Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 181 (2021) 861–868

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2020

A Systematic Comparison of Roundtrip Software Engineering Approaches applied to UML Class Diagram

Dionisie Rosca*, Luísa Domingues

ISCTE-IUL - Instituto Universitário de Lisboa, ISTAR-IUL, Lisboa, Portugal

Abstract

Model-based software engineering contemplates several software developments approaches in which models play an important role. One such approach is round-trip engineering. The objective of this paper is to benchmark the comparative analysis of the round-trip engineering capability of three modelling tools: Papyrus, Modelio and Visual Paradigm. The conclusions drawn throughout the paper will answer the question: How effective are current code generation tools for documenting application evolution? Throughout the discussion, we have pointed out several improvements that these UML modelling tools have recently brought to the discussion. We have also proposed some improvements that we consider desirable to improve Roundtrip Engineering approaches pointing out possible directions and solutions.

Keywords: Model-Driven Engineering, Round-trip Engineering, Forward Engineering, Reverse Engineering, UML Modelling tools, Metamodel, Models, Code Generation, Traceability, Benchmarking

* Corresponding author.
E-mail address: deniz.r.v@gmail.com

10.1016/j.procs.2021.01.240

1. Introduction

Software models are a fundamental tool for engineers to rationalize a given system without, however, relying on a high level of technological detail. According to Khaled[1] through the last few years, building systems became very complicated. The current systems assume an increasing level of complexity derived from the level of demand for operations to be performed and the importance they assume in organizations. The relationship between the complexity of the system and its importance is directly proportional. Furthermore, these systems present an index of volatility in accordance with the organizational, technological and human changes that occur in society at an impressive pace. In addition, the time-to-market puts pressure on organizations to respond at a faster pace. In this context, ensuring the evolutionary maintenance of systems ensuring the alignment of implemented technical solutions and the abstract models that document them, is a challenge that requires resources of the organization that are by nature scarce: human resources and time. Faced with this challenge, organizations need to find optimal trade-off and thus it is frequent to find systems whose implementation does not match the models that support them. However, models are powerful tools to facilitate communication between the various stakeholders involved in system development, and visual notation is more comfortably assimilated than pure code [2][3]. The automatic or semiautomatic support of tools to models-oriented approaches, such as roundtrip engineering, allows to increase the consistency of this subprocess of software development, automating integration activities between phases of software development from analysis, through design until implementation, and therefore the different types of model and source code. Therefore, reducing the time and effort of the development and contributing to the maintenance of the quality of the system under construction.

The important role of the introduction of the roundtrip engineering topics with Unified Modelling Language (UML) in the teaching programs of undergraduate degree students is recognized, and to this end tools have been developed to support it, particularly in connection with the Java language, in the form of plugins for Eclipse [4][5].

According to Khaled[1] any complicated system needs a tool to be built. For this reason, the research work developed by Khaled[1] and Wang[6] focused on the generic comparison of UML modelling tools. In the last decade there is no evidence of such research and it is therefore appropriate that scientific research be conducted to assess the use of the modelling tools most commonly used today, as supporting tools for round-trip engineering approaches. The benchmarking to be carried out in this document will serve as decision support for the engineers involved in model-oriented software development projects, namely the decision on which engineering tool to adopt. In addition, with the proposal of a qualitative assessment framework and the definition of metrics for quantitative assessment, opportunities for improvement in the roundtrip engineering process will be identified in the UML modelling tools under study.

2. Literature Review

2.1 UML Modelling Concepts

The software development industry is most often driven by business opportunities. In this scenario, the ability to increase market share and reduce time to market are important issues that strengthen two aspects of software development processes, the ability to use experience and the ability to gain independence from software platforms. These aspects are related to reuse at different levels of abstraction and raise some important questions:

- How to automate the generation of software artefacts from high level requirements?
- How to ensure the properties of the requirements throughout the development process?

To address this problem, Software Engineering professionals and organizations such as Object Management Group (OMG) have developed ways to systematize software asset creation, Model Driven Architectures (MDA), which can be used to achieve wide reuse scale through frameworks and configurable framework platforms independent of software asset representations [7]. UML diagrams are formal representations of various software components and their stockholders; therefore, they play an important role in creating software.

Models serve different purposes: establish a clear understanding of the problem; communicate a clear vision of the problem and solution; and generate low level implementations from high level models [8]. To make automation, models must have a defined meaning. A language consists of syntax and semantics. The syntax can be machine-centric or human. “Semantics define what the syntax means by linking the syntax to a semantic domain, rather like arithmetic expressions “mean” numbers “.

The Unified Modelling Language is one of the most successful cases in the Information Technology (IT) industry, being used to outline the requirements of an IT system. Now, UML is used in a variety of ways by professionals with different backgrounds. Since UML was standardized, the language has been extended into full System Modelling Language (SysML) to solve system development problems. Programmers, business analysts, architects, systems analysts and other stakeholders that rely of UML must learn the language, for them, UML diagrams are not simple images for communication and documentation purposes is a view of a formally defined mode [9]. Software systems models are constructed, visualized, documented and specified using UML which is a system of language and notation. Models represent a system, they can be structural - diagram class and collaboration; behavioral - state machine, activity diagram and sequence; physical - component and deployment diagrams) of external functionalities and instances use case and object diagram [10]. Each model is applied in specific problem domains and solutions [8]. The path from the problem domain to the solution domain requires an understanding of architecture. To get the logical architecture implementation in relation to the application in development, some levels of abstraction must be crossed [11]. "Each of the views or models in the system captures the structure or behavior of the system with a specific abstraction depth" [12]. For this study we focus in structural view, more precisely in class diagrams.

2.2. Concepts to Understand Round-Trip Engineering

According [13], Model-Driven Development (MDD) "automates the transformation of models from one form to another, express each model, both source and target, in some language. Must be defined the two languages someway, because modeling is an appropriate formalism to formalize knowledge, we can define a modeling language's syntax and semantics by building a model of the modeling language so-called metamodel".

The purpose of round-trip engineering is to keep several artifacts up-to-date and consistent in propagating changes between artifacts. Making the artifacts consistent by propagating the changes is also called synchronization. Round-trip engineering is a special case of synchronization that can propagate changes in multiple directions, from models to codes, and vice versa. Round-trip engineering is difficult to achieve in a general scenario, due to the complexity of the mappings between the artifacts. The ability to trace new and changed requirements to their impacted components provides critical support for managing change in an evolving software system. Tracing is a standard abstraction stereotype primarily used to trace requirements and changes to models for elements or sets of elements that represent the same concept in different models. Thus, tracking is an "intermodal" relationship. These tracking dependencies between models are usually represented by dependencies between elements contained in models. The direction of the trace (i.e. customer and vendor designation) is at the discretion of the modeler, and since model changes can occur in both directions, the direction of dependency can often be ignored. Mapping specifies the relationship between the two but is rarely computable and is usually informal.

Sendall[14] states that "round-trip engineering is a challenging task that will become an important facilitator for many approaches to Model-Driven Software Development." Round-trip engineering involves the synchronization and maintenance of the model, allowing software engineers to move freely between different representations. This round-trip engineering vision is only realized in some limited tools due to maintenance difficulties [14].

Two other conversion techniques are most often found in the tools: forward engineering and reverse engineering Forward Engineering involves the generation of software artifacts with a level of detail close to the final implementation, from conceptual UML models; Reverse Engineering is the process of which a software artifact is deconstructed to reveal its designs or architecture, i.e. generating UML conceptual diagrams. Some tools also support round-trip engineering, which involves several stages of forward and reverse engineering, so that software artifacts, such as programming language code and UML class diagrams, are synchronized whenever changes. However, it is important to note that the roundtrip engineering process, although including forward and reverse engineering functionalities, is functionally different, as these two methods when applied in isolation do not assure the synchronization between the models.

3. Methodological Approach

In this chapter will describe step by step the benchmarking method [15] to be used in this research work that will be applied to a group of UML modelling tools. Figure 1 depicts the phases benchmarking process:

Understand the context – At this stage the basic set of criteria for the selection of working cases to apply the roundtrip engineering approach was defined. The case should include modelling elements to choose UML metamodel according to clear criteria. Consequently, the class diagrams that would constitute the case studies were selected from the case base platform: uml-diagrams.org. In this article we focus on the demonstration of results in a single illustrating case, "The Hospital Management - class diagram", chosen because of its representativeness;

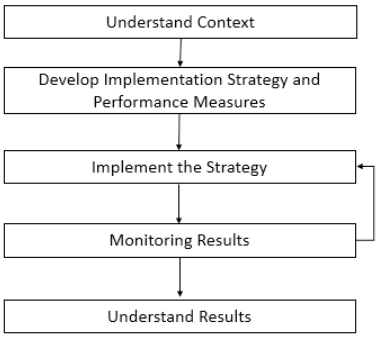

Figure 1 – Benchmarking Phases

Develop implementation strategy and performance measures- which aims (i) define the range of approaches will be applied in the case studies and, (ii) define the criteria used to compare the implemented case studies, the metrics identification was based on previous research work;

Implement the strategy – conduct the experiments that consist of generating the code from UML diagrams, applying three techniques: (i) forward engineering; (ii) generating the UML diagrams from software artifacts, i.e., reverse engineering; and (iii) synchronize the diagrams and the code, when each one is modified, that mean applying round-trip engineering. At this stage data were also collected for analysis at a later stage;

Monitoring results – at this stage the data collected has been analyzed for each tool and case study, as well as a comparison between the tools and their cases, ensuring that all the proposed cases were covered. If during this phase inconsistencies were detected, the previous phase step three would be repeated until satisfactory results;

Understand results - where conclusions were drawn, that is, the results of the comparisons made in the fourth phase led to conclusions and, in turn, recommendations have emerged.

4. Tool Selection Criteria

After an exhaustive survey of the available tools supporting the UML standard, we made a preliminary selection of those supporting the MDA responsible for ensuring the application of the roundtrip engineering approach. This way Papyrus, Modelio, BoUML, MetaEdit ++, Visual Paradigm and Acceleo among others were selected. At a later stage, and through a more in-depth study of each tool, we selected the three that would constitute those used in the analysis of the case study: Papyrus, Visual Paradigm and Modelio.

The Papyrus tool was chosen because it is a UML2 graphical modeler that currently supports eight of the all diagrams, is therefore more robust. Because modelling is not enough to claim to be an MDA tool, it also provides code generation (C ++, Java) and facilitates the connection of external tools (planning analysis) to enable models to be the main artefacts of the development process. There are studies where Papyrus has been used in similar experiments, so it was a point of reference and one of the reasons for choosing. Another reason is that it is an open source project based on an Eclipse environment, which is the most widely used tool for software development and offers the freedom to enhance and evolve the tool without relying on any entity. Papyrus is used in industry and research, making it a solid and reliable tool.

Visual Paradigm is a tool like Papyrus, but with specific characteristics. It is a professional tool, and, unlike Papyrus, it is a proprietary software that gives the tool guaranteed stability and support. It is used in industries for

process automation. It has an integrated graphics editor for UML 2.x and can generate source code through MDA. Another reason is that it has direct integration with Eclipse, since it makes it easy to have everything focused on a single IDE. Generic Modelling Environment (GME) is a configurable Meta-CASE tool that provides tool sets for creating a domain-specific modeling environment, the configuration is done by specifying the modeling metamodel that represents the modeling language of the application domain.

Modelio like Papyrus is an open source tool with its own IDE. And one reason for choosing was beyond a predefined set of model-driven code generation modules, Modelio provides a strong integrated MDA capability through its UML profile editor and rich Java API for metamodel access, model transformation and tool customization. Another reason is that modelling provides complete coverage to support modelling activities for each stakeholder within a company that in turn is reflected in the diagrams the tool supports. Its modelling support includes UML2, BPMN, Enterprise Architecture, SOA Architectures, Requirements Analysis, Dictionary and Business Rule Analysis, which in turn with the other tools, is used in industry and makes it a reliable tool.

5. Case Study

The Hospital Management case study is represented by the class diagrams depicted in Figure 2. This case study has a high level of compliance because almost the totality of class diagram elements is encompassed in this case study. More specifically, it is possible to identify in this diagram associations, covering the various multiplicities, aggregations, compositions and generalizations. It also includes all types of attributes. This diversity was determinant in the choice of this case because it ensures greater representativity of the different elements of the class diagram in the generation of the code. Another reason for the choice of this case was the familiarity of the subject and the easiness of understanding of its semantics, allowing the analysis of the results obtained in code generation, in the application of forward engineering, or UML diagrams, resulting from the reverse engineering approach.

The domain model for the Hospital Management System is represented by two class diagrams. The purpose of the class Organization diagram (on the left side) is to represent the hospital structure, staff, patient relationship, and patient treatment terminology. A"Person" may be associated with different hospitals, and a "Hospital" may employ or serve several "People". The "Person" class derived the name and "homeAddress" attributes. The "Patient" class inherits attributes from the "Person" class. Several inherited attributes, name, gender, and "date of birth" are shown with the caret symbol(^). The Ward and Teams class diagram (on the right side) represents: (i) the Wards, i.e., division of each hospital or a suite of rooms shared by patients who need a similar kind of care; and Teams, also called firms, established groups of doctors.

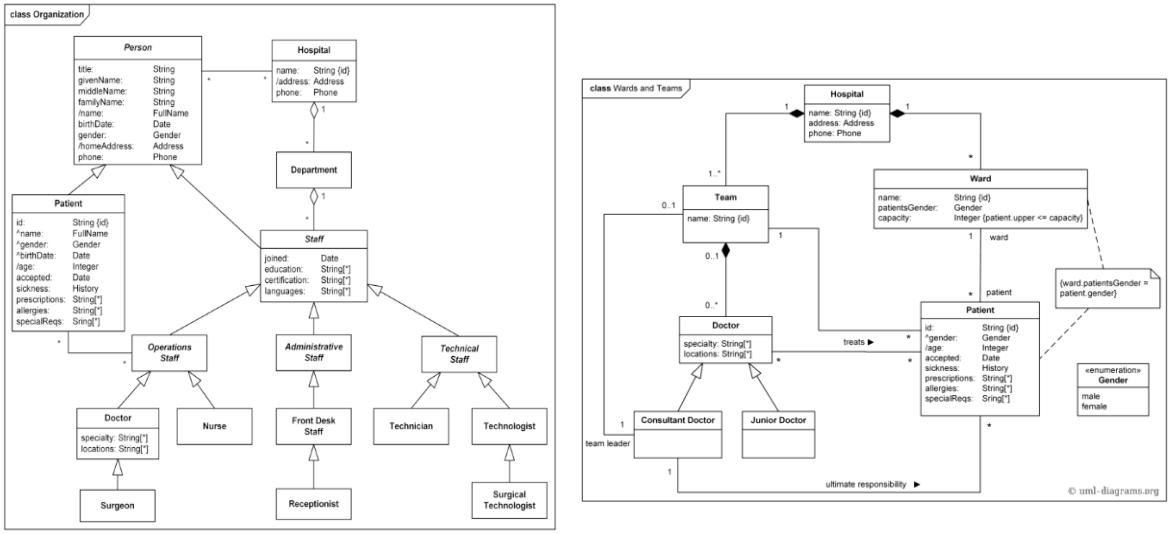

Figure 2 – “The Hospital Management” Class Diagram

The following three scenarios explains how the experiment was performed:

- *Scenario1: Forward and Reverse Engineering without any changes* - The experiment itself begins with drawing the diagrams and validate them using the validation framework of the UML modelling tool. In a first step we proceeded to code generation, i.e., applying Forward Engineering. At this step, we certified the quality of the code obtained using the metrics and manual analysis of the generated code. In a second step we applied Reverse Engineering using the generated code and repeat the same quality validation process focusing on the generated class diagram. This scenario aims to assess the strength and coherence of the process.
- *Scenario2: Forward and Reverse Engineering with changes* - This scenario followed the same procedure as scenario 1, however after applying forward engineering the obtained code was changed, namely by adding new elements, before applying reverse engineering. In the code generated by class Organization was added a new class named "Reclamation" with five attributes: id, complaint, person, staff, date. This class was linked by a multiple association with the class "Staff". Regarding class Wards and Teams: (i) new attributes, id and name, were added on class Junior Doctor; (ii) relation between Team and Patient was deleted; and (iii) a new class Administrative was created with relation with class Hospital and Patient.
- *Scenario3: Round-Trip Engineering with changes* - Applying the same procedure that was applied in the previous scenario, the difference is about the method applied, in this scenario we used Round-trip Engineering. This method ensures the synchronization between the models and the generated code and does not lose reference to the initial state and successive changes.

6. Analysis and Discussion of Results

6.1 Comparison Criteria

From the broad set of possible metrics, we select design metrics, i.e. metrics that can be extracted from the source code itself. These metrics are generally used to evaluate the size and, in some cases, the quality and complexity of the software. The metrics we use are called direct measurement metrics because their calculation involves no other entities [16]. The metrics selected are based on the object-oriented language indicator framework. The option for selecting metrics extracted from code entities was mainly motivated by the fact that they have a simple and clear definition and provide a set of reliable indicators. Indirect measurement, which includes programmer productivity, defect detection density, or defect density module such as [16], where metrics are combined to generate new metrics, is not used. As such, we do not use composite metrics, which raise the question of consistency dimension. These are the set of selected metrics applied to class diagram: Number of Children(NSC), Number of Overridden Methods (NORM), Number of Attributes (NOF), Number of Static Attributes (NSF), Number of Methods (NOM), Number of Static Methods (NSM), Number of Class (NOC), Number of Interface (NOI), Number of Packages (NOP).

6.2 Analysis of Results

Resulting from the scenario 1, Table 1 and Table 2, exhibit respectively the result from the application of forward engineering method and reverse engineering method without changes. Analysing the results of table 1, we found that they are similar, that is, there is no relevant difference between the generated code (real column) and the diagram model (theoretical column) in all modelling tools used. Analysing the table 2 we can draw the same conclusions.

		class Organization						class Wards and Teams					
		Papyrus		Modelio		Visual Paradigm		Papyrus		Modelio		Visual Paradigm	
Metrics	Acronyms	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical
Number of Children	NSC	11	11	11	11	11	11	8	8	8	8	8	8
Number of Overridden Methods	NORM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Attributes	NOF	28	28	27	28	27	28	18	17	17	17	18	17
Number of Static Attributes	NSF	0	0	0	0	0	0	0	0	0	0	0	0
Number of Methods	NOM	54	54	56	54	57	54	43	43	45	43	46	43
Number of Static Methods	NSM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Class	NOC	21	21	21	21	21	21	19	16	18	16	18	16
Number of Interface	NOI	0	0	0	0	0	0	0	0	0	0	0	0
Number of Packeges	NOP	1	1	1	1	1	1	1	1	1	1	1	1

Table 1 – Scenario 1- Forward Engineering without changes

		class Organization						class Wards and Teams					
		Papyrus		Modelio		Visual Paradigm		Papyrus		Modelio		Visual Paradigm	
Metrics	Acronyms	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical
Number of Children	NSC	11	11	11	11	11	11	8	8	11	8	11	8
Number of Overridden Methods	NORM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Attributes	NOF	28	28	27	28	27	28	18	18	27	17	27	18
Number of Static Attributes	NSF	0	0	0	0	0	0	0	0	0	0	0	0
Number of Methods	NOM	54	54	56	54	57	54	43	43	42	45	43	46
Number of Static Methods	NSM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Class	NOC	21	21	21	21	21	21	19	19	18	18	18	18
Number of Interface	NOI	0	0	0	0	0	0	0	0	0	0	0	0
Number of Packeges	NOP	1	1	1	1	1	1	1	1	1	1	1	1

Table 2 – Scenario 1- Reverse engineering without changes

Making a qualitative analysis of the code, as well as the generated diagram, we verify that there is consistency between them and that they are complete.

Considering the results of the second scenario the conclusions did not differ much from the previous scenario. Thus, the code generated from the class diagrams presents the same results as the Forward Engineering (Table 1) in the previous scenario as would be expected. The results of Reverse Engineering application after the changes made in code (see Table 3) reveal that quantitatively the real values are close to the expected ones (Theoretical column).

		class Organization						class Wards and Teams					
		Papyrus		Modelio		Visual Paradigm		Papyrus		Modelio		Visual Paradigm	
Metrics	Acronyms	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical
Number of Children	NSC	11	11	11	11	11	11	8	8	8	8	8	8
Number of Overridden Methods	NORM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Attributes	NOF	33	33	33	33	33	33	20	20	20	20	20	20
Number of Static Attributes	NSF	0	0	0	0	0	0	0	0	0	0	0	0
Number of Methods	NOM	56	60	54	60	57	60	45	45	45	45	45	45
Number of Static Methods	NSM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Class	NOC	22	22	22	22	22	22	20	21	21	21	23	21
Number of Interface	NOI	0	0	0	0	0	0	0	0	0	0	0	0
Number of Packeges	NOP	1	1	1	1	1	1	1	1	1	1	1	1

Table 3 – Scenario 2 - Reverse engineering after changes

When analysing the code generated in the third scenario (see Table 4), application of Round Trip Engineering approach, it was found that the code generated reflects the changes made in the diagram, however some inconsistencies were detected compared to the code generated before the changes, namely in the attributes types and foreign keys.

Round-trip engineering		class Organization						class Wards and Teams					
		Papyrus		Modelio		Visual Paradigm		Papyrus		Modelio		Visual Paradigm	
Metrics	Acronyms	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical	Real	Theoretical
Number of Children	NSC	9	11	8	11	7	11	6	8	6	8	7	8
Number of Overridden Methods	NORM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Attributes	NOF	25	33	23	33	24	33	14	20	16	20	17	20
Number of Static Attributes	NSF	0	0	0	0	0	0	0	0	0	0	0	0
Number of Methods	NOM	50	60	48	60	48	60	40	45	38	45	40	45
Number of Static Methods	NSM	0	0	0	0	0	0	0	0	0	0	0	0
Number of Class	NOC	21	22	21	22	21	22	21	21	21	21	21	21
Number of Interface	NOI	0	0	0	0	0	0	0	0	0	0	0	0
Number of Packeges	NOP	1	1	1	1	1	1	1	1	1	1	1	1

Table 4 – Scenario 3 – Roundtrip Engineering with changes

Based on the analysis of the quantitative metrics extracted from the application of the third scenario to the class diagrams of all the case studies considered, it was possible to determine that the tools had a success rate higher than 80% (Modelio -82%; Papyrus - 85%; Visual Paradigm - 88%). The calculation of this success rate was based on the average of all metrics, resulting from the values obtained by the code and diagrams generated compared to what was expected by the original diagrams and code. In general, one may conclude that the generation of the code and diagrams were partially executed, and with some corrections we may have an executable code or a consistent diagram.

7. Conclusion

Model round-trip engineering will be a key factor in many next-generation model-driven software development approaches as it will allow modelers to move freely between different system representations. In this paper we compared the Papyrus, Modelio, and Visual Paradigm tools in terms of round-trip engineering capabilities.

The evaluation of UML modelling tools behaviour when applying forward engineering, reverse engineering and round-trip engineering approaches was based on quantitative metrics extracted from the artefact code and conceptual models generated. However, it was necessary to complement the quantitative evaluation with a qualitative analysis based on manual comparative analysis. The focus of this analysis was to understand whether diagrams and code artefacts were generated semantically correctly.

Three scenarios were followed to evaluate the behaviour of the tools in applying the generation methods to the class diagram. In the first scenario, applying Forward and Reverse Engineering methods without any change to the initial model, after analysing the quantitative metrics it was possible to conclude the robustness and coherence across all UML modelling tools. The qualitative analysis confirmed the correct semantic interpretation of all classes and links between them. Although these tools always generate new diagrams or new code, with a new context, we could see that all the tools followed them successfully. The use of scenario two, Forward and Reverse Engineering, with changes made to the code generated after forward engineering was applied, led to the same conclusions as the previous scenario, at the level of quantitative metrics and semantic evaluation. However, the conclusions were different in the

application of the third scenario, Roundtrip Engineering with the same changes introduced in the previous scenario. In this case the changes were always applied in the same model and context, and from the manual analysis we can see that semantic inconsistencies emerged. Therefore, we were able to conclude the Rountrip Engineering approach applied to the class diagram should be based on a clear definition of the consistency of the model used to locate possible inconsistencies. Once detected, the Rountrip Engineering method can use different model reconciliation strategies to make it consistent.

In general, all the UML modelling tools tested were able to successfully apply the different scenarios and obtain positive results. However Visual Paradigm stood out for the quality of the generated code, as well as the quality of the diagrams. Moreover, Papyrus stood out for its rich customization, for example it facilitates the definition of metamodels, which is not possible with the other tools. The Modelio distinguished itself in the ease of creating the models and generating the code. Another factor to consider is that proprietary and therefore paid-for applications offer faster support when compared to open source applications, which limits the resolution of several emerging issues. Comparing the work of other authors, namely Khaled[1], we could notice that there has been a considerable evolution in the definition of models, i.e., it is done in a more formal way, and the quality of UML increases what makes the application of the Roundtrip Engineering approach increasingly feasible in software development. Regarding the inconsistencies detected by the author Wang[6] in the Roundtrip Engineering application, some of them have been solved as, for instance, in static classes the accuracy increases when the code is generated, the attributes and their types are well identified and the links between the classes semantically well defined. Therefore, the generated code has better quality and does not require major interventions.

Throughout this discussion, we have pointed out several improvements that the UML modelling tools have recently brought to the discussion. We have also proposed some improvements that we consider desirable to improve Roundtrip Engineering approaches pointing out possible directions. The most relevant recommendation was that these tools should cover a greater consistency of the model, applying the MDA to the metamodel, i.e., creating layers of increasingly generic models to be applied to various models and model contexts. As future work, we are interested in investigating how to integrate the tasks of defining, deciding and solving the coherence into a single approach.

References

[1] Khaled, Lena (2009). A Comparison between UML Tools. In Proceedings of the 2009 Second International Conference on Environmental and Computer Science (ICECS '09). IEEE Computer Society, Washington, DC, USA, 111-114. https://doi.org/10.1109/ICECS.2009.38

[2] Ardis, M., Daley, N., Hoffman, D., Siy, H. and Weiss, D. (2000), Software product lines: a case study. Softw: Pract. Exper., 30: 825–847. DOI: https://doi.org/10.1002/(SICI)1097-024X(200006)30:7<825::AID-SPE322>3.0.CO;2-1

[3] Atkinson, C., Kühne, T. (2003) *Model-Driven Development: A Metamodeling Foundation*. IEEE Software, vol. 20, 5, 36-41. DOI: https://doi.org/10.1109/MS.2003.1231149

[4] Model Goon team (2011). *UML4Java: Bring UML Visual models into the Java World.* Available online at: http://www.modelgoon.org

[5] M. Usman, A. Nadeem, and T. Kim, "UJECTOR: A tool for Executable Code Generation from UML Models", International Conference on Advanced Software Engineering and its Applications (ASEA'08), IEEE Computer Society Press, Hainan Island, China, December. 13-15, 2008.

[6] Gene Wang, Brian McSkimming, Zachary Marzec, Josh Gardner, Adrienne Decker, and Carl Alphonce. (2007). Green: a flexible UML class diagramming tool for Eclipse. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (OOPSLA '07). ACM, New York, NY, USA, 834-835. DOI: https://doi.org/10.1145/1297846.1297913

[7] S. Deelstra; M. Sinnema; J. Gurp, Model Driven Architecture as Approach to Manage Variability in Software Product Families. Workshop on Model Driven Architecture: Foundations and Applications June 26-27, 2003.

[8] Brown, A. W., Conallen, J., Tropeano, D. (2005) Introduction: Models, Modelling, and Model-Driven Architecture (MDA). In Beydeda, S., Book, M.and Gruhn, V. (Eds.) Model-Driven Software Development. New York, Springer Verlag, 1-16.

[9] Booch, Grady & Rumbaugh, James & Jacobson, Ivar. (1999). Unified Modelling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). J. Database Manag, 2000

[10] Terrasse, M.-N., Savonnet, M., Becker, G. (2001) A UML-based Metamodeling Architecture for Database Design. In Proceedings of 2001 International Symposium on Database Engineering and Applications (IDEAS'01). Grenoble, France, July 16-18, IEEE Computer Society.

[11] Cook, S. (2004) Domain-Specific Modeling and Model Driven Architecture. Available at http://www.bptrends.com.

[12] Sendall, S., Kozaczynski, W. (2003) Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software, vol. 20, 5, 42-45.

[13] Mellor, Stephen J., Clark, Tony and Futagami, Takao (2003) Model-driven development: guest editors' introduction. IEEE Software, 20 (5). pp. 14-18. ISSN 0740-7459

[14] S. Sendall, J. M. Küster, "Taming Model Round-Trip Engineering" (October 2004), Proceedings of Workshop on Best Practices for Model-Driven Software Development, Vancouver, Canada.

[15] Coffel, K. *et al.* (2010) *The national academies press*. doi: 10.17226/14402.

[16] N. Fenton and S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, second ed. London: Int'l Thomson Computer Press, 1996.