Integrating Model-Driven Development Practices into Agile Process: Analyzing and Evaluating Software Evolution Aspects

Elton Figueiredo da Silva[1], Rita Suzana Pitangueira Maciel[1] and Ana Patrícia F. Magalhães[2,3]
[1]*PGCOMP Computação, UFBA Universidade Federal da Bahia, Salvador, Bahia, Brazil*
[2]*Exact and Earth Science Department, UNEB Universidade do Estado da Bahia, Salvador, Bahia, Brazil*
[3]*Post Graduated Program in Computing and Systems, Salvador University, Salvador, Brazil*

Keywords: Agile Methodology, Metamodeling in Agile Methods, Evolution of Software, Evolution of User Stories, Software Process Model, Model-Driven Development, Metaprocess and Metamodeling, MDD and SCRUM Integration.

Abstract: Software use is increasing in different areas of society, and new proposals of development processes have been presented to support this demand focusing on increase productivity and reduce time to market. In this context, some software development processes emphasize source code production, such as agile processes, others focus on modeling, such as Model-Driven Development (MDD). ScrumDDM is a hybrid metaprocess that integrates MDD practices into the SCRUM method aiming to specify software processes instances which models can be used in the agile development context. This paper presents a controlled experiment conducted to analyze the effectiveness of a ScrumDDM instance of its ability to support the agility and the evolutionary aspects of this software process. The results of the experiment showed that the models used in ScrumDDM gave extra support for evolution without compromising the development agility by executing a set of model transformations while preserving project code and documentation updated to support future software maintenance.

1 INTRODUCTION

Over the years, increasing demand for software products has been observed in different areas of our society. To support this demand and reduce time to market, high productivity in software development becomes essential. In this scenario, several software development approaches are being proposed, such as agile methods (Beck et al., 2001) and the Model-Driven Development (MDD) (OMG, 2014).

Agile development aims to fast delivery of products, which adds value to the client business, i.e., the software is incrementally developed and continuously delivered. This approach reduces documentation and considers code as the main artifact of the software development process (Tomás, 2009).

However, the possibility of building minimum software documentation can bring some disadvantages like the difficulty of software evolution and maintenance, and understanding about the software aspects by new development team members. Inadequate documentation of software developed by some agile methods makes hard the product maintenance, the changes traceability, and, consequently, proper product evolution (Heeager, 2012)

On the other hand, in the MDD approach, models are central artifacts in the development: models in high abstraction level are specified and (semi) automatically converted into models in lower abstraction levels until generating system source code (Brambilla et al., 2012). MDD favors productivity increase by automating the development process through the conversion of abstract models into other models until generating software code for different platforms. Additionally, it provides detailed documentation to assist maintenance (Brambilla et al., 2012). However, MDD may not be agile enough to assure fast client software delivery together with its documentation.

Hybrid processes proposals that integrate different software development approaches represent alternatives to increase the software life cycle (Alfraihi and Lano, 2017a).

In this direction, (Sales, 2017) proposes a metaprocess, named ScrumDDM, that integrates MDD practice into the SCRUM agile process. ScrumDDM allows hybrid processes instances to be specified for model usage during agile development. Thereby, models can be used for both project docu-

da Silva, E., Maciel, R. and MagalhÃčes, A.
Integrating Model-Driven Development Practices into Agile Process: Analyzing and Evaluating Software Evolution Aspects.
DOI: 10.5220/0009392501010110
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 2*, pages 101-110
ISBN: 978-989-758-423-7

mentation and as input artifacts in code generation for different platforms. ScrumDDM advances the state of the art because the hybridization proposal is not related to a specific domain, and also aspects related to software evolution were considered, for example, the documentation produced during software development.

As a metaprocess, ScrumDDM must be instantiated to a specific process to be used for the desired domain. Therefore, processes for two different domains were instantiated: (Braga, 2011) for Service Oriented Architecture(SOA) and Qualitas (Almeida et al., 2014) for MDD and TDD (Test Driven Development).

This paper presents a controlled experiment that measures the efficiency, software evolution, and agility aspects of ScrumDDM through the ArqProjProcess instance (Sales, 2017). The experiment results show that ScrumDDM is a promising approach as its process instance could guide developers to correct source code and proposed models without significant development time increase.

The rest of the paper is organized as following: section 2 introduces the concepts necessary for a better understanding of the work and section 3 presents the related works that integrate Agile Process and MDD. Section 4 presents the proposed ScrumDDM metaprocess. Section 5 presents the detailing of the controlled experiment developed. Finally, section 6 presents the conclusion and future work.

2 BACKGROUND

Several approaches deal with software process development. In this paper, we emphasize agile processes and model-driven development (MDD).

2.1 Agile Processes

Agile processes ideas were proposed in the early 2000 base in the agile manifest (Beck et al., 2001). In contrast to the software development methodologies at that time, agile processes promise software development faster for enabling better adaptability and flexibility to the new user's requests, allowing delivery within the stipulated time by the users.

Agile processes follow the incremental process model and are characterized by the valuing individuals rather than pre-defined processes and tools, constant running software deliveries instead of comprehensive documentation; valuing collaboration with the client, and fast response to change (Beck et al., 2001). An example of an agile process widely used is SCRUM (Schwaber and Sutherland, 2016).

The SCRUM differential is its focus on project management, which allows the integration of agile processes to other practices. The SCRUM framework allows:

- to describe software functionality from the user point of view and then generate software code;
- to specify user stories, describing necessary software features. User software requirements are organized according to their priority in a product backlog.
- to accommodate requirements changes of product backlog along the software development (Pressman and Maxim, 2016).

2.2 Model-Driven Development (MDD)

MDD is an approach that considers models as the main element in software development (Maciel et al., 2013). This approach tends to make software development easier by elevating the problem abstraction level by modeling and transforming the software requiremts, also documenting the system, stakeholder's attention concentrates on the software specification in detriment of code (Pressman and Maxim, 2016). MDD foments the increases of productivity by reusing models in the generation of multiplatform applications contributing to reducing the cost of development and also creating software product documentation.

3 RELATED WORKS

This section presents proposals for the integration of MDD and SCRUM approaches: (i) agility in the MDD process; and (ii) MDD practice into agile methods.

The works proposed by (Ambler, 2006) and (Mellor, 2004) propose the integration of Agile methods and MDD. Thus, they use MDD as the main process in addition to agile practices. The goal is to integrate agile processes to specify models iteratively, i.e., convert them, test, and modify in short, interactive, and incremental cycles. Models should be as complete as possible to be run by themselves.

The proposal of (Braga and Leal, 2013) integrates MDD practices during the agile process life cycle. Therefore, this proposal presents the Agile MDD concept, and so the practices of MDD as metamodeling, modeling, and use of transformations are added to the SCRUM process. Transformations are used to

automate tasks in the phases *Pregame, Game*, and *PostGame* of the scrum. Besides these already mentioned papers, others propose the integration of MDD practices to agile processes (Ambler, 2003), (Santos et al., 2018), (Mellor et al., 2005).

ScrumDDM uses an agile process SCRUM as the primary approach, adding practices MDD to software development. So it adopts the second strategy presented above. Its main difference from the other works is that ScrumDDM, as metaprocess, can be used for multiple domains (e.g., SOA, MDD, TDD, Informations Systems, Web Application) to support software maintenance and evolution. Neither of these papers covers software evolution aspects as an objective to be achieved in their development. Additionally, the metaprocess was evaluated by a controlled experiment.

4 ScrumDDM METAPROCESS

ScrumDDM is a metaprocess that integrates a set of practices from MDD to the agile method Scrum. It was developed trying not to deviate the principles and values of agile methodology, affording the development projects in different technologies, documentation, and making possible that the software development occurs quickly.

ScrumDDM is specified using the Software Process Engineering Metamodel (SPEM) (OMG and Notation, 2008) and implemented in Eclipse Process Framework Composer (EPF, 2014). Software Processes can be instantiated from ScrumDDM and adapted to different domains. As recommended in Scrum, ScrumDDM comprises three phases, *PreGame, Game*, and *PostGame*. The first two phases integrate Model-Driven Development practices while the *PostGame* phase does not hold any activity related to MDD; it follows the SCRUM standard.

Figure 1 illustrates the life cycle of ScrumDDM (Sales, 2017). After the software architecture and requirements are defined in *PreGame* phase, the phase *Game* starts to develop a new iteration of the software (called sprint) according to the requirements previously defined to it. Each sprint comprehends, besides activities originated from the agile methods *(Planning, Development, Testing, and Integration Meeting)*, the activities related to MDD, e.g. *Modeling and Run the Transformations*.

The main goal of the *PreGame* phase is to collect the requirements that will be used as input to the activities of the *Game* phase, as well as select / build the metamodels, models and transformations (Model-to-model (M2M) / Model-to-text (M2T)), which will be used in software development.

In the *Game* phase, models are converted into other models through the task *running and models M2M transforming*. The execution of this phase occurs iteratively through the models refinement until attending the goal defined in the *Vision* activity or until the model is converted into code.

In *PostGame* phase here are not any activities or tasks related to MDD. It focuses on finalizing the project and deliver it to the client.

Table 1: Summary of the ScrumDDM: Phases (PreGame, Game and PostGame), activities (Vision, Sprint, and Closure), subactivities (Development, Testing, and Integration) and tasks.

ScrumDDM Metaprocess		
PreGame	**Game**	**PostGame**
Activities		
Vision	**Sprint**	**Closing**
Subactivities		
Requirement Architecture	Development Test Integration	Deliverable approval Project conclusion
Scrum Tasks		
Sprint planning meeting Backlog product creation Version planning Write users' stories Decompose users' stories Write acceptance test Write use case Update use case	Sprint planning meeting Daily meeting Sprint review meeting Sprint's retrospective	
MDD Tasks		
Define: Metamodel Models	Metamodeling Modeling M2M transformation M2T transformation	
Artifacts' Set		
Domain Metamodels Proposed models Users' stories Use Cases	Models UML extensão mechanism Profile OCL expressions Code	Work finished

4.1 ArqProjProcess Instance

This section presents the ArqProjProcess (Sales, 2017), an MDD development process for SOA instantiated according to the ScrumDDM and used in our controlled experiment (Section 5).

ArqProjProcess comprises three phases *PreGame, Game*, and *PostGame* according to our metaprocess and it integrates into them the following activities: *Specify Trade Model, Analyze Services, Project Services, and Development*, from the ArqProjProcess original process (Braga, 2011).

The *PreGame* phase in Figure 2 of ScrumDDM instance (Sales, 2017) is initialized in *Vision* activity, with the *Sprint Planning Meeting* task. Its goal is to build the *Product Backlog* through the tasks, *Create, Estimate, and Prioritize Product Backlog*. The user stories and the use cases developed are the input artifacts for model building, such as the functionality model and the business information model. Even though you can use the *Update and Write Use Case* tasks.

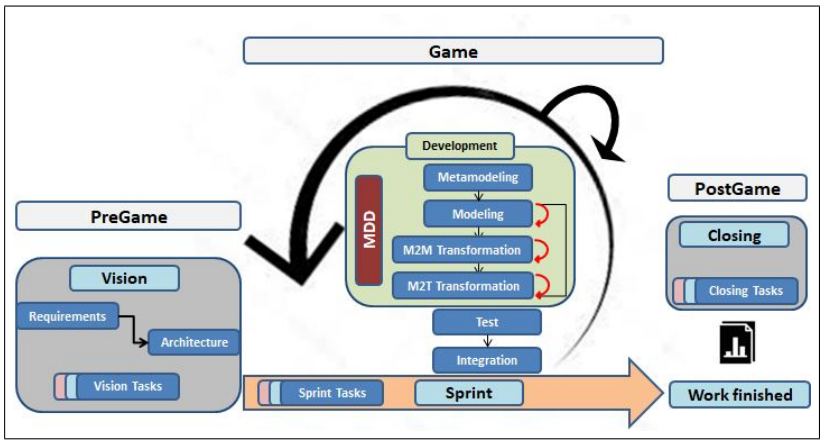

Figure 1: ScrumDDM Metaprocess Lifecycle (Adapted from Sales, 2017).

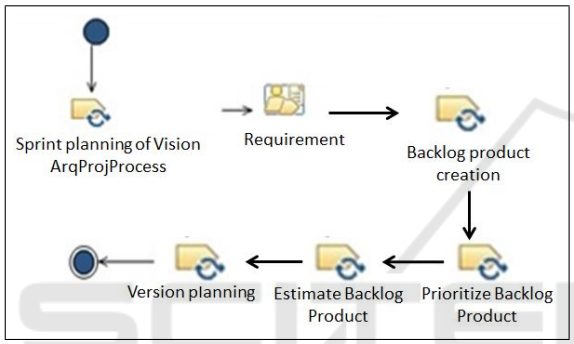

Figure 2: Activity Flow VisãoArqProjProcess (Adapted from Sales, 2017).

The *Game* phase in Figure 3 of ScrumDDM instance (Sales, 2017) starts with the task *Planning Meeting*. It comprises subtasks related to software *Development*, such as *Specify Business Model, Analyze Service* and *Design Service*(not showed in the figure). After these, the *Daily Meeting* task is triggered, and the *Sprint Review Meeting* will later be executed. Finally, the *Sprint Retrospective* task will be executed.

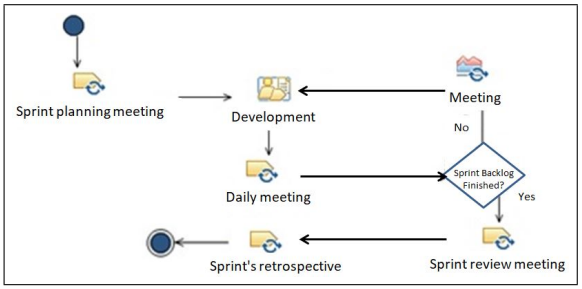

Figure 3: Activity Flow *Sprint* the phase of the Game (Adapted from Sales, 2017).

The *PostGame* phase in Figure 4 of ScrumDDM instance (Sales, 2017) aims to deliver the software to the client. This project and the generated documentation must comply with the needs specified by the client in the *Pregame* phase and developed during the *Game* phase. This phase consists of the *Closing* activity and comprises the tasks *Deliverables Approval* and *Project Conclusion*.

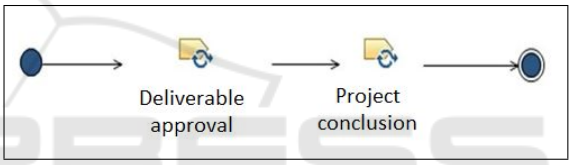

Figure 4: Activity Flow *Closing*. Activity Flow *Closing* (Adapted from Sales, 2017).

In the end of *PostGame* phase, Stakeholders perform the assessment according to the requirements developed and any nonconformities are adjusted before delivery. Therefore, at this stage, the activities and project closure are finalized.

5 ScrumDDM EVALUATION

Methods and processes for validation that involve humans are very challenging, and they should be carried out in phases. Each phase should be an evolution from the previous one. So, we used an incremental strategy to evaluate ScrumDDM metaprocess. Initially, (Sales, 2017) analyzed the hybridization capability between the agile method SCRUM and the MDD approach. This paper presents the second study, which analyzes aspects of the software evolution of ScrumDDM. For this, three attributes were defined in the controlled experiment: *correctness*, *completeness* of code as well as *timing* of development. So, we developed a new release of an existing software adding new user stories as well as evolving stories from the early releases. The evaluation was performed as a controlled experi-

ment and followed the phases: *Scope, Planning, Running*, and *Data Interpretation* and *Analysis* (Wohlin et al., 2012).

5.1 Scope: Goal

The controlled experiment aimed to analyze ScrumDDM about its capacity to evolve the software and its development agility while performing this evolution.

In order to assess the software evolution, we analyzed the source code generated using the ArqProjProcess process (ScrumDDM) compared to the source code developed in an ad-hoc way, using a standard release of Scrum, regarding correctness and completeness attributes. To analyze the agility of the evolutions, we used the timing attribute in minutes.

Figure 5 presents the experiment goal definition according to the template Goal Question Metric (GQM) (Caldiera and Rombach, 1994). To guide the experiment, we also defined the following research questions (RQ):

- **RQ1:** Do the artifacts generated by ScrumDDM influence in the amount of user stories correctly developed compared to the development performed directly in code? This question aims to investigate if ScrumDDM influenced the number of user stories correctly developed facing the same stories developed using only Scrum;
- **RQ2:** Do the artifacts generated by ScrumDDM influence the completeness of the user stories compared to the development performed directly in code? This question aims to investigate if ScrumDDM influenced in the amount of the user stories wholly developed compared to the same stories developed with Scrum;
- **RQ3:** Do the artifacts generated by ScrumDDM influence the time spent in developing the user stories compared to the development performed directly in code for the same users stories? This question aims to investigate if ScrumDDM influenced in the agility of the development when compared to the development performed by Scrum.

The metric used to measure correctness, completeness and timing works as follows: for **correctness**, the development of the evolution is considered correct if the number of stories implemented divided by the total number of stories evolved is equal to 1. For **completeness**, the development is considered complete if all the user stories are implemented. In this case we considered the completeness value as 1, otherwise it is 0. The **timing** is measured in minutes.

5.2 Planning: Context Definition

The controlled experiment was conducted in an academic environment composed by undergraduate and graduate students attending courses related to computer science and professionals from the software development area. In this experiment, besides knowledge in UML and Java language, it was also required from the participants, basic expertise in MDD. To guarantee that the students presented those requirements, we selected only those who were/are part of a scientific initiation project at the MDD area. Once the participants preselection was made, it was applied a survey so that this knowledge could be attested.

5.3 Experiment Planning

The design model used in this experiment covered *one factor*, the development method, and *two treatments*, the ScrumDDM approach, and the SCRUM approach. Thus, the experiment was performed by two groups, *ScrumDDM Group*, to develop software evolution through modeling in ScrumDDM; and *Controlled Group*, to perform the software evolution using only Scrum. For both groups a presentation with all the steps to be followed had been made.

The scenario of a bank conciliation software was utilized for both groups and proposed changes in the current user stories and new ones.

5.4 Planning: Variable Selection

In the performed experiment, the independent variable was constituted in the method used to implement stories and took two levels, the ScrumMMD approach and the SCRUM approach. The dependent variables were defined as the user stories correctness, completeness, and development timing.

5.5 Experiment Running

The study was conducted in an academic setting with undergraduate and graduate students and some technology professionals selected according to a predefined profile. A group of 25 people answered the survery which evoluated the profile of the candidates. Among them, 20 ones were selected and randomly divided in two groups of 10. The others did not have the minimum knowledge required to perform the study or were unavailable.

The experiment was performed between October and December 2018, and lasted, on average, 55 minutes of running time to Controlled Group and 38 minutes of running time to ScrumDDM Group.

Analyze the code developed during a Sprint.
for the purpose of evaluate software process maintainability and agility about *Correctness*, *Completeness* and *Development Time* attributes of defined user story in a specific Sprint.
from the point of view of the used Methods of development.
in context of students graduating and post-graduating and professionals from technology area.

Figure 5: Goal of the ScrumDDM metaprocess validation.

5.6 Data Collect

The data collected during the experiment was used as input for the experiment analysis. We used the scale in Table 2 to analyze the experiment data containing the expected results after the experiment conclusion by each participant.

Table 2: Template evaluation of results obtained after the conclusion of experiment by each participant.

Template						
A	B	C	D	E	F	G
6	4	4	4	4	8	10

In Table 2, the columns (A) to (G) represent the evolved or modified user stories, and the values attributed to these columns represent the number of changes expected to the development of each user story in the existing software. For example, based on the definition of the researcher, for the user story represented by column D (i.e., Perform bank conciliation), the participant should perform 4 (four) evolutions considering implementations in methods and attributes.

5.7 Data Interpretation and Analysis

The collected data was analyzed considering each attribute (metric), Group (ScrumDDM, Controlled) and user stories (requirements) according to what is presented in Table 3: the *minimum value observed (Min), the first quartile (Q1), median*, the *average*, the *third quartile* (Q3), the *maximum amount observed*(max), the *standard deviation* (SD) and the *coefficient of variation* (CV).

In Table 3, we observed the calculated descriptive measures based on the collected data, considering the defined and analyzed attributes of each group.

For **correctness attribute**, the average descriptive measure of correctly developed stories of each group points that the development performed by the ScrumDDM Group (median= average = 86%) had more correctly developed stories than the Controlled Group (median= 64%; average= 66%), matching the HP1 hypothesis (average correctness attribute percentage to ScrumDDM Group was larger than average correctness attribute percentage to Group Controlled). It is noticeable if we analyze the coefficient of variation (CV), a dispersion measure that covers both average and standard deviation, we observed that the stories developed by the ScrumDDM Group (13%) showed less variation than the ones developed by the Controlled Group (32%).

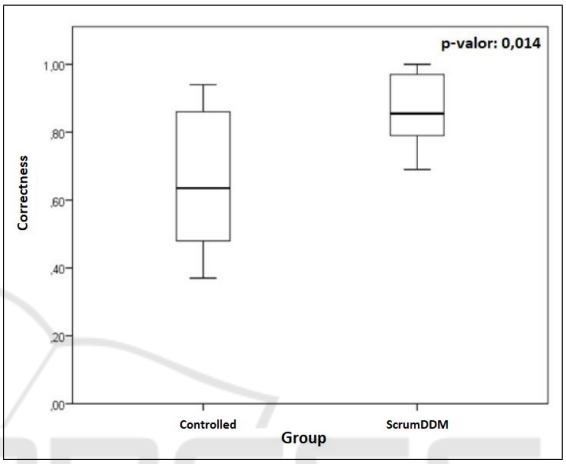

Figure 6: Comparison between ScrumDDM Group and the Controlled Group for attribute *correctness*.

This variation can also be seen in the boxplot chart in Figure 6 because the amplitude of the box for the Controlled Group was larger than the amplitude of the Group ScrumDDM. If we compare the median of the two groups, we observe that the development performed by the ScrumDDM Group was also superior to the one performed by the Controlled Group, confirming a better user stories evolution concision in the results of the Group ScrumDDM. In the Controlled Group, 75% of the participants got a number of hits below 86% compared to the hits obtained by ScrumDDM Group that exceeded this number.

For **completeness attribute** in Figure 7, the mean descriptive measure of the stories correctly developed

Table 3: Measures for the attributes *correctness, completeness and timing* about the participant Groups *(ScrumDDM and Controlled)*.

ATTRIBUTE	Group	Descriptive Measures Calculated Group							
		Min	Q1	Median	Average	Q3	Max	SD	CV%
CORRECTNESS	Controlled	0,37	48%	64%	66%	86%	0,94	21%	32%
	ScrumDDM	0,69	79%	86%	86%	97%	1,00	11%	13%
COMPLETENESS	Controlled	0,00	0%	36%	36%	57%	0,86	32%	89%
	ScrumDDM	0,29	57%	79%	72%	86%	1,00	24%	33%
TIMING (In minutes) Partial Documentation Update	Controlled	35	43	52	55	68	87	17	31%
	ScrumDDM	12	27	42	38	48	64	17	45%
TIMING (In minutes) Full Documentation Update	Controlled	35	43	52	55	68	87	17	31%
	ScrumDDM	20	43	59	59	77	87	21	36%

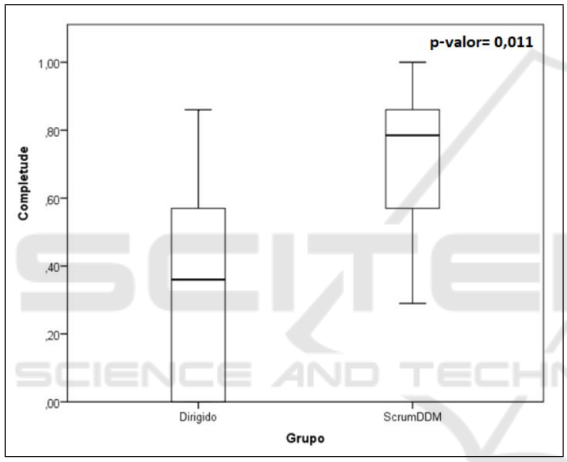

Figure 7: Comparison between ScrumDDM Group and the Controlled Group for the attribute *completeness*.

of each group points that the development performed by the ScrumDDM Group (median= 79%, average = 72%) had more correctly developed stories than the stories developed by the Controlled Group (median = average = 36%), matching the HP1 hypothesis (the average percentage of user stories developed in a complete way to Group ScrumMMD was larger than average completeness attribute percentage to Controlled Group). This is confirmed if we analyze the coefficient of variation (CV), we observe that the stores developed by ScrumDDM Group (33%) presented less variation than the ones produced by the Controlled Group (89%). The results obtained by ScrumDDM Group may point that these group results were more uniform than the results obtained by Controlled Group. Therefore, the lower variability of the ScrumDDM Group can evidence it presented more standardized behavior compared to the Controlled Group.

The **development timing attribute** was analyzed considering two aspects: (i) when partial documentation (Class Diagram) was updated and (ii) when all the documentation (System Architecture Diagram, Component Diagram, Sequencing Diagram, Architecture Diagram, Use Case Diagram) were updated, besides the documentation that generates the project source code.

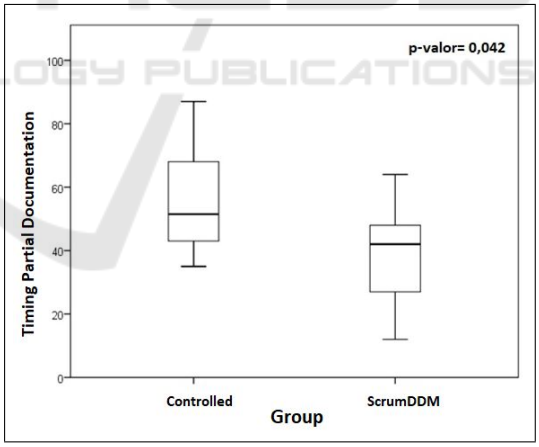

Figure 8: Evaluation of *timing* attribute, when the ScrumDDM Group partially updated the documentation.

Concerning only update of (i) documentation partial (Class Diagram) in Figure 8, the central tendency measures (median and average) for timing attribute of the users stories developed by each group points that the ScrumDDM Group (median= 42 minutes; average = 38 minutes) performed the evolution faster and more agile than the Controlled Group (median= 52; average= 55 minutes), matching HP2 hy-

pothesis. The timing taken in software evolution by ScrumDDM Group was shorter than the timing taken by the Controlled Group. However, if we analyze the coefficient of variation (CV) in Figure 8, we see that ScrumDDM Group development timing (43%) presents larger variability than the Controlled Group (31%). ScrumDDM Group presented asymmetry by the left, fact explained by the shorter timings obtained by half of the participants from this group that were more agile than (75%) the participants of the Controlled Group.

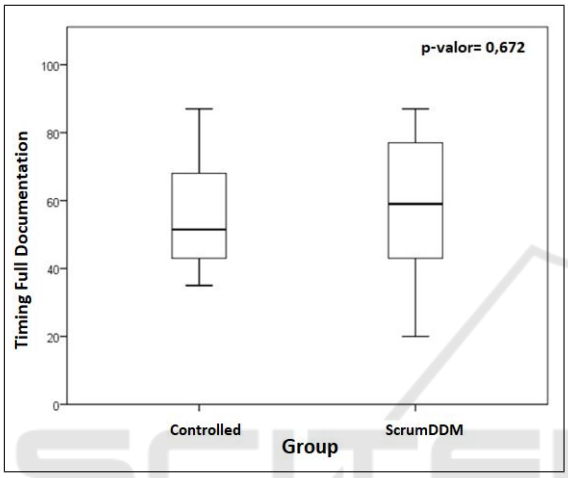

Figure 9: Evaluation of attribute *timing*, when the ScrumDDM Group has updated all documentation.

Concerning update of (ii) all the documentation (System Architecture Diagram, Component Diagram, Sequencing Diagram, Architecture Diagram, Use Case Diagram), besides the documentation that generates the project source code in Figure 9, the central tendency measures revealed that Controlled Group (median= 52; average = 55 minutes) presented shorter timing to solicited evolutions development than the development timing used by ScrumDDM Group (median = average = 59 minutes), matching the HP1 hypothesis. The development timing taken by ScrumDDM Group was longer than the development timing taken by Controlled Group. This group presented lower variability with a coefficient of variation equals 31%, while ScrumDDM Group presented a coefficient of variation equals 36%. We can see that even ScrumDDM Group having used more timing and varied more, it did not mean that this timing increase (4 min) influenced in the velocity of the metaprocess usage; we may consider that all project documentation, not only the source code, was updated.

5.8 Hypothesis Analysis

Aiming to verify the existence of a significant difference in the code developed by the two groups, the hypothesis test was run (Wohlin et al., 2012) through the *Student's t-test*.

The meaningfulness difference between ScrumDDM and Controlled group was analyzed trough Student t-test to independent samples. These statistic method assumptions, the distribution normality and the homogeneity of variations in both groups were analyzed, respectively, using Shapiro Wilk's test (SW) (Miot, 2017). For the (SW(10) ScrumDDM Group >0,932; p= 0,471; SW(10) Controlled Group>0.932; p= 0,471) and the average differences whose p-value in the test was less than or equal to 0,05 were considered statistically significant.

For correctness attribute, t-Student test, the differences observed (0,200) (what means that ScrumDDM Group obtained 20% more hits in face of Group Controlled) are statistically significant (t(18)= -2,72; p-value= 0,014; d= 0,54). The attribute correctness effect dimension to ScrumDDM Group is high (d=0,54) and according to IC a 95%] -0,35; -0,05[, the participants of ScrumDDM Group got a hit average between 0,05 (5%) and 0,35 (35%) more comparing to Controlled Group participants.

For completeness attribute, ScrumDDM Group obtained on average 72% (standard deviation: 24%)in the amount of completely evolved stories, while Controlled Group got on average 36% (standard deviation: 32%).According to t-Student test the differences observed (0,360)(what means that ScrumDDM Group obtained 36% of completeness compared to Controlled Group) are statistically significant t(18)= -2,83; p-value= 0,011; d= 0,55. The effect/confidence difference of completeness attribute to ScrumDDM Group is high (d=0,55) and according to IC at 95%] -0,62; -0,09[. This dimension means that ScrumDDM Group developed on average between (9%) and (62%) more stories completely developed than the participants of the Controlled Group.

For timing attribute, when updated/evolved *part* of documentation, the ScrumDDM Group obtained an average of 38 minutes (standard deviation: 17 minutes) while participants of Controlled Group run the evolution utilizing, on average, 55 minutes (standard deviation: 17 minutes). According to t-Student test the observed differences between both groups of average timings shows that this difference is statistically significant (t(18)= -2,18; p-value= 0,042; d=0,45). The timing attribute dimension effect to ScrumDDM Group is high (d=0,45) and according to IC at a 95%] -30,2; -0,6[, the ScrumDDM Group participants, to

part of the documentation, took on average is between 0,6 and 30,2 minutes less than participants of Controlled Group.

For timing attribute, when updated/evolved *all* documentation, ScrumDDM Group took on average 59 minutes (standard deviation: 21 minutes), while participants of the Controlled Group took 55 minutes on average (standard deviation: 17 minutes). According to t-Student test, the differences observed (4 minutes) between both groups for average timing are not statistically significant t(18)= 0,43; p-value= 0,672 d= 0,10). The timing attribute dimension effect to ScrumDDM Group is low (d=0,10) and according to IC at 95%] -14,3; 21,7[in minutes, the timing differences variation among the participants were not enough to highlight any difference between the groups.

5.9 Threats to Validity

To minimize possible threats about *Internal validity* a survey was applied for identify the level of knowledge of the participants about UML, and Java (Eclipse). The participants only had known about the experiment goal in the end of the development. In addition, the researcher accompanied both groups, and interaction between participants was not allowed. To minimize possible threats about *External validity* 20 (twenty) people were selected and randomly divided in two group (10 people each). The inability to experiment in the industry was minimized by selecting technology area professionals in both groups. To minimize possible threats about *Construction validity* several documents and transformations were made available to the participants.

6 CONCLUSION AND FUTURE WORKS

Combining MDD practices with agile development process aim to keep software development cycle time short and increase productivity and quality. By having short iterations, developers managed to build the system incrementally and early software verification, which contribute to saving time. Generating code and other artifacts from models (semi) automatically helped to speed up development process by reducing efforts in developing code. Requirements changes can be easily reflected in code since it is generated from model transformations (Alfraihi and Lano, 2017b). Additionally, integrate these aspects into a process metamodel helps software process reuse, customization, and domain independence

In the controlled experiment developed in this paper, the metaprocess ScrumDDM was analyzed about its capacity of evolving software and to maintain the agile methodology quickness characteristic while developing.Pre-existing software code already developed within a ScrumDDM instance, named ArqProjProcess was used as a base. A new version of the software was developed covering new user stories or user stories modifications.

The experiment performed pointed the capacity of evolving software and maintain the agility from the presented data. The group that used ScrumDDM metaprocess, for example, enacted the process more completely and produced the sofwtare artifacts more correctly than the Controlled group. Besides that, the evolutions occurred quickly, indicatong that the metaprocess maintained the agile methodology aspect. The usage of the metaprocess made possible to update the documentation artifacts and the software product source code. Additonally, by the end of the experiment running, it was shown that the metaprocess provided the software evolution, generating a correct code and with a more significant number of completely developed stories.

We intend to perform other studies with students and professionals from technology in the area, aiming to amplify the usage possibility of ScrumDDM. It would be interesting to test it for different domains and purposes. Besides that, even having technology area professionals as participants of the experiment, it would be relevant that ScrumDDM was evaluated in an industry scenario, the inclusion of a metamodeling engineer in the development process, aiming to build the MDD elements before developing the software, optimizing the work. Thus, we identify for future work:

- Instance ScrumDDM to other purposes and domains, different from the ones that were mentioned in this paper, like SOA and MDD, to make the evaluation/analysis of more aspects of instanced process possible.
- Reanalyze ScrumDDM in a real scenario, outside the academic environment.
- Analyze other software quality aspects. Besides the code and documentation generation aspects, it might be important to seek the software development process quality.

REFERENCES

Alfraihi, H. and Lano, K. (2017a). The integration of agile development and model driven development.

In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*, pages 451–458. SCITEPRESS-Science and Technology Publications, Lda.

Alfraihi, H. and Lano, K. (2017b). Practical aspects of the integration of agile development and model-driven development: An exploratory study. In *MODELS (Satellite Events)*, pages 399–404.

Almeida, C. C. d. J. et al. (2014). Qualitas: uma modelo de processo de desenvolvimento de software orientado a modelos.

Ambler, S. W. (2003). Agile model-driven development is good enough. *IEEE Software*, 20(5):71–73.

Ambler, S. W. (2006). Agile model-driven development (amdd). page 13.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Hunt, A., Jeffries, R., Kern, J., et al. (2001). Manifesto para desenvolvimento agil de software. *AGILE manifesto*.

Braga, A. d. A. and Leal, R. d. S. (2013). Estudo sistemático em dependabilidade e métodos ágeis: uma análise de falhas e defeitos.

Braga, V. (2011). *Um Processo para Projeto Arquitetural de Software Dirigido a Modelos e Orientado a Serviços*. PhD thesis, Dissertação, Universidade Federal de Pernambuco, Brasil.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182.

Caldiera, V. R. B.-G. and Rombach, H. D. (1994). Goal question metric paradigm. *Encyclopedia of software engineering*, 1:528–532.

EPF, E. F. (2014). Eclipse process framework composer. *Eclipse Foundation*.

Heeager, L. T. (2012). Introducing agile practices in a documentation-driven software development practice: a case study. *Journal of Information Technology Case and Application Research*, 14(1):3–24.

Maciel, R. S. P., Gomes, R. A., Magalhães, A. P., Silva, B. C., and Queiroz, J. P. B. (2013). Supporting model-driven development using a process-centered software engineering environment. *Automated Software Engineering*, 20(3):427–461.

Mellor, S. (2004). Agile mda, a white paper.

Mellor, S. J., Scott, K., Uhl, A., and Weise, D. (2005). Mda destilada: Princípios de arquitetura orientada por modelos. *Ciência Moderna Ltda*.

Miot, H. A. (2017). Assessing the normality of data in clinical and experimental trials. *Jornal Vascular Brasileiro*, 16(2):88–91.

OMG (2014). Model object management group model-driven architecture.

OMG, S. and Notation, O. (2008). Software & systems process engineering meta-model specification. *OMG Std., Rev*, 2:18–71.

Pressman, R. and Maxim, B. (2016). *Engenharia de Software-8ª Edição*. McGraw Hill, Brasil.

Sales, P. M. (2017). Integrando práticas do desenvolvimento dirigido a modelos ao scrum, dissertação de mestrado. pages 1–162.

Santos, N., Pereira, J., Ferreira, N., and Machado, R. J. (2018). Modeling in agile software development: Decomposing use cases towards logical architecture design. In *International Conference on Product-Focused Software Process Improvement*, pages 396–408. Springer.

Schwaber, K. and Sutherland, J. (2016). The scrum guide. pdf document.

Tomás, M. R. (2009). Métodos ágeis: características, pontos fortes e fracos e possibilidades de aplicação.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.