MAKING MDD AGILE
THE AGILE MODEL-DRIVEN METHOD

Klaus Mairon[1], Martin Buchheit[1], Martin Knahl[1] and Shirley Atkinson[2]

[1]Faculty of Business Information Systems, Hochschule Furtwangen University, Furtwangen, Germany
[2]Centre for Security, Communications and Network Research, Plymouth University, Plymouth, United Kingdom

ABSTRACT

This article takes up the idea of model-driven development in a new way and analyses existing points of criticism of this approach, which is well established in practice. The advantages of model-driven development seem obvious on the one hand, on the other hand there is criticism of the practicable use and the accusation of missing suitable process models. This environment of professional software development is currently characterized by the use of agile process models such as Scrum, XP, etc. However, an agile process model for the use of model-driven development (MDD) does not yet exist. An analysis of the similarities between existing approaches to MDD process models and existing agile modelling techniques forms the basis for the definition of a new agile process model. The Agile Model-Driven Method (AMDM) is the result of these studies.

KEYWORDS

Model-Driven Development, Model-Driven Architecture, Process Model, Agile Method, Software Engineering, UML

1. INTRODUCTION

Model-driven development has been a well-established term in modern software engineering for years. With the Model-Driven Architecture (MDA) defined by the Object Management Group (OMG) back in 2001, a defined industry standard exists for this purpose. It defined a uniform basis for the underlying technologies and modeling languages and thus responded to the trend that model-driven software development (MDSD or MDD) and model-driven engineering (MDE) became increasingly relevant for professional software development [28][33][40]. The MDA provides for a multi-stage transformation from the Computation Independent Model (CIM), through the Platform-Independent Model (PIM) and Platform-Specific Model (PMS) to implementation. The CIM is the model with the highest abstraction level and is the starting point for the other model types, the PIM and PSM. The Platform-Independent Model describes the structures and functional requirements of an application and concretizes the CIM without taking technological aspects of the target platform into account. Only the Platform-Specific Model enriches this model with additional model elements (e.g. stereotypes, tagged values) and defines the final transformation into the implementation [18][32]. The MDA provides the Unified Modeling Language (UML) as the modeling language, whereby any modeling language based on the OMG standards MetaObject-Facility (MOF) and Common Warehouse MetaModel (CWM) is allowed [34]. The UML will be extended by so-called profiles, which enriches the modeling

David C. Wyld et al. (Eds) : IPPR, SOENG, DaMi, CSIT, AIS, CSE, CSIP, CCNET - 2018
pp. 105–118, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.80508

language with the model elements necessary for the business or technical domain. The result is a domain-specific language (DSL) based on UML (or another modeling language).

While the MDA covers the spectrum of model-driven development from the description of technicality to design, construction, deployment, operation and maintenance and describes all development stages via models, this has not been accepted in the practice of software development.

It is true that the model with the functional requirements is also the focus here. But it is often a Platform-Independent Model based on a technically motivated DSL or a mixture of Platform-Independent and Platform-Specific Model, which already contains rudimentary information for the transformation to the target platform [45]. The actual platform-specific model is often not modeled, but the transformation into the code (model-to-artefact transformation) takes place directly on the basis of the PIM. The background to this is the simpler way of incorporating functional changes into the model during the development process, without having to adapt another model - the PSM - accordingly. The automated model-to-model transformations from PIM to PSM, which are otherwise necessary for this purpose, often do not work in round-trip or reengineering or are too complex. The transformation to the target platform, i.e. to the chosen architecture, programming language, test definitions and documents is implicitly carried out in one step of the model-to-artefact transformation (often by a corresponding generator framework). In practice, this is often quite sufficient, since the target platforms rarely change fundamentally and several software products are produced based on a defined form. Examples of this can be a collection of JEE components for a logistics solution or several web and/or microservices for insurance companies. This is also referred to as software product families, i.e. building blocks of different functional contents which, however, all correspond to the same design principles.

Although OMG has laid the foundations for the use of MDD in professional software development by creating the necessary standards, the technology is currently not widespread. Even the positive factors of model-driven development, which Hutchinson et al. [27] name, could not change this. This not only includes a more efficient realization, but also a higher quality and faultlessness of the produced product. Hutchinson et al. see a long history of bad experiences (e. g. with CASE) as the cause. The following chapter identifies and describes further negative influences on the spread of model-driven development.

The lack of suitable process models for MDD as well as the additional complexity and a high initial effort are identified as additional problems. For this reason, we finally present an agile, model-driven process model which is suitable to reduce these influences.

2. PROBLEMS IN THE CONTEXT OF THE MODEL-DRIVEN APPROACH

A more efficient and high-quality software development should be in the interest of the software industry. So what are the reasons for the restrained use of model-driven software development? The following points can be identified:

(a) In their review, the authors Asadi and Ramsin point out that model-driven software development makes no sense without being embedded in a corresponding and supporting process model [5]. And in the scientific environment, there are individual process models for model-driven development. Thus, there are process models, e.g. the ODAC methodology [20] [21], MASTER [31], C3 [26], DREAM [44], MODA-TEL [19] and DRIP Catalyst [22], which all support the model-driven software development. However, Asadi and Ramsin critically point out that the process models they examined did not

adequately support software engineering activities and did not consider overlapping activities.

In addition to Asadi and Ramsin, Chitforoush et al. concluded in [10] that, in principle, there are very few methods for the use of model-driven software development and the description of the processes is usually very incomplete and imprecise.

(b) Another problem is the high initial effort that characterizes an MDD project. This starts with the preparation of the necessary infrastructure, and extends from the development of the DSL, the corresponding metamodels to the definition of the necessary transformations to the target platform. This is referred to as "MDD infrastructure" or "Domain Architecture" in the literature (cf. [45]) and in the process models described in (a). And it is also described as a necessary step in the development process, which is time consuming and costly.

(c) Hailpern and Tarr identify additional problems that arise in the context of model-driven development [23]. For example, the concept of the different viewpoints and views leads to redundancies in the models. This results in manually created, duplicate work and makes a consistency management necessary. In addition, the authors criticize the complex relationships between the different levels of abstraction. Any change to an artefact will have corresponding effects on one or more other artefacts on other levels. This leads to massive problems in round-trip engineering. The third point of criticism from Hailpern and Tarr is the additional complexity resulting from an increasing number of artefacts, their increasing number of relationships with each other and the necessary use of tools. This means that they see massive problems with future maintenance, troubleshooting or alteration of the created artefacts. Finally, they fear an additional problem in the diversity of modeling languages. The standardization of the UML should be an important basis for the success of MDD. However, with the powerful extension capabilities of the UML through the Meta Object Facility (MOF) [35], a variety of dialects have been created. This makes the semantically correct use of the UML difficult, also for tool vendors to provide supporting technologies.

Heijstek and Chaudron studied in [24] these factors as part of a large industrial software development project and confirm the factors mentioned. For example, a code generator is another application that needs to be developed, tested and maintained. This means initial effort and increased complexity in the tool use by dependencies. It is therefore not surprising that Singh and Sood in [43] also make the future use of MDD in industrial software development dependent on the fact that MDD must be fully integrated into a software development process. In addition, the tool support is a major success factor and the complexity of the modeling language with its additionally required knowledge is a potential barrier.

During this research, these assessments were confirmed by three case studies from practice. These were three projects from the business environment of insurance companies. Here it was also found that there is certainly potential for the promised increase in efficiency and quality. However, various problems in dealing with MDD could also be confirmed. In one case, the development of the "Domain Architecture" was discontinued due to the high effort and the project was developed traditionally. In another case, the consequences of the high complexity and dependencies between the artifacts are evident, which often leads to project delays. In addition, these experiences were also confirmed by case studies by IBM [8][11], ABB Robotics and Ericsson [48], Autoliv, Sectra and Saab Aerospace [17] and Motorola [6].

3. AGILITY AS A POSSIBLE SOLUTION

An iterative approach in the development of the "Domain Architecture" as well as a stronger interaction with the actual application development was sketched in a case study (see above) as a possible solution. As a result, parts of the "Domain Architecture" would be available for application development earlier. It would also contribute to the exchange of experience between application developers and MDD infrastructure providers. In addition, application development is enabled to achieve results for end customers at an early stage. These are properties that are already well known in agile software development and are well established in the industry. Methods such as eXtreme Programming [7], Scrum [15][41][42], Crystal [13][14], Adaptive Software Development [25], DSDM [46][47], or Feature Driven Development [3][12][37] have been established in the practice of software development [38] and have also been adapted to large projects by methods such as LeSS [30].

However, before we can think about using agile concepts in model-driven software development, we must first study the extent to which agile techniques for modeling exist. In practice, the agile principles are often misinterpreted by many developers as an order to code directly and not to document (or to model). Scott Ambler and Mark Lines, however, have defined the concept of agile modeling in [4] and described some agile modeling techniques such as “Model storming”, “Iteration modeling” or “Architecture envisioning” in [1].

There are also some agile process models that include modeling as a phase or work step, but do not fully implement the model-driven approach with its support of automated artefact creation. They often see the modeling "only" as a means of communicating with the customer and describing the functional requirements. These process models include:

(a) Agile Model Driven Development (AMDD): Scott Ambler describes an approach to integrate modeling in agile software development in [2]. However, Ambler places the focus on creating models with a minimum of effort and keeping them as simple as possible. He only wants to identify the most necessary requirements. This includes requirements for architecture as well as the basic functional requirements. However, this approach shows that AMDD is not a model-driven approach in terms of MDA or MDD and the use of their concepts. The use of tools for modeling and the automated transformation into various other artefacts is not applied. Instead, we must speak of a model-based approach.

(b) Feature-Driven Development (FDD): FDD was first described by Peter Coad et al. in [12] as a lean method for software development. The method provides the notion of "features" in the center of development. A "feature" is defined as a property of an application that is useful in the eyes of the customer. Unlike other agile process models in Feature-Driven Development modeling is a defined activity in the process model. So already in the first step of the process, an overall model is created. The aim of this first step in the process is to achieve a common understanding of the content and scope of the system under development. Another major step in the process flow, which is supported by modeling, is the design of a feature. During a walk-through, the chief programmer is developing along with the feature team a refined model. The design of the feature is checked during the inspection. But FDD is not a process for implementing MDD approaches. On the one hand, the definition of an architecture is not provided at all in the development process; on the other hand, manual coding by programmers is the focus.

(c) MIDAS Framework: In their comparison of different MDA-based methods, Chitforoush et al. [10] and Parviainen et al. [39] mention the MIDAS framework. MIDAS should

support the agile development of web information systems. For this MIDAS uses UML as modelling language for the creation of the necessary PIMs and PSMs. In addition, MIDAS defines mapping rules for the transformation of models from PIM to PIM, PIM to PSM and PSM to PSM. However, unlike the other MDD methods MIDAS defines no concrete development process. In another paper, Caceres et al. describe the experiences they have had in a case study with the integration of agile practices and activities from XP in MIDAS [9]. According to the authors, it turned out to be positive, to develop the CIM (Computation Independent Model) as an early general vision of the future application. In addition, MIDAS uses various agile techniques such as pairwise development or continuous integration. Based on their case study, the authors conclude that it is important to identify the strengths of agile modelling, to guide developers in creating the models, and to make a breakdown of the different requirements.

Of the three presented process models, only MIDAS can actually establish a relationship with MDA and model-driven software development. However, here too there is criticism about a missing development process.

4. DEVELOPMENT OF AN AGILE APPROACH FOR MDD

As outlined in sections 2 and 3, there are various rudimentary approaches for a process model for model-driven development. MODA-TEL, MASTER, etc. are oriented more towards the classical approach, while MIDAS, for example, tries to implement agile concepts. In the following, an agile process model for model-driven software development is presented, which is based on these basic structures and fully reflects the aspects "Domain Architecture", "Development Process" and "Team / Roles". It is designed to support the agile development of small to medium business applications through model-driven development. To develop this process model, the common elements of the individual existing process models were first identified and abstracted, and then described using metamodels. The thus obtained metamodels allows the comparison of the individual methods and the identification of gaps. Metamodels were defined for the description of the process steps, the team roles and the artefacts arising in the process. Fig. 1 shows, for example, the identified artefacts in the model-driven development. Some examples of such artefacts include:

- the domain-specific language, which is defined by a metamodel and describes the elements of the problem domain; and
- a reference model that uses the defined DSL and
- the associated reference implementation, which enables the derivation of model-to-artefact (or model-to-code) transformations based on the application architecture.
- A generated prototype enables a verification of the transformation against the reference implementation.

There was an additional focus on the identification of the dependencies between the individual artefacts and the possibility to further develop and refine them iteratively and incrementally without taking too much influence on the dependent artefacts. The goal was to provide the "Domain Architecture" in its early stages for the development of the application. However, the "Domain Architecture" will be changed during the project by

- the identification of other non-functional requirements affecting the application architecture and architecture and

- the extension of the domain-specific language by additional language elements.

The affected artefacts must therefore depend exclusively on the resulting models and transformation rules and must never depend on manual extensions. In addition to the definition of a new process model (i.e. a further instantiation of the developed metamodel), it was also examined which agile working techniques appear to be suitable for this process model in dealing with models and how these can be integrated.

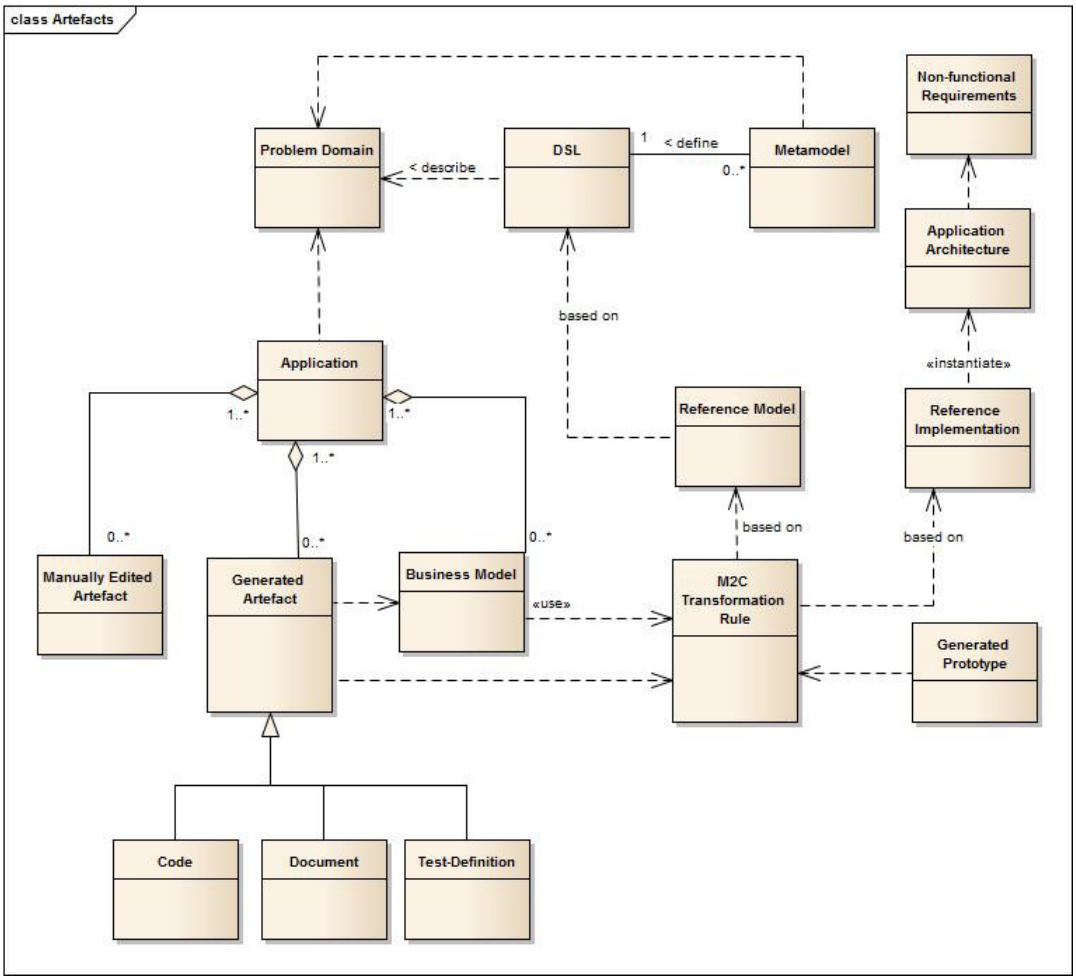

Figure 1: Artefacts in Model-Driven Projects.

5. THE AGILE MODEL-DRIVEN METHOD

As learned from the studies of Asadi and Ramsin [5] and Chitforoush et al. [10], the existing methodologies for MDD projects are incomplete and their description is imprecise. Essentially, they are based on traditional development processes, and the process framework by Chitforoush or the development lifecycle of Asadi and Ramsin do not regard agile aspects. Other approaches like AMDD [2] focus on the use of models in agile methods, but they do not consider MDD.

Based on the developed metamodels, the Agile-Model-Driven Method, "AMDM", was defined as a new process model for agile model-driven development. The individual aspects "Process", "Team" and "Domain Architecture" will be briefly outlined below. The terms often refer to

elements of Scrum or XP, since these approaches are widely used in practice [29] and their underlying principles are established. Here, for example, the iterations are also referred to as sprints (cf. SCRUM [15]).

5.1 Process

The development process of AMDM is divided into three types of sprints as well as a parallel optimization of the software architecture (see Fig. 2).

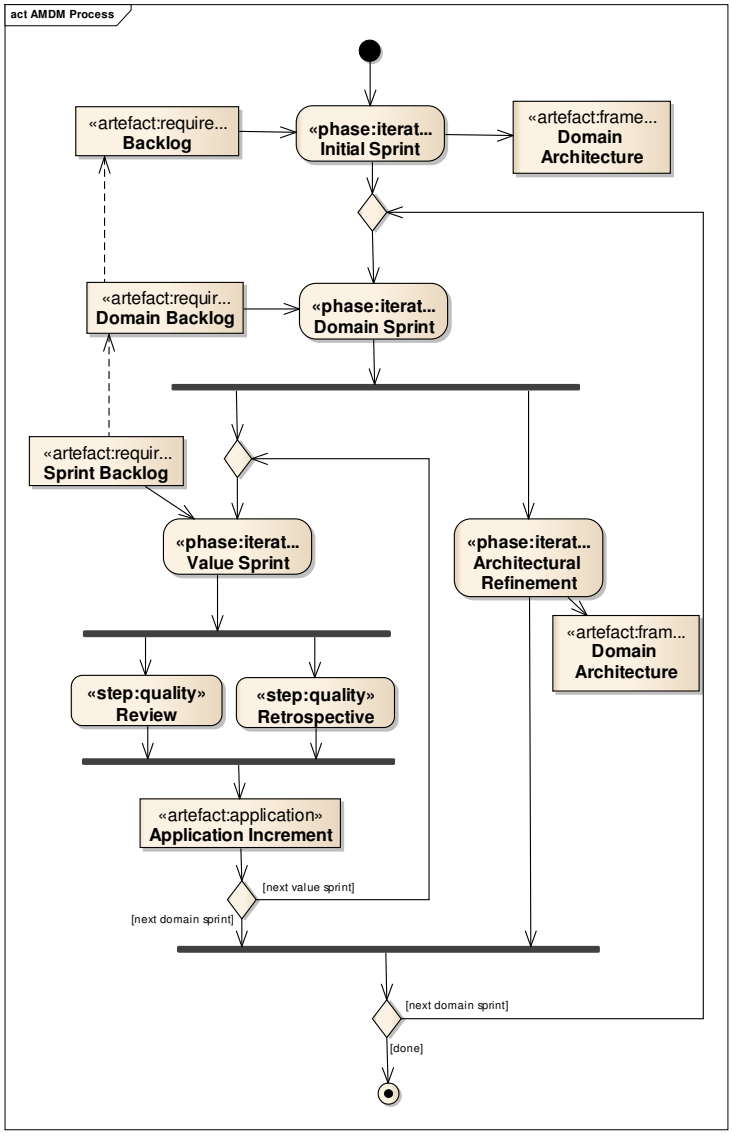

Figure 2: The Process in the Agile Model-Driven Method

The initial sprint defines the key requirements and the basics for the project. The backlog is created with the basic functional and non-functional requirements. Based on this, the rough application architecture is defined as a "big picture". In addition, a first version of the MDD infrastructure (Domain Architecture) with the DSL, the first transformation rules and the necessary frameworks as well as a minimal prototype (walking skeleton) are created here. The

initial sprint is followed by a domain sprint where the DSL is defined in relation to a selected subset of the problem domain. Within this domain sprint, the DSL and transformation rules for a specific aspect of the application are defined.

The domain sprint is followed by one or more value sprints in which the requirements from the backlog of the problem domain are modelled with the DSL, generated and implemented using the corresponding transformation rules. Following a value sprint, a review of the results as well as a retrospective of the process are given. This serves to continuously increase the quality of the results. Parallel to the development in value sprints, a refinement of the application architecture takes place considering the individual non-functional requirements or the findings from the retrospectives. A domain sprint with an adaptation of the DSL or the transformation rules follows upon a sequence of value sprints with the implementation of the technical requirements.

5.2 Team

Teamwork and communication is a main aspect in all agile process models. In AMDM the team works interdisciplinary. Each team member has his own know-how and contributes to the project success. In AMDM, the team is divided into three main groups:

(a) The first group knows and understands the functional requirements of the business application. On the one hand, there is the typical product owner, who represents the customers view in the project and names the requirements and prioritizes them. In addition, however, there are those project staff who model the problems using the domain-specific language. The concrete roles are named Product Owner and Business Analyst.

(b) The second group defines the architecture of the application and the domain architecture for the model-driven development. They define the domain-specific language formally by developing a meta-model and create the necessary rules for the model-to-artefact transformations. The roles are named Application Architect, Domain Architect, Domain Developer.

(c) The third group is the group named Application Developers who, according to the architecture specifications, manually supplement the generated source code with non-generateable functionality. During the Initial Sprint or Domain Sprint they also develop an application prototype as a “walking skeleton”.

5.3 Domain Architecture

The domain architecture deals with the definition of the domain-specific language (DSL) based on a metamodel as well as the definition of the necessary rules for the model-to-artefact transformation. The metamodel describes the elements of the domain-specific language. Depending on the chosen modeling language (e.g. UML) and diagram form (e.g. class diagram, activity diagram), it relies on the associated metamodels. In the case of UML, this is MOF [32][35][36]. The business analyst and domain architect as well as the domain developer are involved in the development of the metamodel. The business analyst describes the terms of the problem domain; the domain architect develops the corresponding metamodel and explains the meaning of the model elements to the domain developers. For the required model-to-artefact transformation, the knowledge of the application architecture is also necessary, additionally to the understanding of the DSL. For this purpose, the Domain Architect and the Application Architect work on the definition of the transformations. Because the process enables the iterative and incremental development of the application architecture parallel to the development process, it

ensures that not too many non-functional requirements need to be considered at the same time and that the complexity increases gradually.

6. APPROACHES TO EVALUATION AND VERIFICATION

In the previous paragraph the core aspects of the Agile-Model-Driven-Method were presented. However, it is difficult to verify the approach in a practicable way. One possibility to verify AMDM is the projection on the examined case studies. Initially, the following problem areas of model-driven development were mentioned:

(a) High initial effort: AMDM is an iterative and incremental development process. The domain architecture as the basis of the model-driven development is created successively. The waterfall-like approach to create all MDD artefacts at the beginning would have been avoided. In addition, AMDM is designed to produce results continuously. The development of a software application can be supported much earlier.

(b) Management mistrust: Confidence in the technology of model-driven development can be created by the early and continuous provision of usable results. Overall, the development of software becomes more efficient and of higher quality.

(c) High complexity and many dependencies: By focusing on the problem domain in the domain-specific language, AMDM avoids mixing technical with non-technical elements. The architecture is developed in parallel with the modeling and implementation of business requirements. Adjustments resulting from architectural changes do not affect the models, but only the transformation rules. And because of the evolutionary development of the architecture, their changes are always limited.

Additional suggestions from the other case studies of ABB Robotics and Ericsson, IBM, Motorola etc. were also considered and evaluated:

(a) AMDM involves business analysts, architects and developers in the development of the domain-specific language, the transformations and the implementation of the requirements. The work of the teams overlaps, so that the respective sub teams always have the necessary know-how.

(b) AMDM assumes that parts of the application must be encoded manually. This is done in tight cycles within a value sprint.

(c) The use of concepts of known and established agile methods reduces the inhibition threshold for the use of AMDM. The scepticism towards model-driven development can be counteracted by the early availability of partial results.

(d) AMDM also causes additional costs by defining domain-specific language and transformation rules. However, the early provision of applicable partial results at an early stage will bring an early benefit. The quality of automated application development reduces future error analysis costs. In addition, the architecture is being further developed in an evolutionary and iterative manner. And only as far as it's necessary. This also eliminates unnecessary costs for the creation of a bloated architecture and the resulting complexity of the model-to-code transformation.

(e) The case study by Elmqvist and Nadjim-Tehrani [17] confirms that model-driven development leads to cost savings in manual code implementation. However, they are

concerned about the availability of adequate tools for the entire development process, from specification to implementation. In the meantime, however, there are sufficient tools available, from UML modeling tools to generators, which are based on the relevant OMG standards. In AMDM, these tools can be used in an agile process. From the description of the requirements to the evolutionary development of the architecture and the definition of the domain-specific language.

(f) IBM's case studies [8][11] also confirm the potential of model-driven development. However, they assume that many manual changes to the models and the generated source code remain necessary. AMDM tries to keep the business models as stable as possible by basing them on the domain-specific language. However, IBM's studies also assume a pure MDA approach and a two-stage transformation from PIM to PSM to source code. AMDM does not follow this approach. AMDM uses a direct transformation from the commented PIM (based on DSL) directly into the source code.

(g) Likewise, criticisms from the Motorola study [6] are considered. In AMDM the intensive communication between architects, domain architects, business analysts and developers ensures that the knowledge about the models, transformations and the target architecture are evenly distributed in the team. This avoids implicit or explicit assumptions about implementations.

Finally, we have to assess whether AMDM is agile. It can be said:

(a) Customer Involvement: In AMDM, the customer's interests are represented by the product owner analogously to Scrum. He takes up new requirements, prioritizes them and leads them to the development process.

(b) Incremental delivery: This aspect has already been discussed before. Frequent partial deliveries are supported.

(c) People not process: AMDM also focuses on the communication and efficient collaboration of team members. However, due to the additional complexity of the model-driven development, the process is more strongly emphasized than in other agile approaches.

(d) Embrace change: Openness towards changes is also implemented in AMDM. Functional and non-functional changes can be included in the development at any time.

(e) Maintain simplicity: KISS (keep it simple, stupid) is a basic principle in the evolutionary development of software architecture, because the degree of complexity is growing in proportion to the requirements.

Looking at the sum of these criteria and comparing them to the Agile Model-Driven Method, this can be justly described as agile.

7. CONCLUSION AND FURTHER RESEARCH

The Agile Model-Driven Method combines agile working techniques with the development approach of model-driven development. For this, the elements of model-driven development were first identified and the existing limits and risks were considered. The criticism and scepticism of the model-driven development, which has often been expressed in practice, has also been analysed. Case studies from specific projects in the field of the author as well as case studies from

other branches were used for this purpose. Thus, potentials and criticisms of the model-driven development could be identified.

For the definition of an agile model-driven development methodology it was necessary to characterize the project phases of an MDD project, the involved roles and artefacts. For this purpose, common features of existing process models for model-driven development were identified and described based on a metamodel. On this basis and considering appropriate agile modeling techniques, the Agile Model-Driven Method has been defined. It enables an agile model-driven development of business applications in a continuous process, from the specification of the requirements to the implementation. It fulfills the criteria of an agile approach and is designed to minimize the problems and criticisms encountered in the investigated case studies.

AMDM is particularly suitable for the development of small to medium-sized business applications based on standardized components or services. The problem domain of these applications is limited and well structured so that the requirements and domain-specific language can be broken down into features. In addition, the experience with agile software development is widespread in this environment. The same applies to the principles of model-driven development based on MDA or MDD, even if they are rarely used. The limits of AMDM are reached when developing specific applications with very individual components. Here, manual implementation and optimization is still the method of choice. A possible additional difficulty is probably scaling to larger or distributed teams. The role of a mediator is recommended for this purpose, as described by Jutta Eckstein in [16].

Finally, the AMDM method must be applied in practice. Only through the use in small and medium-sized projects comparable to the showcases described above, further questions arise which can be examined.

Another open point, which can only be answered by appropriate experience, is the effort assessment and thus the planning of the sprints: How does the combination of modeling, generation and manual coding affect the effort required to implement a user story? Is the assumed timeframe of the typical two weeks for a sprint sufficient or even too long in this case?

The quality assurance of models can be a further focus of research. How can they be validated and verified? This is an independent field of research, but its results may be interesting for AMDM to conduct reviews.

And finally, another topic may be the classification of domain-specific languages. These are increasingly defined as text-based domain-specific modeling languages. Here an investigation of the different types of DSLs with an assessment of the suitability for different problems would be helpful.

REFERENCES

[1] Ambler, S., Jeffries, R. (2002). *Agile modeling. Effective practices for eXtreme programming and the Unified Process.* New York, NY: Wiley.

[2] Ambler, S., (2004). *THE OBJECT PRIMER. Agile model-driven development with UML 2.0*. New York: Cambridge University Press, 3rd Edition.

[3] Ambler, S. (2005). Feature Driven Development (FDD) and Agile Modeling. [online] http://www.agilemodeling.com/essays/fdd.htm [08/10/2017].

[4] Ambler, S., Lines, M. (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, Boston: Pearson Education.

[5] Asadi, M.; Ramsin, R. (2008). MDA-Based Methodologies: An Analytical Survey. In: Ina Schieferdecker, Alan Hartman (Eds.): *Model Driven Architecture – Foundations and Applications:* Springer Berlin / Heidelberg (Lecture Notes in Computer Science, vol 5095), pp. 419–431.

[6] Baker, P., Loh, S., Weil, F. (2005). Model-Driven Engineering in a Large Industrial Context - Motorola Case Study. In: *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005*: Springer, Berlin / Heidelberg (Lecture Notes in Computer Science, vol. 3713), pp. 476 – 491.

[7] Beck, K. (2003). *Extreme programming explained. Embrace change*. Boston: Addison-Wesley, 8th print.

[8] Brown, A., Conallen, J., Tropeano, D. (2005). Practical Insights into Model-Driven Architecture: Lessons from the Design and Use of an MDA Toolkit. In: Beydeda, S., Book, M., Gruhn, V. (Eds.): *Model-Driven Software Development.* Berlin, Heidelberg: Springer, 1st ed., pp. 403–431.

[9] Cáceres, P., Diaz, F., Marcos, E. (2004). Integrating an Agile Process in a Model Driven Architecture. In: GI *Jahrestagung 2004*, pp. 265–270.

[10] Chitforoush, F., Yazdandoost, M., Ramsin, R. (2007). Methodology Support for the Model Driven Architecture. In: *Software Engineering Conference, 2007.* APSEC 2007. 14th Asia-Pacific, pp. 454–461.

[11] Chowdhary, P. et al. (2006). Model Driven Development for Business Performance Management. In: IBM *Systems Journal* (Vol. 45, No 3), pp. 587–605.

[12] Coad, P., Lefebvre, E., Luca, E. de (1999). *Java modeling in color with UML. Enterprise components and process*. Upper Saddle River, NJ: Prentice Hall PTR.

[13] Cockburn, A. (2001). *Agile Software Development*. Reading, Massachusetts: Addison-Wesley.

[14] Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams.* Boston: Addison-Wesley.

[15] Cohn, M. (2010). *Succeeding with Agile. Software development using Scrum*. Upper Saddle River, NJ: Addison-Wesley.

[16] Eckstein, J. (2010). *Agile software development with distributed teams. Staying agile in a global world.* New York: Dorset House Pub.

[17] Elmqvist, J., Nadim-Tehrani, S. (2005). Intents and Upgrades in Component-Based High-Assurance Systems. In: Beydeda, S., Book, M., Gruhn, V. (Eds.): *Model-Driven Software Development*. Berlin, Heidelberg: Springer, 1st ed., pp. 289–303.

[18] Frankel, D. (2003). *Model driven architecture. Applying MDA to enterprise computing*. Indianapolis: Wiley (OMG Press).

[19] Gavras, A., Belaunde, M., Pires, L., Almeida, J. (2004). Towards an MDA-Based Development Methodology. In: Oquendo, F., Warboys, B., Morrison, R. (Eds.): *Software Architecture*: Springer Berlin / Heidelberg (Lecture Notes in Computer Science, vol. 3047), pp. 230–240.

[20] Gervais, M.-P. (2002). Towards an MDA-Oriented Methodology. In: *Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*. Washington, DC, USA: IEEE Computer Society (COMPSAC '02), pp. 265-270.

[21] Gervais, M.-P. (2003). ODAC: An Agent-Oriented Methodology Based on ODP. In: *Autonomous Agents and Multi-Agent Systems* 7, pp. 199–228.

[22] Guelfi, N., Razavi, R., Romanovsky, A., Vandenbergh, S. (2004). DRIP Catalyst: An MDE/MDA Method for Fault-tolerant Distributed Software Families Development. In: *OOPSLA and GPCE Workshop on Best Practices for Model Driven Software Development.*

[23] Hailpern, B., Tarr, P. (2006). Model-driven development: the good, the bad, and the ugly. In: *IBM Systems Journal* 45, pp. 451-461.

[24] Heijstek, W., Chaudron, M. (2010). The Impact of Model Driven Development on the Software Architecture Process. In: *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pp. 333–341.

[25] Highsmith, J. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House.

[26] Hildebrand, T., Korthaus, A. (2004). A Model-Driven Approach to Business Software Engineering. In: *Proceedings of the 8th World Multi-Conference on Systemics. Cybernetics and Informatics*. Orlando, USA, pp. 74–79.

[27] Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S. (2011). Empirical assessment of MDE in Industry, ICSE 11, May 21-28, 2011, Waikiki, Honolulu, HI, USA. ACM, 2011.

[28] Kent, S. (2002). Model Driven Engineering. In: *Proceedings of the 3rd International Conference on Integrated Formal Methods*. London, UK, UK: Springer (IFM '02), pp. 286-298.

[29] Komus, A. (2014). *Status Quo Agile 2014. Study on success and forms of usage of agile methods.* University of Applied Sciences Koblenz. [online] http://www.hs-koblenz.de/en/rmc/fachbereiche/wirtschaft/forschung-projekte-weiterbildung/forschungsprojekte/status-quo-agile-en/ [10/09/2016]

[30] Larman, C. (2016). *Large-Scale Scrum: More with LeSS*. Boston: Addison-Wesley.

[31] Larrucea, X., Diez, A.G.; Mansell, J. (2004). Practical Model Driven Development Process. In: Akehurst, D.H. (Ed.): *Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations*. Canterbury, UK, 7th-8th September 2004. University of Kent, pp. 99–108.

[32] Miller, J., Mukerji, J. (Ed.) (2014). *MDA Guide Version 2.0*. Object Management Group (OMG). [online] http://www.omg.org/cgi-bin/doc?ormsc/14-06-01 [10/08/2017].

[33] Mohagheghi, P., Dehlen, V. (2008). Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In: Schieferdecker, I., Hartman, A. (Eds.): *Model Driven Architecture – Foundations and Applications*: Springer Berlin / Heidelberg (Lecture Notes in Computer Science, vol. 5095), pp. 432–443.

[34] Object Management Group (OMG) (2010). *The MDA Foundation Model*. [online] http://www.omg.org/cgi-bin/doc?ormsc/10-09-06 [10/08/2017].

[35] Object Management Group (OMG) (2011). *Meta-Object Facility (MOF) Specification, Version 2.4.1 (August 2011)*. [online] http://www.omg.org/spec/MOF/2.4.1 [20/10/2011].

[36] Object Management Group (OMG) (2001). *UML Profile for Enterprise Distributed Object Computing*. Document ptc /2001-12-04.

[37] Palmer, S.R., Felsing, J.M. (2002). *A Pracitcal Guide to Feature-Driven Development*. Englewood Cliffs, NJ: Prentice Hall.

[38] Parsons, D., Ryu, H., Lal, R. (2007). The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects. In: McMaster, T., Wastell, D., Ferneley, E., DeGross, J. (Eds.): *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda, vol. 235*: Springer Boston (IFIP International Federation for Information Processing), pp. 235–249.

[39] Parviainen, P., Takalo, J., Teppola, S., Tihinen, M. (2009). *Model-Driven Development. Processes and practices.* [online] http://www.vtt.fi/inf/pdf/workingpapers/2009/ W114.pdf. [08/12/2016]

[40] Schmidt, D.C. (2006). Model-Driven Engineering. In: *Computer 39 (2),* pp. 25–31.

[41] Schwaber, K. (2004). *Agile Project Management with Scrum*. Seattle: Microsoft Press.

[42] Schwaber, K., Beedle, M. (2002). *Agile Software Development with Scrum*. Englewood Cliffs, NJ: Prentice Hall.

[43] Singh, Y., Sood, M. (2009). Model Driven Architecture: A Perspective. In: *International Advance Computing Conference (IACC 2009).* Patiala, India, 6-7 March 2009. IEEE.

[44] Soo Dong Kim; Hyun Gi Min; Jin Sun Her; Soo Ho Chang (2005). DREAM: A Practical Product Line Engineering Using Model Driven Architecture. In: *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*, vol. 1, pp. 70–75.

[45] Stahl, T., Völter, M., Bettin, J., Czarnecki, K., Stockfleth, B. von (2006). *Model-Driven Software Development. Technology, Engineering, Management.* Chichester: Wiley.

[46] Stapleton, J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, UK: Pearson Education.

[47] Stapleton, J. (2003). *DSDM: Business Focused Development*. Harlow, UK: Pearson Education. 2nd ed.

[48] Staron, M. (2006). Adopting Model Driven Software Development in Industry - A Case Study at Two Companies. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (Eds.): *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006,* Genova, Italy, October 1-6, 2006, Proceedings: Springer (Lecture Notes in Computer Science, vol. 4199), pp. 57–72.

AUTHORS

Klaus Mairon is a part-time PhD student at the University of Plymouth and Furtwangen University (HFU). In his profession, Klaus Mairon is an IT consultant and software architect at msg systems ag as well as a lecturer at the Baden-Wuerttemberg Cooperative State University.

The co-authors Dr. Shirley Atkinson from the University of Plymouth, Prof. Dr. Martin Buchheit and Prof. Dr. Martin Knahl from Furtwangen University (HFU) supervise his doctoral thesis.