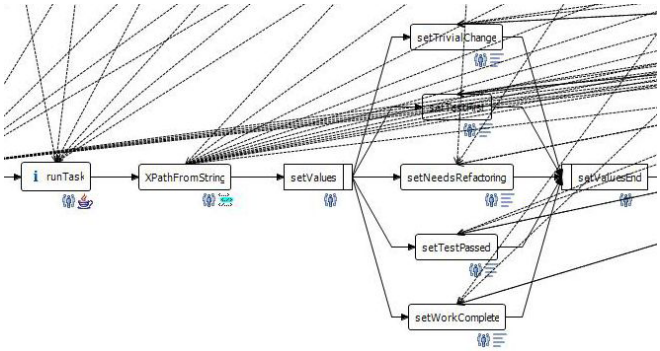

Figure 7. Start Task Template Screenshot

The ADEPT2 Process templates need a mechanism to place new tasks in the EmForge task management system and wait for their completion. A component TaskRunner was developed that provides two methods: runTask which starts a new task and waits for the end event in the Space, and the method stopTask which removes an open task. A sequence diagram for runTask is shown in Fig. 8.

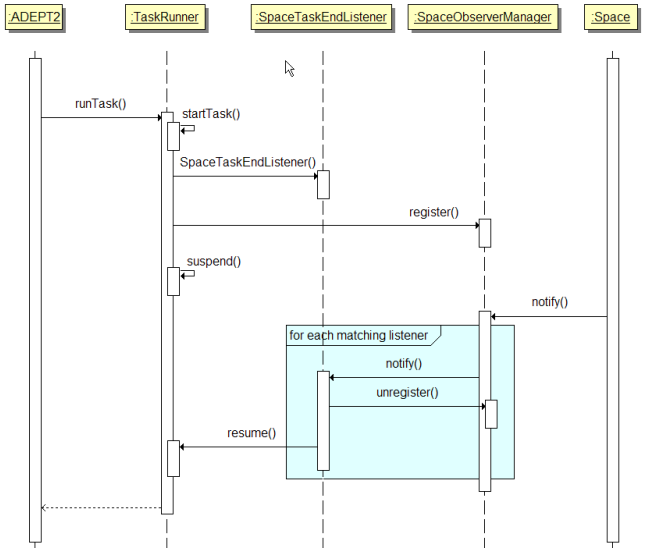

Figure 8. Task Management Sequence

The PAIS governs the SE process enactment within the Process Management module in conjunction with EmForge tasks shown in the Eclipse IDE via the Mylyn plugin.

B. Complex Event Processing with Esper

Since TDD has to some extent to do with the intent of the software engineer, the detection of TDD depends on detecting patterns that are indicative of TDD usage. To detect the complex Test-First event, a hierarchical abstraction with helper events (shown as the middle row in Fig. 9) to simplify processing was specified in Esper using patterns. The bottom row consists of the simple events generated by the Hackystat sensors.

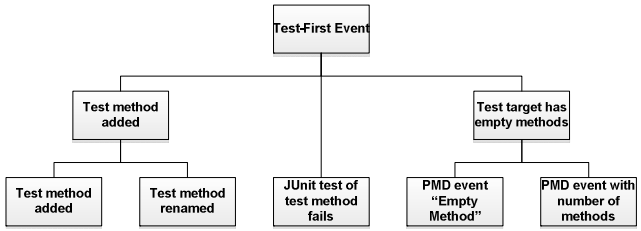

Figure 9. Complex Test-First Event

Various assumptions were necessary to handle this simplified case: the test methods had the same base name as the target implementation method, PMD must be called after running the test in order to provide the count of empty methods, and the developer must be following the TDD paradigm (no blurred scenarios).

Two new custom PMD rules were necessary to determine the number of truly empty methods since PMD does not provide information on empty methods: one to count empty methods that contain at most a return statement but ignore proper get methods. Since the Hackystat PMD sensor removes the actual method names and only leaves an empty method count, the count was used pragmatically rather than the more ideal method name matching.

Detection of patterns and abstraction levels are shown in Fig. 10, with complex events shown in black.

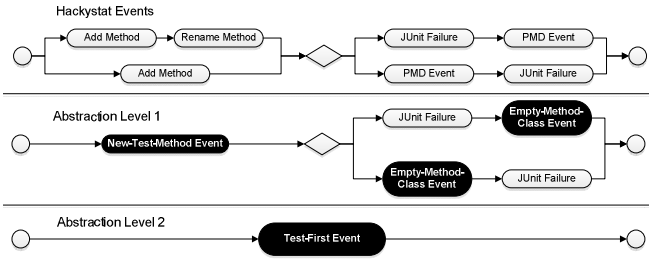

Figure 10. Event Abstraction Levels

In summary, the solution approach for automatic TDD detection and OpenUP process enactment adjustment was realized on the basis of the SEEEK Implementation Architecture in combination with the ADEPT2 templates and custom components, the Esper Helper events and patterns, and the PMD custom rules.

V. RESULTS

To evaluate initial technical limitations of this approach, both functional and performance evaluations were performed.

Functional testing consisting of applying the TDD scenario and monitoring the Mylyn tasks. When test-first was detected for an activity, the EmForge tasks as viewed in Mylyn (see Fig. 11) were observed to be adjusted automatically.

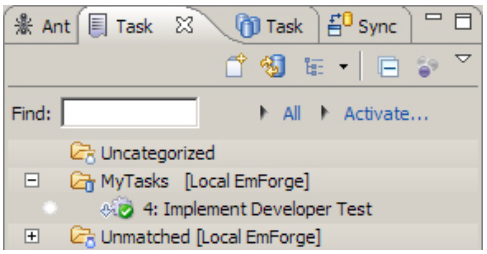

Figure 11. Mylyn Task List

For CEP performance testing, the test system consisted of an AMD Athlon 64 1,8 GHz Processor (3000+), 1.2GB RAM, Windows XP Pro (SP3), Esper 3.0.0, and Java Runtime Environment 1.6.0_07. For each scenario, the average of 100 measurements from the time of the significant PMD Event to Test-First Event detection was used. The following scenarios were used to determine the Test-First event performance:

1) Reaction time for a complex Test-First event when there is a 10ms lag between the individual events: Add Method Event, JUnit Failure Event, and a PMD Event. The reaction time was 23 ms.

2) As 1) above, additionally test-first-irrelevant PMD Events are included at 200ms intervals. The reaction time was 57 ms.

3) As in 1) above, while additionally Add Method Events that are irrelevant to the Test First Event are included at 200ms intervals. The difference to 2) is that a new pattern is started that waits for sequential events. The reaction time was 69 ms.

Based on these measurements summarized in Table I, an increasing number of non-relevant events slowed the detection reaction time in Esper, which might affect its scalability. Based on the event arrival rate expected in actual smaller project scenarios, the reaction time appears usable. Performance tuning and scalability for large projects will be considered in future work.

TABLE I. CEP PERFORMANCE TESTING

Case	Reaction time (ms)
1	23
2	57
3	69

To consider the performance of the task management integration approach, the performance for the ADEPT2 TaskRunner component was measured for the sequence shown in Fig. 8 for a single process instance (case A) and separately with 2 simultaneous process instances (case B).

The test configuration consisted of VMWare Player 2.5.1 containing Windows XP Pro SP3 with 1GB RAM, hosted on an AMD Athlon 64 (3000+, 1,8 GHz), 1.3 GB RAM and Windows XP Pro (SP3). The VM ran ADEPT2, Apache Tomcat 6.0.18, EmForge 0.27, Apache CXF 2.2, Java Runtime Environment 1.5.0_18, and the XML Space with an eXist 1.2.4 database. A VM was needed due to firewall incompatibility issues that are being addressed.

A measurement template Template Sequence was created that realizes a sequence of 3 sequential activities. The TaskRunner component, which is executed by the ADEPT2 Template, places a new task in EmForge, and waits for its completion, was modified to generate log entries at the start of each activity and to generate an immediate notify. Two differences between these 3 times resulted from a run, and 10 manual runs were made and the average of the 20 values was used resulting in 20.7 seconds per activity.

For the parallel case B, 4 time differences resulted from each run and 5 manual runs were done, with the average of the 20 values resulting in 33.8 seconds per activity.

TABLE II. TASK RUNNER COMPONENT PERFORMANCE OVERHEAD

Case	Activity overhead (sec)
A	20.7
B	33.8

As summarized in Table II, the current implementation of Task Runner has performance and scalability issues. These can be attributed to the high overhead of starting the Apache CXF SOAP notification Listener for each runTask(). An optimization that allows a singleton of this Listener will be investigated in future work.

VI. CONCLUSION AND FUTURE WORK

Due to the relatively large amount of effort related to testing in SE projects, the efficiency and effectiveness of test practices is crucial in today's business environment. Process compliance requirements also incur overhead and provide documentation challenges, especially for SMEs. Technical challenges remain in current SEEs for automatically capturing and analyzing test (and SE) practices.

The solution approach based on SEEEK to address automatic test practice detection was described and a TDD scenario was selected for evaluation due to the challenges posed. The results show that CEP is practicable for addressing automated test practice detection. While the results were applied to TDD, other test practices can be analogously detected due to the event-based nature. By combining PAIS with SE task management tools within an event-driven architecture, process guidance and adaptability are supported for the software engineer, while conformance and governance are supported for the quality professional.

Although the general and flexible event-driven approach to address the challenges in test practice detection and governance in SEEs is promising, future work is needed to optimize the implementation and evaluate SEEEK in industrial SME project settings.

ACKNOWLEDGMENTS

The author thanks Michael Vögeli for his assistance with the experiments, implementation, and providing figures, and the Institute of Databases and Information Systems at Ulm University for access to ADEPT2 technology.

REFERENCES

[1] P. Dadam, M. Reichert, S. Rinderle, M. Jurisch, H. Acker, K. Göser, U. Kreher, M. Lauer: "ADEPT2 - Next Generation Process Management Technology". Heidelberger Innovationsforum, Heidelberg, April 2007.

[2] K. Beck, Test Driven Development: By Example, Addison-Wesley Professional, Massachusetts, ISBN 0321146530, 2002.

[3] G. Caire, D. Gotta, and M. Banzi, "WADE: a software platform to develop mission critical applications exploiting agents and workflows," Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track, 2008, pp. 29-36.

[4] P. Dadam and M. Reichert, "The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support," Technical Report UIB-2009-01, Fakultät für Ingenieurwissenschaften und Informatik, Ulm University, Germany, 2009.

[5] EmForge http://www.emforge.org [June 17, 2009]

[6] http://esper.codehaus.org [June 17, 2009]

[7] eXist http://exist.sourceforge.net [June 17, 2009]

[8] D. Gelernter, "Generative communication in Linda," ACM Transactions on Programming Languages and Systems, vol. 7, nr. 1, January 1985, pp. 80-112.

[9] Hackystat Framework http://www.hackystat.org [June 17, 2009]

[10] B. Hayes-Roth, "A blackboard architecture for control," Artificial Intelligence, vol. 26, issue 3 (July 1985), pp. 251-321.

[11] P. M. Johnson and H. Kou, "Automated Recognition of Test-Driven Development with Zorro," AGILE 2007, pp.15-25.

[12] H. Kou and P. M. Johnson, "Automated recognition of low-level process: A pilot validation study of Zorro for test-driven development," Proceedings of the 2006 International Workshop on Software Process, Shanghai, China, 2006.

[13] H. Kou, P. Johnson, and H. Erdogmus, "Operational Definition and Automated Inference of Test-Driven Development with Zorro", 2009, http://csdl.ics.hawaii.edu/techreports/09-01/09-01.pdf [accessed June 17, 2009]

[14] L. T. Ly, S. Rinderle, and P. Dadam, "Integration and verification of semantic constraints in adaptive process management systems," Data and Knowledge Engineering, Vol. 64, No. 1, 2008, pp. 3-23.

[15] L. Madeyski and L. Szala, "The impact of test-driven development on software development productivity - an empirical study," In: Software Process Improvement, 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, Springer-Verlag, Heidelberg, Germany, Lecture Notes in Computer Science, vol. 4764, 2007, pp. 200–211.

[16] O. Mishali, Y. Dubinsky, and S. Katz, "The TDD-Guide Training and Guidance Tool for Test-Driven Development" In XP 2008: 9th International Conference on Agile Processes in Software Engineering and Extreme Programming, Springer-Verlag, Heidelberg, Germany, Lecture Notes in Business Information Processing, vol. 9, pp 63–72.

[17] R. Müller, U. Greiner, and E. Rahm, "AgentWork: A workflow system supporting rule-based workflow adaptation," Data and Knowledge Engineering, 51 (2), 2004, pp. 223-256.

[18] Eclipse Mylyn Open Source Project http://www.eclipse.org/mylyn/ [June 17, 2009]

[19] M. Pesic, M. Schonenberg, N. Sidorova, and W. van der Aalst, "Constraint-based workflow models: change made easy," In: Proceedings of the 15th Int'l Conf. on Cooperative Information Systems (CoopIS'07), Vilamoura, Algarve, Portugal, LNCS 4803, 2007, pp. 77-94.

[20] M. Reichert and P. Dadam, "A framework for dynamic changes in workflow management systems," In: Proc. 8th Int'l Workshop on Database and Expert Systems Applications, Toulouse, 1997, pp. 42–48.

[21] Y. Wang and H. Erdogmus, "The role of process measurement in test-driven development," In: Extreme Programming and Agile Methods - XP/Agile Universe 2004, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, 2004, ISBN 978-3-540-22839-4, pp. 32–42.

[22] C. Wege, "Automated support for process assessment in test-driven development," PhD thesis, Eberhard-Karls-University at Tubingen, 2004.

[23] http://pmd.sourceforge.net/ [June 17, 2009]

[24] http://epf.eclipse.org/wikis/openup/ [June 17, 2009]

[25] P. Kruchten, The Rational Unified Process: An Introduction, 2nd ed., Addison-Wesley Professional, ISBN 0201707101, 2000.

[26] K. Beck, Extreme Programming Explained: Embrace Change, 2nd ed., Addison-Wesley Professional, ISBN 0321278658, 2005.

[27] http://v-modell.iabg.de/v-modell-xt-html-english/index.html [June 17, 2009]