2011 Ninth International Conference on Software Engineering Research, Management and Applications

# Agile Model Driven Development: An Intelligent Compromise

Reza Matinnejad
Information and Communication Technology Institute
Isfahan University of Technology
Isfahan, Iran
r_matinnejad@alum.sharif.edu

***Abstract*—Both the model-driven and agile development approaches have significantly enhanced productivity and predictability of software development in practice. Agile Model Driven Development or AMDD is an attempt to effectively bring together the fast pace of agile development and the guaranteed quality of model-driven development. Despite the proliferation of processes claim to comply with AMDD principles, there has been little academic research examining merits and demerits of such an approach. This paper first presents the underlying theoretical foundations of AMDD and then reviews, classifies and compares existing AMDD processes. The results indicate significant disparity between compared processes that implies the AMDD concepts are still in their infancy.**

***Keywords- Agile Model Driven Development; Agile Modeling; Model Driven Architecture; Agile Software Development***

## I. INTRODUCTION

Model Driven Development or MDD is an approach to software development that expands the role of models in software development process from a chance to think through complex issues before code development to the primary artifact of all software development activities. While automatic code generation from a chain of models is the ultimate objective of MDD, several important software quality attributes such as reusability, portability and interoperability are additional potential benefits of MDD thanks to the OMG's MDA standard [1]. As extensively discussed in [2], it should be kept in mind that MDD is not a concrete methodology, but a generic approach that can be applied to software development processes to take advantage of its promises.

Agile software development is an iterative and incremental approach to software development, based on a collection of invaluable concepts and principles [3]. To be agile, the software development process must be kept as lightweight as possible; however agile methodologies are highly-disciplined at the same time. Actually the need for a clear-cut process in methods that claim to be agile is inevitable in order to effectively benefit from applying agile principles, and also to keep critics from accusing agile methods of being nothing but modern "code and fix".

As one may expect, Agile Model Driven Development or AMDD is the agile-flavored MDD or the MDA-based agile development. But what does that exactly mean and how is that technically possible to merge such diverse technologies? While MDD is model-centric, agile software development highly values executable code and hesitates to develop extensive models as well as other non-code artifacts. In fact, this documentation phobia is so strong that sometimes has resulted in rather questionable assumptions in agile software development, like the principle of identifying face-to-face conversation as the most effective way of information interchange within a software development team [3]. On the other hand, MDD has such strong faith in modeling that has sparked debates, even among advocates of modeling [4] [5]. The rest of this section describes how to resolve this seeming contradiction.

AMDD can best be described, in our opinion, as an intelligent compromise. Due to the contrasting views of agile and model-driven development, an effective compromise should be reached in order to gain the advantages of modeling without suffering from the disadvantages, which is the principal goal of AMDD according to [6]. To fulfill this aim, Ambler introduces the concept of "barely good enough" models and documents, which are only adequate for the given task [6]. Ambler also dedicates a chapter of his book "The Object Primer" [7] to AMDD principles that is perhaps the most comprehensive overview of AMDD. He presents a high-level life cycle for AMDD, which is later examined and evaluated in comparison to other AMDD processes.

As mentioned earlier, neither MDD nor agile development is a concrete software development methodology; however there exist several MDD-based [2] as well as agile [8] methodologies in the literature. Consequently, a method engineering approach could be employed when it comes to develop an AMDD process [9]. It is possible either to integrate MDD principles and development tools into existing agile software processes, or to introduce agility into current MDD-based methodologies through agile practices and guidelines, which are both obvious examples of extension-based method engineering approach. Throughout the paper, we refer to them as agile-based and MDD-based strategies, respectively. It is also possible to employ an assembly-based method engineering approach and construct an AMDD methodology using method fragments from other agile and MDD-based methodologies. Since these method

978-0-7695-4490-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERA.2011.17

engineering approaches can be easily distinguished among contemporary AMDD processes, the processes are classified according to their method engineering strategy.

This paper presents a survey of AMDD processes and highlights the current state of the art of the technology. The most prominent AMDD processes in the literature are discussed in terms of their objective, their development process, and their main contribution towards AMDD technology. They are classified, compared and evaluated according to an objective criteria set. The evaluation results have been used to characterize and study AMDD processes.

The remainder of this paper is organized as follows: Section 2 briefly reviews the existing AMDD processes. Section 3 summarizes, evaluates and compares the introduced processes. This section also provides a detailed analysis of the evaluation results. Eventually, Section 4 concludes the paper.

## II. REVIEW OF AMDD PROCESSES

In this section, we present a review on the fundamental aspects and distinguishing features of the existing AMDD processes. Each process is characterized in terms of the application, and thereby, the remarkable contributions of the process as well as an informative process model diagram.

### A. Sage Process

Sage is a model-driven method that employs an iterative and incremental approach to develop high assurance reactive multi-agent systems [10]. One main goal of Sage is to benefit from agile development advantages in developing high assurance systems through enhancing agile process documentation. For this purpose, a model-driven development method is proposed including four inter-connected models that capture developers' decisions and inject the desired level of documentation. These are the models: *The environmental model*, *the behavioral model*, *the design model*, and *the run-time model*. Fig. 1 shows the models and the transformations required along the development process.

As the names imply, the models have sequential responsibilities during the development process: The boundary of the reactive system, and thus a set of system environmental variables, are defined by the environmental model. The behavioral model is responsible for capturing the system behavior through a selected set of environmental variables called controlled variables. The design model and the run-time model consist of classes and agents respectively, which satisfy the desired functionality through their interaction.

Sage takes advantage of UML class diagrams to depict elements of different models and defines five sets of attributes for each class: provides output, provides input, requires input, requires output, and local variables. Sharing attributes between different classes make it possible for them to interact with each others.

Sage doesn't incorporate an explicit agile process, but attempts to adopt a subset of fundamental agile practices

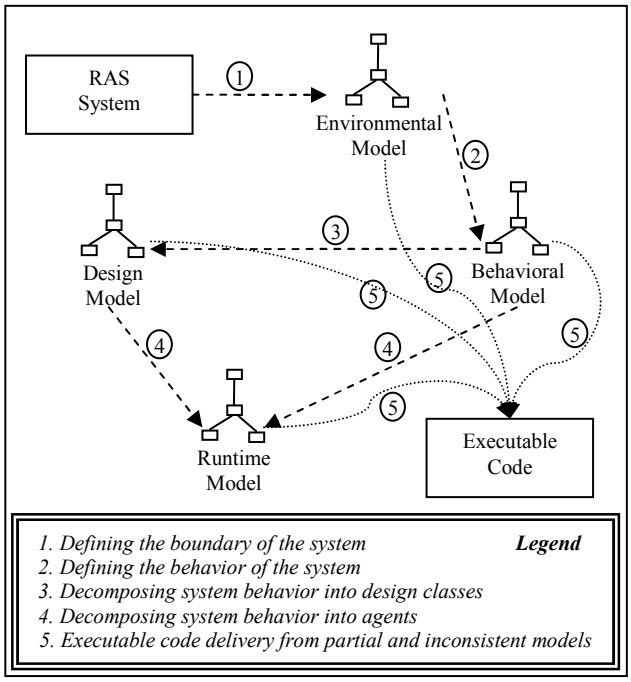


Figure 1. The Sage Process

and principles. As illustrated in Fig. 1, the tool set of Sage provides support for delivery of executable code from incomplete and even inconsistent models, which enables frequent delivery of working software to the customer. In addition, despite the sequential nature of the models, they can be constructed in any preferred order enhancing responsiveness of the process to customer needs, both early and late in the project.

### B. MDD-SLAP Process

System level agile process (SLAP) is a Scrum-based [11] agile methodology, developed and adopted at Motorola. SALP splits the software life cycle into successive iterations, and each iteration, in its turn, comprises three sprints, namely: *requirements and architecture*, *development*, and *system integration feature testing (SIFT)*.

The Motorola's MDD process is a V-model process based on USDP and includes the following development activities: application requirements and architecture, requirements analysis and high-level design, detailed design, code generation and UML unit integration testing, and subsystem testing and system testing.

MDD-SLAP is mainly targeted at real-time telecommunication systems to increase the development pace, ensure frequent delivery and enhance the product quality [12]. It takes an innovative approach to combine SALP and Motorola's MDD process by establishing a simple correspondence between SLAP sprints and the Motorola's MDD process development activities. MDD-SLAP is an explicit example of agile-based AMDD processes, since it employs SALP as the backbone process and defines a mapping from MDD process activities to the

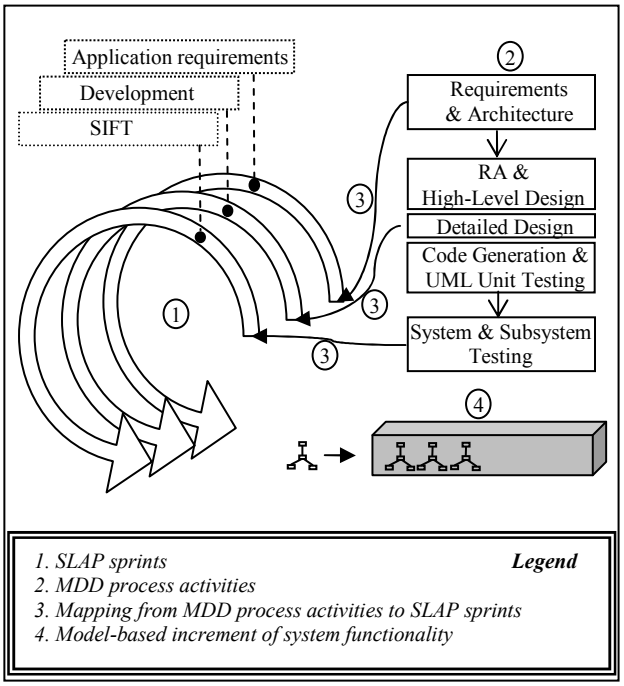


Figure 2. The MDD-SLAP Process

equivalent SLAP sprints, as depicted in Fig. 2. The software is built up from the model increments of system functionality, in an iterative and incremental fashion.

But perhaps the most remarkable contribution of this work is represented as a tabular correspondence between MDD and agile development activities as well as management practices. This correspondence defines two types of relationships; corresponding and related. The corresponding relationship means that the agile practice is equivalent to the counterpart MDD practice. The related relationship means that the agile and the corresponding MDD practices support or enable each other. These relationships help to decide where and how to apply agile practices to MDD activities during the development process.

But perhaps the most remarkable contribution of this work is represented as a tabular correspondence between MDD and agile development activities as well as management practices. This correspondence defines two types of relationships; corresponding and related. The corresponding relationship means that the agile practice is equivalent to the counterpart MDD practice. The related relationship means that the agile and the corresponding MDD practices support or enable each other. These relationships help to decide where and how to apply agile practices to MDD activities during the development process.

### C. *Hybrid MDD Process*

Hybrid MDD process, the name the process developers selected upon our request, specifically aims at effectively and economically applying MDD to small and middle size projects, which seriously lacks in the current MDD literature [13]. Hybrid MDD selects MDD process elements that are more appropriate to develop small-scale projects, and introduces these elements into a traditional lightweight process, which is opted due to the project development team experience.

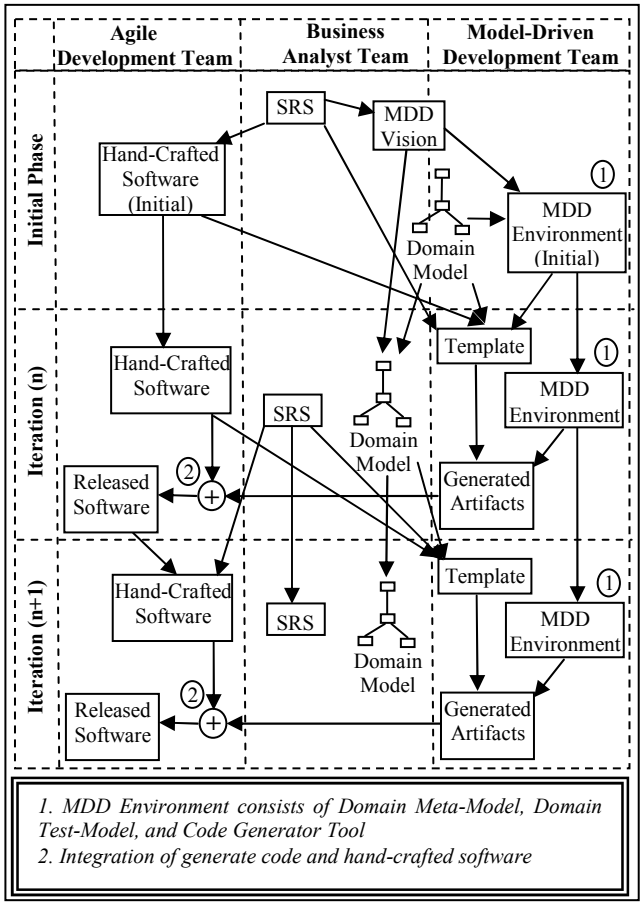


Figure 3. The Hybrid MDD Process

Under tight time and budget constraints, the process developers have decided to integrate the following features into the process, in order to mitigate the risks of AMDD technology adoption:

- Automatic code generation, using MDD tools, happens only for a selected subset of software features where it pays off. It goes in parallel with traditional handcrafted programming all through the lifetime of the project that makes it possible to draw back from MDD development path whenever necessary
- The MDD development tools are built on demand. In other words, in each iteration and for each new requirement, new features may be incorporated into the MDD tool set.

As seen in Fig. 3, the software is developed through the close collaboration of three different teams. To avoid the diagram being over crowded, only the process artifacts and the effects they have on each other are represented. MDD vision is a strategic artifact that outlines the MDD-

related activities of the project and identifies the requirements to be implemented by MDD part of the process. MDD environment consists of domain meta-model, domain test-model and code generator tool. Domain test-model captures metadata about the model transformations and facilitates testing the code generation process. The other artifacts are common among software development processes and need no further explanation. During the integration activity, represented by a plus sign in the diagram, the generated code is merged into the hand-crafted software. Integration activity results in a release of the working software

### *D. AMDD High-Level Life Cycle*

We have already mentioned the AMDD high-level life cycle, which is the earliest AMDD process in the literature to our knowledge [7]. Though put forward as a generic life cycle, the AMDD high-level life cycle is concrete enough to be used as a standalone process, albeit after a few augmentations. Hence, it is counted as an AMDD process, and is evaluated and compared to other AMDD processes. Fig. 4, which is quoted from [7], shows the activities and iterations of the process.

The high-level life cycle borrows the concept of "cycle", which is an extreme programming term for iteration [14]. Cycle0, named initial modeling, is comprised of two activities: initial requirements modeling and initial architectural modeling. Tasks, such as creating a high-level domain model and risk mitigation of high-priority requirements, occur in initial modeling cycle. Detailed modeling, aka model storming, is the activity where more details are added to the models, of course not more than what is necessary for the given task. Implementation is the most time consuming activity in which the software is implemented following agile implementation practices such as code refactoring and test-driven development.

### *E. Other Works*

Besides the introduced processes, there are other research studies on AMDD worth mentioning such as [15], which introduces an agile approach to develop the platform independent model (PIM) of MDA and [16], which utilizes techniques from model driven engineering to enhance agile development methods. But because these works define no or only a partial process, and since we are primarily concerned with the process rather than other aspects of AMDD, we left them out.

## III. ANALYSIS OF AMDD PROCESSES

In this section, we are going to classify, compare and evaluate the processes reviewed in the previous section. After summarizing the characteristics of the reviewed processes, a criteria-based evaluation is presented and based on the evaluation results, an empirical analysis is made on the AMDD processes.

The introduced AMDD processes are summarized and classified due to their method engineering approach in table 1. As deeply discussed before, MDD-based AMDD

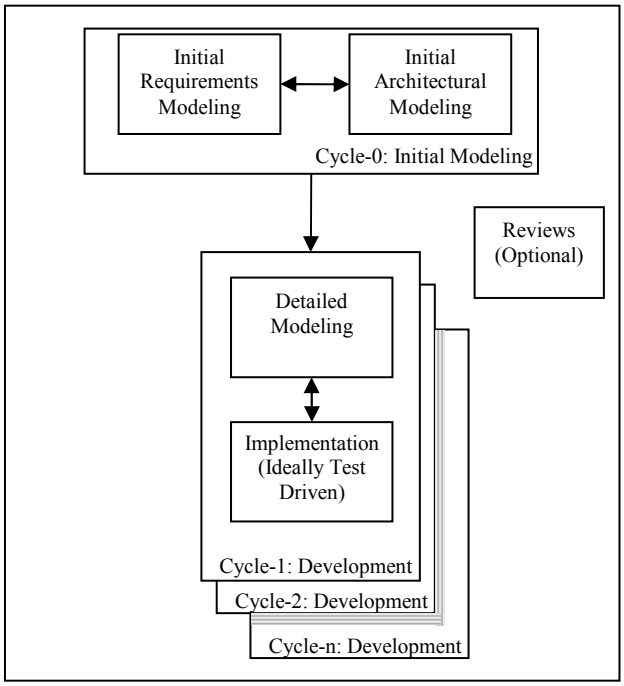


Figure 4. AMDD high-Level Life Cycle [6]

processes are the ones originated from a model-driven method and agile principles are incorporated into them, while agile-based AMDD processes are the ones derived from introducing MDD into an agile method as the skeleton process.

Table 1 also indicates the main objective, as well as the main contribution of each AMDD process. Benefiting from the capabilities of AMDD approach, the processes aim to address the problems and limitations of MDD or agile development. One of the greatest potentials of AMDD is to scale agile approaches. In [17], Ambler discusses the problem in detail and counts AMDD among several techniques that potentially scale agile development. As the AMDD high-level life cycle is suggested in [17] as an AMDD process, scaling agile development is considered as the main objective of the AMDD high-level life cycle.

Finally, the year of introduction of each process is listed in the last column of the table. Evidently, the concept of AMDD is still young and demands greater attention in research and practice.

In the rest of this section, first we provide a criteria-based evaluation, represented in a tabular form, and then perform an analysis on the evaluation results.

### *A. Evaluation of AMDD Processes*

Like every other child, AMDD inherits some of its characteristics from one parent and some from the other, and some of its properties are intrinsic to itself. As a result, here the evaluation criteria are organized into the following complementary categories: Agility Evaluation Criteria, MDD Evaluation Criteria, and AMDD Evaluation

TABLE I. SUMMARY OF EXISTING AMDD PROCESSES

| Process Name | ME Approach | Main Objective | Main Contribution | Year |
|---|---|---|---|---|
| Sage | MDD-Based | Applying agile approach to high assurance software | Supporting executable delivery from partial conflicting models | 2006 |
| Hybrid MDD | Assembly-Based | Applying MDD approach to small and middle size projects | Supporting the partial usage of MDD activities in collaboration with traditional programming practices | 2009 |
| MDD-SLAP | Agile-Based | Benefiting from both agile and MDD advantages in developing real-time telecommunication systems | Establishing a simple, yet fundamental correspondence between MDD activities and agile practices | 2011 |
| High-Level Lifecycle | Agile-Based | Scaling agile development | Putting forward the notion of AMDD approach | 2004 |

Criteria. The agility and MDD evaluation criteria are selected from [2] and [18], respectively, and are slightly refined and then used.

The AMDD evaluation criteria, developed as a part of this research, are discussed and explained here:

- Smoothness: By smoothness we mean how uniform and consistent the process is from the process user's point of view. Although an AMDD process is the result of a unification procedure, it must give the impression of being one single seamless process to the developer.
- Coherency: By coherency we mean how united and tightly connected the process elements are. An AMDD process must bring together the roles, activities and artifacts from agile and MDD processes, in a clear and well-organized manner.
- Simplicity: By simplicity we mean how simple and easy to understand the process is. This criterion is derived by considering that excessive complexity is a major threat to the practicality of an AMDD process, since there are too many goals and objectives when designing a new AMDD process.
- Generality: By generality we mean how general and applicable to several problem domains the process is. Seeing that the reviewed AMDD processes focus on a specific problem domain, this criterion assesses whether they are applicable to a wider range of projects or not.

In Table 2, the AMDD processes are evaluated, and thus can be compared with each other, based on the proposed evaluation criteria. It should be noted that a table entry is ticked if the process meets the corresponding criterion either explicitly or implicitly.

### B. *Analysis of the Evaluation Results*

According to the review findings and the evaluation results, here we present a list containing the outputs of our analysis.

- AMDD processes adopt a diversity of methods

TABLE II. EVALUATION OF AMDD PROCESSES

| | AMDD Process / Criterion | Sage | Hybrid MDD | MDD-SLAP | High-Level Life Cycle |
|---|---|---|---|---|---|
| Agility Evaluation Criteria | Flexibility | ☑ | ☑ | ☑ | ☑ |
| | Rapidity | ☑ | ☒ | ☑ | ☑ |
| | Learning | ☒ | ☑ | ☑ | ☒ |
| | Responsiveness | ☑ | ☒ | ☑ | ☑ |
| | Cost Effectiveness | ☒ | ☑ | ☒ | ☒ |
| MDD Evaluation Criteria | Metadata Management | ☑ | ☑ | ☒ | ☒ |
| | Automatic Test | ☒ | ☑ | ☑ | ☑ |
| | Tool Support | ☑ | ☑ | ☑ | ☒ |
| | Problem Domain Analysis | ☑ | ☑ | ☑ | ☒ |
| | Verification / Validation | ☒ | ☒ | ☒ | ☒ |
| AMDD Evaluation Criteria | Smoothness | ☒ | ☒ | ☑ | ☑ |
| | Coherency | ☒ | ☑ | ☑ | ☑ |
| | Simplicity | ☑ | ☒ | ☑ | ☑ |
| | Generality | ☒ | ☑ | ☑ | ☑ |

and paradigms, and show no sign of convergence among themselves. For example none of them comply, even partially, with the proposed AMDD high-level life cycle. This seems to be a consequence of lack of academic research in this area.

- AMDD processes that are in the same group, and thereby use the same ME approach, share common characteristics and exhibit the same behavior to some extent, which corroborates our classification scheme.
- Naturally, both agile-based and MDD-based AMDD processes preserve their original process characteristics, and are more successful in meeting agility evaluation criteria and MDD evaluation criteria, respectively.
- Agile-based AMDD processes are more successful in meeting AMDD evaluation criteria, and hence agile-based approach is generally superior to MDD-based or assembly-based strategies, for designing and constructing a new AMDD process.

## IV. CONSLUSIONS AND FUTURE WORKS

AMDD is a promising research context and a great practical concept, but it is not mature enough and is still in its infancy. We have reviewed existing AMDD processes and classified them based on their ME strategy into three groups: agile-based, MDD-based or assembly-based. We have also analytically examined AMDD processes and analyzed the evaluation results. The analysis results can be used for selecting and adapting existing AMDD processes, as well as creating a new AMDD process. We aim to further this research by proposing an assembly-based AMDD process, which has some process elements from agile and some from MDD methodologies, and uses experiences form evaluation of current AMDD processes, to successfully overcome current limitations of AMDD processes.

## REFERENCES


[1] I. Mukerji, and J. Miller, MDA Guide Version 1.0.1. OMG, 2003.

[2] M. Asadi, R. Ramsin, “MDA-Based Methodologies: An Analytical Survey”, In I. Schieferdecker and A. Hartman (Eds.): ECMDA-FA 2008, LNCS 5095, pp. 419-431, 2008.

[3] K. Beck, et al., “Principles behind the Agile Manifesto”, Agile Alliance, 2001, http://agilemanifesto.org/principles.org.

[4] D. Thomas, “MDA: Revenge of the Modelers or UML Utopia?”, IEEE Software (May/June 2004), pp 22-24, 2004.

[5] A. Uhl and S. W. Ambler, “Point/Counterpoint: Model Driven Architecture Is Ready for Prime Time / Agile Model Driven Development Is Good Enough,” IEEE Software, vol. 20, no. 5, pp. 70–73, 2003.

[6] S. W. Ambler, “Agile Model Driven Development (AMDD)”, http://agilemodeling.com/essays/amdd.htm, 2007.

[7] S.W. Ambler, The Object Primer, Cambridge Univ. Press, 2004.

[8] A. Cockburn, “Agile Software Development“, Addison-Wesley, Boston, 2002.

[9] J. Ralyté, R. Deneckère, and C. Rolland, "Towards a Generic Model for Situational Method Engineering", 15th International Conference on Advanced Information Systems Engineering (CAiSE 2003), Klagenfurt/Velden, Austria, pp. 95-110, 2003.

[10] J. Kirby, “Model-Driven Agile Development of Reactive Multi-Agent Systems”, In Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006.

[11] K. Schwaber and M. Beedle, Agile Software Development with SCRUM, Pearson, 2002.

[12] Y. Zhang and S. Patel, "Agile Model-Driven Development in Practice," IEEE Software, vol. 28, no. 2, pp. 84-91, Mar./Apr. 2011.

[13] G. Guta, W. Schreiner, and D. Draheim, “A Lightweight MDSD Process Applied in Small Projects”, In Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications, IEEE, 2009.

[14] K. Beck, Extreme Programming Explained, Addison-Wesley, 2000.

[15] V. Cuong Nguyen and X. Qafmolla, “Agile Development of Platform Independent Model-Driven Architecture”, vol. 2, pp.344-347, Third International Conference on Information and Computing, 2010.

[16] M. Conrad, M. Huchard, and T. Preuss, “Integrating Shadows in Model Driven Engineering for Agile Software Development”, cisis, pp.549-554, International Conference on Complex, Intelligent and Software Intensive Systems, 2008.

[17] S. W. Ambler, “Agile Software Development at Scale”, In B.Meyer, J. R. Nawrocki, & B. Walter (Eds), 2nd IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007, Poznan, Poland, Oct 2007, LNCS 5082, pp. 1-12 , 2008.

[18] A. Qumer and B. Henderson-Sellers, “An evaluation of the degree of agility in six agile methods and its applicability for method engineering”, Information and Software Technology, 50(4):280–295, 2008.