Agile Limitations and Model-Driven Opportunities for Software Development Projects

K.Mairon[1,2], M.Buchheit[2], M.Knahl[2], S.Atkinson[1], S.M.Furnell[1] and U.Schreier[2]

[1]Centre for Security, Communications and Network Research,
University of Plymouth, Plymouth, United Kingdom
[2]Furtwangen Research Node, Faculty of Business Information Systems
Hochschule Furtwangen University, Germany
e-mail: klaus.mairon@hs-furtwangen.de

Abstract

The development of business applications has become increasingly complex and cost-sensitive. Thus discussions about the appropriate software development process model and the possibility to increase efficiency are frequent. This paper summarizes the limitations of agile process models and analyses how the limitations can be overcome through the concepts of the Model-Driven Software Development. Finally there is an outlook to the further research with the intention to combine agile and model-driven concepts.

Keywords

Agile Software Development, Software Development Process Model, Model-Driven Software Development

1. Introduction

Today the development of business applications is influenced by increased project complexity, shortened development cycles and high expectations in quality (Baskerville & Pries-Heje, 2004). Rising costs in the software development are an additional motivation to improve the productivity by the choice of a suitable development process (Jones, 2008).

In the development of complex applications models are of great importance. Models reduce complexity by abstraction. Additionally models offer the possibility to build different views onto an application. If models are sufficiently formal they are suitable for the automated transformation into source code. For this reason an important acceleration and quality factor in the software development is attributed to the Model-Driven Software Development (Stahl and Völter, 2005). On the other hand Model-Driven Software Development requires quite high initial work for the definition of meta-models, domain-specific languages and transformation rules for the code generation process.

A different approach to improve productivity is the use of agile process models like Scrum, Extreme Programming (XP) or Feature Driven Development (FDD) (Lindval *et al.* 2004). For these process models an early production of source code and the

adjustment of executable partial results are important aspects of the process. The communication with the end user and the direct feedback are the most important success factors for a project and facilitate quick reactions on requirement changes (Eckstein, 2010). In agile methods modelling often plays a subordinated role. The requirements will be documented via "user stories" (XP) or "features" (Scrum, FDD). They are summarized either in Product- or Sprint-Backlogs (Scrum)(Cohn, 2005) or in Feature-Sets (FDD) (Coad *et al.* 1999). This doesn't mean that there is no documentation or modelling in the development process. But only FDD describes modelling as an explicit step in the development process.

In the development of large and in many cases complex business applications it is common practice to use more formal process models with strong administrative aspects, such as for example the Rational Unified Process (RUP) or the V-model. However, Eckstein describes in (Eckstein, 2004) that agile process models can be used in large projects instead of the heavyweight process models. This raises the question to what extend the usage of models and the Model-Driven Software Development can be integrated into the agile development process. In the heavyweight software development processes like RUP modelling is a substantial part of the process and the technique of the Model-Driven Software Development may be integrated into these processes well. But in the less formal process models like Scrum there is no specified approach to integrate Model-driven Software Development. Another challenge is if the development team is distributed to various locations.

The paper will discuss the limitations of agile process models and how Model-Driven Software Development can assist these processes to get a successful high-quality and maintainable overall result. Based on three weaknesses of agile process models is shown how these can be mitigated by typical MDSD-technologies (e.g. through the use of domain-specific languages, or refactoring at the architecture level). For this, the paper will outline the first approaches and further steps of a corresponding research project.

2. Limitations of Agile Development Processes

In (Ramesh *et al.* 2006) some challenges for the application of agile development processes in large (especially distributed) teams were identified. As an example there is the conflict between communication need and communication independence. Agile development processes are based on informational communication rather than detailed documentation. But in large projects with many team members there is a need for formal methods such as detailed specifications or architectural design to give the developers the information needed. Also in (Turk *et al.* 2002) the importance of face-to-face communication in projects is indicated as a limitation of agile processes for distributed teams.

In (Turk *et al.* 2002) the authors explain several limitations for agile processes. These are amongst others:

- No or limited/poor support for distributed development.
- No process support to identify reusable software components.
- Problems in refactoring large and complex software systems.

In the following these points are clarified.

2.1. Agile principles in distributed development projects

The so-called agile principles (Agile Alliance, 2001b) underpin the value system of the Agile Manifesto (Agile Alliance, 2001a). They give guidance on the implementation of an agile approach. However, principles such as "continuous delivery of valuable software" lead to a variety of challenges for distributed projects.

Therefore, the early and continuous delivery of software requires a stronger collaboration between all locations as in non-distributed projects. To build a software release across different geographic locations is more difficult than if the team members would sit together. The challenge is, not to accomplish several individual systems on the various sites but one coherent system.

Furthermore, it is very difficult to achieve a close cooperation between business people and developers. In addition to the spatial distance there are often also cultural differences, and large differences in time zones can complicate the cooperation further. Nevertheless, all project members must get a common understanding of the business requirements. In (Eckstein, 2010), the author describes different roles (e.g. the "traveller") to enhance the communication and collaboration in distributed projects.

2.2. Problems creating reusable software components

In agile processes the focus is on the development in short cycles and an early delivery of valuable software. This precludes developing generalized solutions (Turk *et al* 2005). But it is clear that reusability could yield long-term benefits. According to (Turk *et al.* 2002) the development of reusable software components or generalized solutions is best assigned in teams that are primarily engaged in the development of reusable artifacts.

(Turk *et al.* 2002) refers to a study (Basili and Rombach, 1991), after which it is best to separate the product development from the development of reusable software components. The development of reusable software components requires a special attention to the quality, because errors in these components are often of greater relevance. In fact it is desirable to develop reusable components in a timely manner, but after (Turk *et al.* 2002) it is not clear how agile methods can be adapted accordingly. A possible solution to this problem is discussed by (Hummel and Atkinson, 2007). The authors propose to integrate the identification of reusable components tightly to the test-driven development cycles.

2.3. Problems in refactoring large systems

Agile methods are based on the premise that good design is achieved through constant refactoring (Fowler, 1999). This cannot be sustained in large complex systems. The increasing dependencies between software components make the code refactoring over the entire application costly. At the same time it increases the risk of errors. (Turk *et al.* 2002) also refers to software in which functionality is so closely coupled and integrated that it isn't possible to develop the software incrementally. In these cases can also be developed iteratively, but the code parts that are created within an iteration will always be incomplete.

In agile projects Test Driven Development (TDD) is a well-proven method to reduce the risk of errors during the refactoring process. But, with the increasing complexity and the growing number of dependencies between components the effort for the maintenance of test cases increases too. Incomplete code parts will complicate this additionally.

3. Agility and Model-Driven Development

In below it is to be shown how it is possible to support the agile principles with Model-Driven Software Development (MDSD).

- *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."* The Model-Driven Software Development starts with the definition of the domain architecture and the derivation of the transformation rules. This means a certain lead-time before a first executable result can be delivered. However, (Stahl & Völter, 2005) recommend deriving the domain architecture from a prototype or a reference implementation. This prototype can be used as a first delivery to the customer and is already providing valuable feedback for further development. If the prototype is developed incrementally, the project will get feedback continuously, and the architects can develop the domain-architecture in parallel.

- *"Business people and developers must work together daily throughout the project."* The close cooperation between business people and developers can become more intensive by the modelling. When using a domain-specific language (DSL) this is still more strongly accentuated. Mistakes can be detected more quickly because the developer and the domain expert are talking at the same abstraction level. In (Ambler, 2002) the author argues that the communication between developers and business people is the primary reason for modelling and emphasizes the advantages of the model as a basis for reviews and feedback sessions.

- *"Continuous attention to technical excellence and good design enhances agility."* A frequent criticism at the Model-Driven Software Development is that the developers have less freedom for their own choices in the

development of the software. On the other hand high quality is guaranteed by the automated transformation from models into code and the standardized implementation (Stahl & Völter, 2005).

- *"Working software is the primary measure of progress."* As explained in (Stahl & Völter, 2005) the creation of executable software can be accelerated significantly, because of recurring tasks that are automated. The developers can focus on the implementation of business logic.

In addition, the MDSD helps to mitigate the limitations of agile software development processes.

3.1. Reusing domain artefacts

In the context of Model-Driven Software Development two aspects must be considered to support building reusable software components: the domain-architecture and the development of business logic.

In addition to the application architecture the domain architecture is an important artefact of the Model-Driven Software Development. According to (Stahl & Völter, 2005) the domain architecture is defined as the aggregation of the meta-model of a domain and a platform with the corresponding transformations and tools. The domain architecture defines the concepts that will be formally supported in the model and how those are mapped on the given platform.

The development of the domain architecture should be implemented in parallel to the application development. An essential part is the reference implementation from which the transformation-rules are derived. The reference implementation has a much higher relevance as a conventional prototype. Together with the reference model or reference design, it demonstrates the application and implementation of the domain modelling language.

The domain-architecture itself is a reuse of architectural elements. The development of the DSL and the derivation of the relevant transformation-rules assist the identification of reusable components and modules.

During the application development, the modelling may help the developer to focus on the business logic and the semantics. Because of the higher abstraction level it is easier to identify reusable business components.

3.2. Refactoring at an higher level (architectural refactoring)

Support for refactoring is one of the strengths of Model-Driven Software Development. Refactoring can be applied to models, platforms, transformation rules and the implemented code. Thus the Model-Driven Software Development facilitates the reaction to changes clearly.

Changes in business requirements can be adopted through the generation process very quickly and in a consistent way. For example: additional attributes in business classes can automatically be reflected in the user interface, in the database definition, and in all relevant data structures. Only the adaptation of the affected business logic has to be done via source code refactoring. This is the most common type of refactoring and is needed whenever new requirements affect existing code.

But another kind of changed requirements is significantly more complex to manage. These are changed technical requirements like adjustments of the architecture or the replacement of an underlying technical framework. In the model-driven approach, architectural changes can be performed at a central point: at the templates and the transformation rules. These changes are taking over for the whole application automatically.

3.3. Supporting agile and distributed projects

In the context of distributed development, it is difficult to decide what should be developed at what location. This is also in an agile project. Additionally there is the question of how to achieve a common understanding of the future application.

Section 3.1. describes the need of developing additional artefacts in the context of Model-Driven Software Development. But the creation of the domain architecture can be separated well from the development of the business logic and can be developed by a team located at a remote location without direct customer contact. At the same time the development of the reference implementation, the description of the reference design and associated programming model contributes to the general understanding of the application architecture.

(Ambler, 2004) argues that the quality of the requirements descriptions is enhanced with a domain specific modelling language. This applies to all teams in a distributed project environment that operate close to the customer. However, the additional abstraction by the domain specific modelling language is well suited to help all project members to get the required overall picture of the application.

(Eckstein, 2004) describes how agile approaches can be applied in large projects. On this basis, she describes in (Eckstein, 2010) the use of agile methods for distributed teams. According to the author, just the emphasis on communication in agile approaches is the essential advantage for working in large and distributed teams. The challenge for such projects is to achieve a common vision of the target system and mutual trust. While the usage of a domain specific modelling language supports the communication, the reference design and reference implementation provides transparency. This enables the team to reach this goal.

In this way, the Model-Driven Software Development can improve the limited support of agile methods for distributed development projects.

4. Conclusions and further research

According to (Parsons et al. 2007), almost 40% of the surveyed IT professionals use one or more agile methods in software development. Close cooperation with the customer and refactoring are commonly referred to as the agile techniques with the greatest benefit in terms of quality, productivity and satisfaction.

The close cooperation with the customer may be supported additionally through the Model-Driven Software Development and the use of domain-specific languages. The use of a common modelling language supports obtaining a shared vision of the software that has to be developed.

The agile technique of refactoring assists the continuous improvement and development towards the target architecture. The Model-Driven Software Development supports this effect and brings additional efficiency into the development. The code generation is also a guarantor for a unified and reproducible implementation and high quality. The Model-Driven Software Development can help to scale the agile techniques through the explicit separation of the development of the domain architecture and the application development.

The further research will be focused on the following aspects. First, the issue is examined what kind of modelling is suitable to enhance the communication with the users, without creating unnecessary formalism. For this, approaches for agile model-driven development are considered, as described in (Ambler, 2004). Additional information provides a case study of the Rey Juan Carlos University in Madrid, which took account of these approaches in their framework MIDAS (Cáceres *et al.* 2004). An agile process should be defined, that optimally integrates the modelling and provides the information sufficient for the model-driven development.

Another aspect attends to the process of the model-driven development and the relevant artefacts (e.g. the domain architecture). The goal is to define an agile process for the development of these artefacts. For this, best practice experiences in model-driven development like in (Baker *et al.* 2005) will be analyzed. In addition, a survey about the usage of MDSD as well as the adaptation of agile process models in practice will be done to get additional information about industrial experiences. For this primarily the German IT-market will be examined, which has suffered recently from particularly high and rising costs.

It is the intention to define and develop a framework to facilitate agile and model-driven software engineering. This includes essential procedures, methods and tools. The framework can be applied to the development, testing, operation or project management. It is envisaged that the main focus will be the adoption of the framework to combine agile and model-driven development and to derive a new software development process. The framework will be applied to a software development project to evaluate its usability and effectiveness.

5. References

Agile Alliance (2001a), „Manifesto for Agile Software Development“, www.agilealliance.org (Accessed 15 July 2010)

Agile Alliance (2001b), „Principles behind the Agile Manifesto“, www.agilealliance.org/principles.html (Accessed 15 July 2010)

Ambler, S. (2002), *Agile Modeling. Effective Practices for eXtreme Programming and the Unified Process*, New York: John Wiley & Sons, ISBN: 0-471-20282-7.

Ambler, S. (2004), *The Object Primer, 3rd Edition. Agile Model-Driven Development with UML 2.0*, New York: Cambridge University Press, ISBN: 978-0-521-54018-6.

Baker, P., Loh, S. & Weil, F. (2005), „Model-Driven Engineering in a Large Industrial Context - Motorola Case Study“, in: *Lecture Notes in Computer-Science. Model Driven Engineering Languages and Systems.*, 3713, pp. 476-491.

Baskerville, R. & Pries-Heje, J. (2004) „Short cycle time systems development“, in: *Information Systems Journal*, 14 (3), pp. 237-264.

Basili, V.R. and Rombach, H.D. (1991), ”Support for comprehensive reuse“, in: *Software Engineering Journal*. 6 (5), pp. 303-316.

Cáceres, P., Díaz, F. & Marcos, E. (2004) „Integrating an Agile Process in a Model Driven Architecture“, in *GI Jahrestagung 2004*. pp. 265-270.

Coad, P., Lefebvre, E. and DeLuca, E. (1999), *Java Modeling in Color with UML. Enterprise Components and Process*, Upper Saddle River: Prentice Hall International, ISBN: 978-0-130-11510-2.

Cohn, M. (2009), *Succeeding with Agile: Software Development using Scrum*, Amsterdam: Addison-Wesley Longman, ISBN: 978-0-321-57936-2.

Eckstein, J. (2004), *Agile Software Development in the Large - Diving Into the Deep*. New York: Dorset House, ISBN 978-0-932-63357-6.

Eckstein, J. (2010), *Agile Software Development with Distributed Teams: Staying Agile in a Global World*, New York: Dorset House, ISBN: 978-0-932-63371-2.

Fowler, M. (1999), *Refactoring: Improving the Design of Existing Code.* Amsterdam: Addison-Wesley Longman, ISBN: 978-0-201-48567-7

Hummel, O. and Atkinson, C. (2007), ”Supporting Agile Reuse Through Extreme Harvesting“, in: *Agile Processes in Software Engineering and Extreme Programming, 8th International Conference*. Como, Italy pp. 28-37.

Jones, C. (2008) *Applied Software Measurement: Global Analysis of Productivity and Quality* 3rd ed., New York: Mcgraw-Hill Professional, ISBN: 978-0-071-50244-3

Lindval, M., Muthig, D., Dagnino, A., Wallin, C., Kiefer, D., May, J. & Kähkönen, T. (2004) „Agile Software Development in Large Organizations“, in: *Computer*, 37 (12), pp. 26-34.

Parsons, D., Ryu, H.R. & Lal, R. (2007), "The Impact of Methods and Techniques on Outcomes from Agile Software Development Projects". In *Organisational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*. pp. 235-249.

Ramesh, B., Cao, L., Mohan, K. and Xu, P. (2006), "Can distributed software development be agile?", *Communications of the ACM*, 49 (10), pp. 41-46.

Stahl, T. & Völter, M. (2005), *Model-Driven Software Development. Technology, Engineering, Management*, Chichester: John Wiley & Sons, ISBN: 0-470-02570-0.

Turk, D., France, R. & Rumpe, B. (2002), "Limitations of Agile Software Processes", in: *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering*. Alghero, Italy, pp. 43-46.

Turk, D., France, R., Rumpe, B. (2005), "Assumptions Underlying Agile Software Development Processes", *Journal of Database Management*. 16 (4), pp. 62-87.