Stefanie Betz, Ulrich Reimer (Hrsg.): Modellierung 2016 Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 87

# The potentials of a code generator which faces the stress ratio of requirements engineering processes in agile development projects

David Kuhlen[1], Andreas Speck[2]

**Abstract:**

The agile software development method is common in many software development companies worldwide. The following paper approaches an universal problem of agile development projects in the whole software development branch of industry. Unfortunately, in many cases, the requirements are incomplete and the projects are under-specified. Imprecise requirements and specifications cause not only problems in the development, but they also result in inaccurate cost estimations. This leads to problems even in the development and in the requirements engineering. Moreover, the lack of clear specifications may lead to further customer requests at the development time. They might be realized at very high costs. This increases the time pressure for the developers. In order to realize the software in time, the developers lower the quality level of the code. A lack of quality causes technical debts which reduces the economic efficiency of the system considerably. A potential solution, proposed in the paper, is the use of code-generation systems. The use of a code generator could offer potentials to improve the processes of requirements engineering. The support of these generators enables the software developers to focus on the requirements engineering. As the generated code contributes directly to the sprint goal, this approach could be well accepted by the developers. The potentials of such a concept are investigated.

## 1 Introduction

In agile development projects the developers may decide to skip (or shorten) the process of a comprehensive requirements engineering [Sc13]. Without requirements engineering the software producer might save the effort and finish the project faster [Hr14]. The focus on activities which fulfil the sprint goal is typical for agile projects. However, requirements engineering is a long-term goal. It is not attractive, particularly in short iterations. Nevertheless, good requirements engineering methods help to reduce the error rate of a software system from 0,23 to 0,08 per function point [Hr14]. Increased time pressure enhances the risk of technical debts. The quality of the requirements engineering process results from its contribution to reduce technical debts. There emerges the trend to ignore the long-term profitability [Al12].

[1] Datenlotsen Informationssysteme GmbH, Technische Beratung, Beim Strohhause 27, 20095 Hamburg, David.Kuhlen@nordakademie.org

[2] Christian Albrechts Universität zu Kiel, Abteilung, Hermann-Rodewald-Straße 3, 24098 Kiel, aspe@informatik.uni-kiel.de

### 1.1 Research Question

Agile software development has to solve many challenges. The impact of code generation on these challenges should be examined. The paper should investigate the impact of the use of a code generator on the challenges of requirements engineering processes in agile software development. It tries to show how the universal problem of the whole software development branch of industry could be solved by the application of a code generator.

In the following sections it has to be examined which potentials a code generator offers, in order to solve the stress ratio of requirements engineering in agile development projects. At first, the stress ratio in agile projects has to be analysed. Secondly, the impact of a code generator on the stress ratio has to be assessed. At last, standards for a code generator should be formulated. These standards have to be fulfilled by a code generator in order to aid agile projects.

### 1.2 Related Work

This work bases on various researches in different fields. It includes research on technical debt, on code generation, on requirements engineering and on agile process models. In this section, just a few works will be described, which have a special impact on this paper.

[Al12] and [MS12] described that technical debt harms the long-term profitability of software development projects. Ultimately, the savings have a negative impact on the maintainability of a software. Like every debt, it has to be repaid with interest. Thus technical debt is similar to financial debt which shows its impact. In order to prevent technical debts, their financial consequences have to be transparent. Previous publications recommend documentation of technical debts in an own backlog, with its financial impact [Al12]. Unfortunately, this impact cannot be determined easily [KS14]. Therefore, this method might be hard to apply in practice.

Techniques of code generation have been investigated intensively. An example for a technique, which expresses specified procedures graphically, is the *Business Application Modeler* (BAM). BAM allows checking business process models. The graphical tool bases on the *Eclipse Graphical Editing Framework* (GEF) [Sp11]. A code generator can consist of the combination of a *Domain Specific Language* (DSL) and a possibility for graphical modelling. The *Domain Specific Language* enables the code generation. The graphics editor allows the generation of corresponding diagrams. This generator could be realized by the combination of *Eclipse Graphiti* and *Xtext*. To support the requirements engineering by a code generator it is important to link the generator with the domain. Pulvermüller and Speck describe the *XOpt* generation concept. In this concept, a hierarchical structure consists of generic operators, which are domain-independent. For the adaptation to a specific domain, domain-dependent operators are refined [PS04].

The field of process analysis of software development approaches is topic of [KS15]. Kuhlen and Speck consider the phase of business process modelling as substantial. They

explained software development process to vary by the selected development approach. Because requirements were considered as main cost drivers during the development, the model, which describes the process of development, could be used to calculate the average execution costs.

## 2 Challenges of agile process models

The agile methodology influences the entire project. A reduction in the phase of conception is often justified by the fact that requirements are changing continuously. This shortens several aspects of the process of requirements engineering. [Hr14] explains one to three per cent requirements changes per month!

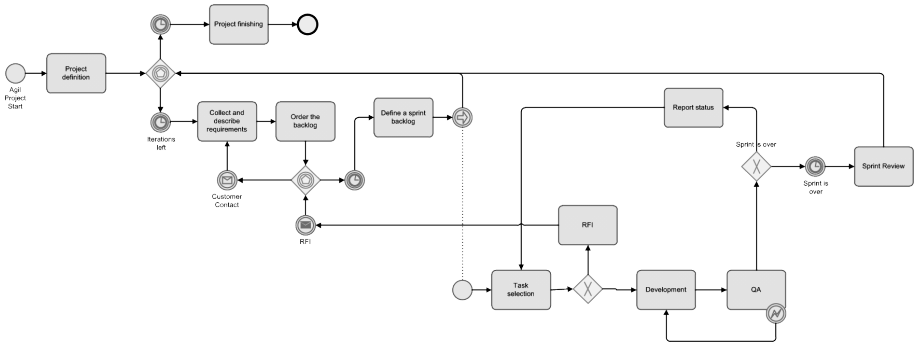


Abb. 1: Model of an agile development process inclusive requirements engineering activities

In Figure 1 the model of an agile development process is illustrated. A special emphasis is put on the requirement engineering activities in this model. As displayed in the diagram, the process consists of several activities which are arranged in a cycle. As explained by [KS15], this cyclic repetition during the requirements increases the costs of the whole process.

"Embrace the Change" leads to requirements that are rarely analysed. An agile process gives developers and customers a certain amount of freedom in terms of content design of software requirements. However if the effort of a requirement is not estimated correctly, the effort for a successful implementation will be difficult to assess at the end. In order to keep the project deadline, software developers partially reduce the quality of implementation. These savings act primarily on those aspects, which have not been described in any design requirement scarce. In the process, this leads to **technical debts**.

### 2.1 Requirements engineering in agile development projects

In agile development projects, the time budget of requirements engineering is often reduced. This is legitimized by the fact that requirements change rapidly in any case. Therefore, it is common to perform the requirements analysis only as it is required for the control

of the project. For those reason requirements descriptions do not have to contain many details of the specification. However, this leads to the risk that important details are missing before development could start (see cycle in Figure 1). The agile process model does not ensure that a written concept will be formulated between developers and customers. In order to reduce the time used for the analysis of the details, representatives of the customers will be integrated in the development process. It is assumed that the customer representatives can resolve ambiguities, resulting from the limited requirement conception during the software development process.

### 2.2 Technical debts and quality of code in agile development projects

Technical debts are savings in the production of software which deteriorate the quality of the service provided [Al12]. Typically, it is caused by the use of "short-cuts", thus save development time. When developers avoid the use of "best practices" or conventions, they can quickly find out what their requirements are. Many developers also avoid documentation in order to win speed [Ka12].

The use of agile software development often results in savings of time and money. This decreases the process costs. It is typical in agile development projects (where the circumference is less rigidly formulated) to regard the function set to be implemented as flexible in contrast to the firm constant of development time.

If developers are developing more and more functionality in a defined period of time, then this pressure on costs misleads the building up of technical debts [MS12]. In agile software development projects, technical debts can occur any time. Technical liabilities are accepted as inevitable and periodic. Therefore, iterations for their solution become scheduled [Al12]. The scheduling of such "bug fix" iterations is not optimal.

These iterations lead to the parts of the request, which become implemented in a later iteration. Better, requirements should be implemented undivided and technical debts would consequently not arise [Al12].

### 2.3 Effort estimation in agile projects

When using traditional development methods, (e.g. the waterfall model) it was not possible to define requirements completely in advance [Ar14]. As the requirements always change, a new development paradigm developed - the agile process model. Agility is often equated with "unpredictability" in software development. Instead, it focuses on flexibility and responsiveness to market changes [Ec13]. The description of this flexible process in formal terms of a process model is hard, but necessary to analyse its performance.

Many agile methods focus on the development of code without waiting for formal requirements analysis [CR08]. A comprehensive elaboration of concepts is not seen as an advantage. This leads to inaccurate goals (which cannot be measured) at the beginning of a sprint.

A cost analysis is a regular part of the requirements engineering. This analysis defines a budget, which must not be exceeded by the implementation. It has to be large enough to prevent the formation of technical debts. The budget must therefore not be too large, otherwise it can lead to the risk that additional features are developed which did not belong to the first scope. This risk can increase because of the communication with the customers.

The execution of a comprehensive requirements engineering for projects with high complexity and duration is criticized as being impossible [Ec13]. However, most of the development projects have to pass a business approval process, which assesses this viability. To obtain estimation rapidly a technical expert analyses the project quickly [Ar12]. Estimations, which are created without a reasonable investigation, will be proven wrong which harms the economics of the project. To increase its quality, those who are responsible for the implementation often perform the estimation. They are liable for the accuracy of the estimation [Al12].

## 3 Potentials of a code generator to solve the stress ratio

The success of a software development depends on four variables: time, functionality, resources and technical debts [Al12]. A process of a code generator has a positive effect on three of these variables: the time, the functionality and the technical debts. All these variables were affected by requirements engineering.

### 3.1 Generation leads to better estimations

A code generator shifts effort in development to the requirements engineering. It enables a process where larger parts of the software become configured instead of developed by the formulation of source code. This could increase the overall performance of the development process.

The application of a code generator leads to a formal procedure. The formal procedure ensures that all necessary steps are taken and all details are collected which are important in order to perform a valid estimation.

An improvement in the cost estimation is a key element in the success of a development project. Although every estimation inherent uncertainty about the future, the companies which can create better estimations are able to compete better on the market than others [Ca11].

### 3.2 Impact on Requirements Engineering

The use of a code generator changes the process of software development. As mentioned above, a code generator has to replace development time for the requirements engineering.

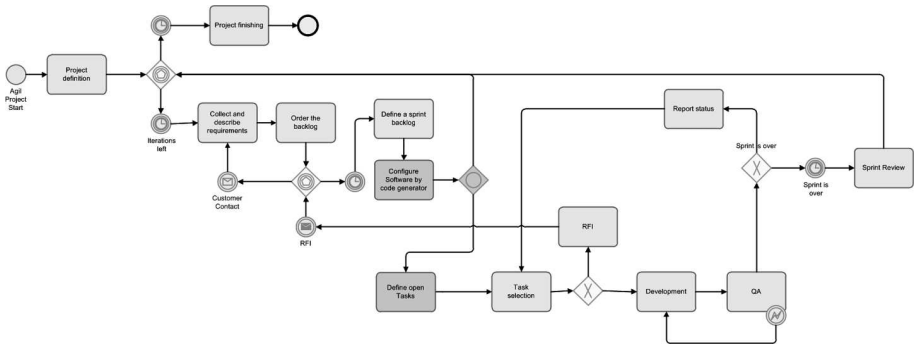


Abb. 2: Model of an agile development process in which a code generator is used

Therefore, the time of requirements engineering has to donate more than just investing the time in the development of a new functionality.

Figure 2 illustrates the process of software development which uses a code generator. This process is similar to the development process in Figure 1. The differences between the models are highlighted in green colour in the diagram. As displayed, the integration of a code generator in the development process allows building a required functionality without the need of developing a new functionality in a common way. However if it is not possible to fulfil a requirement by the use of the code generator, the classic development part of the process starts. Therefore, the integration of a code generator leads to a gateway and a probability that validates the execution of classic development.

By using a code generator, business requirements could be expressed in a high-level language (like *UML*). The expression of these business objects leads to a source code, which fulfils requirements partially, or complete (depending on the complexity and the possibilities of the generator). With this procedure, the development team has to focus on the requirements first.

A valid requirements engineering can be the basis of effort estimation. The quality of the estimations can be improved when more time is devoted to the analysis of the requirements engineering [Ar12]. However, customers will not demand less functionality of software vendors [Al12].

Finally, a code generator enables new ways of traceability. The traceability of requirements to source codes is especially relevant for subsequent changes of the requirements [Hr14].

### 3.3 Impact on technical debt

Generated source would regard to the given conventions and the generator would not take "short cuts". This would ensure that relevant models could be understood at any time. The maintainability of software will also be easier if bug fixes have not been incorporated [MS12]. If errors become fixed in the models, the correction can be realised through a

new code generation. So the quality of source would not become injured after multiple corrections. This could increase the process performance in long terms.

# 4 Standards for a code generator

Concentration on the essentials has a central meaning in agile software development. To reduce technical debts, requirements engineering has to take an important role. Therefore, unattractive, time-consuming and perhaps apparently unnecessary activities need to contribute to the fulfilment of the sprint goal directly. The use of a code generator can make this possible.

## 4.1 Modelling procedures in the requirements engineering

It is possible to express requirements in various ways. Only a bounded subset of these requirements engineering methods are used in agile development projects. Thus, requirements analysis should provide a basic understanding about the customers needs [CR08]. The methods of the classic requirements engineering include the creation of state diagrams and activity diagrams. These methods describe the process of functionality in formal terms. In contrast, in agile projects less formal and detailed methods are used to describe requirements.

In agile requirements engineering, there are essentially six different methods used to investigate more details of requirements [CR08]: (1) Direct communication with the customer, (2) Cyclic repetitions of small analysis, (3) Prioritization of requirements, (4) Scheduling changes in requirements, (5) Prototyping, (6) Test-Driven Development.

The cyclic repetition of the requirement analysis helps to gain detailed information on the requirement [CR08]. This analysis takes place mostly at the beginning of iteration. It does not result in a fully formulated specification. A code generator has to fit into this process model. Therefore, it has to be possible to add contents iteratively, in the entire project. The design of the code generator needs to separate the generated source sharply from the source which was written manually.

## 4.2 Performance needs for a code generator

A code generator could use the requirement specification to produce a source, which meets the request. This should not lead to increasing efforts for the specification and the implementation. The description of algorithms has to be simple enough so that the process of the generator would not become a barrier. It also needs formal descriptions of the algorithm to provide a source code. Being formal is not easy at any time.

If the requirements analysis just provides general information, it would only be possible to generate objects on the same level of abstraction. In order to increase the benefit of code

generation, it is necessary to express the content of a required algorithm. Therefore, the result of functionality and its computation path has to be designed in the requirements engineering. Large descriptions are produced which demonstrate how complex it is to perform computations in concepts. This level of detail enriches the project. Software manufacturers want to make sure that the requirements of their customers are fulfilled. Therefore, they describe what has to be done.

An example of a solution could be realized by XML transformation languages (XSLT). Fötsch and Pulvermüller describe transformation languages by building new high-level operators on top of existing ones, based on the generic XML operator hierarchy concept. The higher the level of the operators is, the more similar they get to everyday languages. This improves the readability and maintenance of code [FP09].

### 4.3 Specification of the Generator

If developers implement a business function, they begin to define different business objects. An object could represent a data record which itself consists of different objects. Therefore, the developers build hierarchical structured objects. Each object could get its own business functions. After the developers have built the business objects, they can identify the business process that deals with these objects.

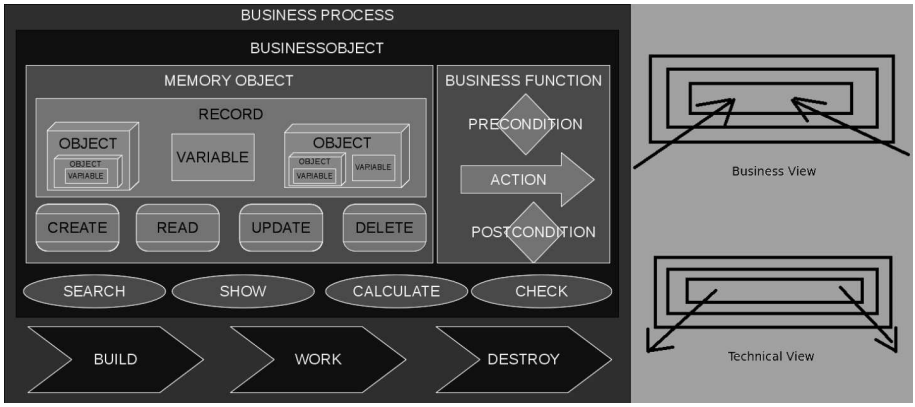


Abb. 3: Specification of the operator's hierarchy

The business view will be taken during the phase of requirements engineering. Requirements analysts start with thinking about the requirement by designing the overall process. The overall process is often described in a formal way. During the process, business objects are used (build, work, destroy). The use of these objects will be described by using more details. Therefore, different functions will be described. On the opposite, technical analysts (for example developers) start with defining memory objects. They look at the requirement from a technical perspective (technical view). After designing objects, they implement business functionality. Therefore, technical analysts start working on a deep technical level and go in the direction of a business level. On the other side, business designers proceed in the direction of the implementation (Cf. Figure 3).

The hierarchical structure of business objects can be presented in a class diagram. In development projects the requirements engineering often stops after describing the functions. The code generator has to support the analysis on different levels of abstraction.

The analysis should start on the highest level of abstraction, which is typical for the requirements engineering. From this point on, the analysis should go deeper so that more details of the requested functionality can be analysed. The different levels of operations are displayed in Figure 3. The requirements analysis goes from the outside to the inside of the layers. The development starts inside and goes out of the layers. By this specification, the code generator also defines a metric. The quality of results depends on the effort software producers want to invest in the requirements engineering.

## 5 Conclusion

A further step is the refinement of the generator systems and their validation in commercial software development operations. This could show the savings which are realized by the code generator. It has to be checked to which extend this efficient approach leads to a reduction of software development.

## Literaturverzeichnis

[Al12] Allman, E.: Manageing Technical Debt. COMMUNICATIONS OF THE ACM, 5(5):50–55, March 2012.

[Ar12] Armour, P., G.: The Business of Software The Goldilocks Estimate. COMMUNICATIONS OF THE ACM, 55(10):24–25, 2012.

[Ar14] Armour, P. G.: The Business of Software Estimation Is Not Evil. COMMUNICATIONS OF THE ACM, 57(1):42–43, 2014.

[Ca11] Cantor, Murray: Calculating and improving ROI in software and system programs. Communications of the ACM, 54(9):121–130, 2011.

[CR08] Cao, Lan; Ramesh, Balasubramaniam: Agile Requirements Engineering Practices: An empirical Study. Software, IEEE, 08(1):60–67, February 2008.

[Ec13] Eckstein, Dipl-Ing Jutta: Agilität – ein Baustein der dritten industriellen Revolution. HMD Praxis der Wirtschaftsinformatik, 50(2):77–83, 2013.

[FP09] Foetsch, Daniel; Pulvermueller, Elke: A Concept and Implementation of Higher-level XML Transformation Languages. Journal on Knowledge-Based Systems (KNOSYS), 22(3):186 – 194, April 2009.

[Hr14] Hruschka, Dr. Peter: Business Analysis und Requirements Engineering. Carl Hanser Verlag München, 2014.

[Ka12] Kamp, P.-H.: The Hyperdimensional Tar Pit. COMMUNICATIONS OF THE ACM, 55(3):52–53, 2012.

[KS14] Kuhlen, David; Speck, Andreas: Wertanalyseverfahren für Kundenanforderungen. In (Plödereder, Erhard; Grunske, Lars; Schneider, Eric; Ull, Dominik, eds): INFORMATIK 2014 Big Data - Komplexität meistern. volume P-232 of Lecture Notes in Informatics (LNI) - Proceedings, Gesellschaft für Informatik e.V. (GI), Stuttgart, pp. 2317 – 2322, September 2014. Thanks to Prof. Dr. Andreas Speck and Prof. Dr. Hinrich Schröder.

[KS15] Kuhlen, David; Speck, Andreas: Business process analysis by model checking. In (Ceravolo, Paolo; Rinderle-Ma, Stefanie, eds): 5th International Symposium on Data-Driven Process Discovery and Analysis SIMPDA 2015. Vienna, Austria, pp. 154–170, December 2015.

[MS12] McKeen, James D; Smith, Heather A: Effective Application Maintenance. Communication of the Association for Information Systems, 30(5):73–82, 2012.

[PS04] Pulvermüller, Elke; Speck, Andreas: XOpT-XML-based composition concept. In: Proceedings of the 3rd International Conference on New Software Methodologies, Tools, and Techniques (SoMeT'04). volume 111, Citeseer, Proc. of 3rd International Conference on Software Methodologies, Leipzig, Germany, pp. 249–262, September 2004. IOS Press, pages , 2004.

[Sc13] Schwaber, K.; Sutherland, J.: , Der Scrum Guide Der gültige Leitfaden für Scrum: Die Spielregeln, July 2013. Zuletzt Abgerufen am 06.09.2014.

[Sp11] Speck, Andreas; Feja, Sven; Witt, Sören; Pulvermuller, Elke; Schulz, Marcel: Formalizing business process specifications. Computer Science and Information Systems, 8(2):427–446, 2011.