Feedback on How MDE Tools Are Used Prior to Academic Collaboration

Vasco Sousa
University of Montreal
Montreal, Canada
vasco.da.silva.de.sousa@umontreal.ca

Eugene Syriani
University of Montreal
Montreal, Canada
syriani@iro.umontreal.ca

Martin Paquin
Fresche Legacy
Montreal, Canada
martin.paquin@freschelegacy.com

ABSTRACT

Several studies in the model-driven engineering (MDE) literature report on companies adopting MDE technologies as a result of long collaborations with academics. However, there are more companies than reported that are already using MDE, even without active MDE researchers partners. Therefore we do not have a clear view of how these industries are using MDE and how it is benefiting them. Fresche Legacy is a company that already started using MDE even before collaborating with us. To better understand how MDE is used in such industries and to what extent MDE can solve challenges in this business, we conducted a series of interviews with the employees working with MDE. The findings presented in this paper report a successful integration of MDE tools with their in-house software used in production, challenges encountered, and how they surmounted them. We also discuss how this success story can help other companies benefit from MDE.

CCS Concepts

•**Software and its engineering** → **Model-driven software engineering;**

Keywords

Model-driven engineering; legacy software; interview

1. INTRODUCTION

Several studies in the model-driven engineering (MDE) literature report on companies adopting MDE technologies as a result of long collaborations with academics. However, there are more companies than reported that are already using MDE, even without active MDE researchers partners. Therefore we do not have a clear view of how these industries are using MDE and how it is benefiting them. For example, we may discover unexpected or improper uses of MDE tools, misinterpretation of the MDE philosophy, new ways to use MDE, or undocumented application domains where MDE is being applied. This is also an opportunity to understand what makes a successful integration in MDE and where the major limiting factors of MDE are. Furthermore, other companies may benefit from such a report, whether they are in a situation where they cannot partner with an MDE expert because they lack contacts, it is difficult to partner with academics, or they cannot find model-driven engineers available on the market.

Fresche Legacy is a Canadian software company that focuses on the maintenance and modernization of legacy applications. In particular, they specialize in IBM System i technologies where legacy applications run on AS/400 machines. The particularity of these very old technologies is that most of the applications are all programmed in the Report Program Generator (RPG) language [20]. RPG falls under the same procedural imperative paradigm as PL/I and COBOL where, in particular, the code is written in columns and lines have fixed length. Even with more sophisticated languages, such as the Synon/2E [17] framework which generates RPG or COBOL code, most applications are still developed directly in RPG. Synon is a 4GL that allows developers to focus on business activities and higher abstractions independent from RPG. Most of Fresche Legacy projects involve both Synon and RPG. In order to keep maintaining and developing applications for customers still using these legacy applications, Fresche Legacy has been looking for ways to automate the modernization of AS/400 legacy applications and migrate to Java applications with a web front-end and a database back-end.

When we approached Fresche Legacy to propose solving their problem using MDE technologies, we were happily surprised to realize that they were already doing so. Therefore this paper reports on how a company, such as Fresche Legacy, is using MDE in their business even before collaborating with MDE researchers. To get a feel of how Fresche Legacy engaged in MDE, we asked ourselves the following research questions:

RQ1: How did the company get into MDE?

RQ2: How has MDE been integrated in the company?

RQ3: How did MDE tools help realize the modernization of legacy systems?

RQ4: How does this MDE solution scale to larger modern-

SAC 2017,April 03-07, 2017, Marrakech, Morocco

http://dx.doi.org/10.1145/3019612.3019775

ization projects?

With **RQ1** we are interested in find out how MDE is discovered and perceived by the industry. Once a company learns about MDE, we want to understand how they apply this new knowledge in their projects with **RQ2**. With **RQ3**, we are interested in the technical details of the adoption of MDE from a practical point of view of software tools. With **RQ4**, we question what is necessary to transition an MDE solution to other projects in the industry of legacy system modernization.

To answer these questions, we employed a series of qualitative methods with the engineers who worked with MDE, including meetings, interviews, and observations. The findings presented in this paper report a successful integration of MDE tools with their in-house software used in production, challenges encountered, and how they surmounted them. We also discuss how this success story can help other companies benefit from MDE.

The remainder of the paper is structured as follows. In Section 2 we present the projects where Fresche Legacy already used MDE before our collaboration. In Section 3, we describe the methods used to collect the necessary information to evaluate the use of MDE by the company. In Section 4 report on our findings and analysis. In Section 5, we discuss how the results of this study relates to other studies about the adoption of MDE in the industry. Finally, we conclude in Section 6.

2. TWO MODERNIZATION PROJECTS

In this section, we present the two projects where Fresche Legacy applied MDE. The purpose of this section is to understand the complexity of this tedious modernization task.

2.1 Synon modernization project

Fresche Legacy decided to make a first proof of concept for using MDE to automatically migrate a legacy application developed with Synon to a Java-based web application. This project was chosen because Synon is already at a level of abstraction higher than RPG and therefore assumed that the complexity of the task will be lesser. Synon dates from 1984 and is used to specify data and processes to be generated for AS/400 systems. It is highly structured, with specifications for `Files` (or database tables) as representations of the application data structures, `Functions` to contain and organize the business logic, `Action Diagram Elements` that correspond to the business logic where each element corresponds to a statement, `Screens` to display representations, and `Reports` for static data representations. Synon can also be viewed as an early day modeling approach, being the precursor of Architected Rapid Application Development (ARAD) [8], and even more general Computer Assisted Software Engineering (CASE) approaches. One major advantage of starting the modernization process with Synon is that it provides a higher level of abstraction than the generated code and has many OO abstractions that are compatible with a modern modeling views of systems.

2.1.1 *Applying MDE*

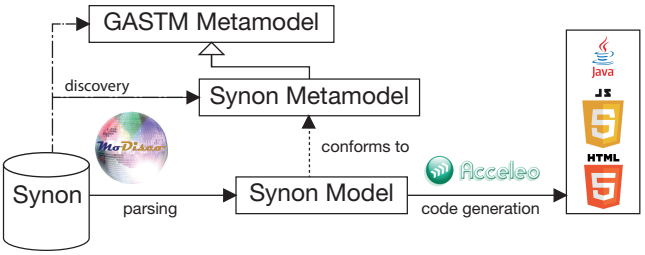

Figure 1: Overview of the process to migrate Synon code to a web application

Figure 1 depicts the process of modernizing Synon applications using MDE. First, they reverse engineer Synon code using MoDisco [3] to extract the required information from the Synon repository and produce an Ecore model of the Synon code. MoDisco also derived a metamodel for Synon. The derived metamodel consists of 245 classes total (see Table 1). Note that it also inherits from the Generic Abstract Syntax Tree Metamodel (GASTM) provided by MoDisco. The extracted model is a 17.5 MB XMI file, containing the representation of 126 `Files`, 263 `Functions`, 13 841 `Action Diagram Elements`, 32 `Screens` and 25 `Reports`.

Metamodel	Synon	RPG
Number of classes	245	477
Number of references	239	322
Number of attributes	392	151

Table 1: Size of Synon and RPG metamodels

With the use of Acceleo templates[1], they generate a web-based implementation of the system from the Synon model. Table 2 summarizes the characteristics of the resulting generated application. Most of the work implementing the templates is dedicated to the business logic and interaction with the database, implemented in Java. The remaining generated artifacts are for the web front and configuration. Note that this generation process ignored the `Report` elements. Up to 10% of the lines of code in each file are comments.

Language	Files	Lines of code
Java	1 125	60 396
CSS	2	18 223
Javascript	101	9 877
HTML	46	4 347
LESS	19	3 861
Maven	2	1 274
JSON	6	174
XML	4	173
DOS Batch	1	127
Bourne Shell	2	114
Total	1 308	98 566

Table 2: Characteristics of the generated application from Synon

[1]https://eclipse.org/acceleo/

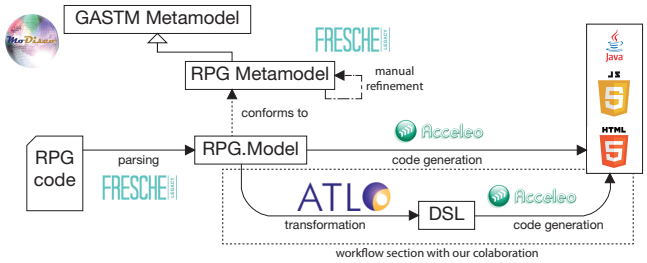

Figure 2: Workflow of RPG to Java modernization

2.1.2 *Successful adoption*

The MDE-based Synon modernization project lasted one year, starting in September 2014. The work was performed by one architect who, within that period of time, learned about MDE and the necessary tools. He applied this acquired knowledge to design, implement, and test the project. In the final month, five additional developers joined the project. Their task was to create the templates for the code generator as well as the data to test the generated application. Once all the components of the process were completed, the migration from Synon to the Java code for this specific project took less than a day.

This pilot project was a success. Therefore the metamodel, reverse engineering tool, and templates are now being used in the core development of the company. This MDE technique has been applied on two other projects. They are of similar scale as this one, in the order 10^5 lines of code with around 10^3 files produced. From past experiences at Fresche Legacy, projects of this scale required around 6.25 man-months effort to produce the final application manually. With MDE, each of the two projects took around 5.17 man-years to automatically produce the final application. This represents a 17% improvement in terms of time productivity for the developers. Once the project was completed, other similar projects only required 1.5 man-weeks effort, thanks to reuse and automation provided by the MDE solution.

These are considered small scale projects. According to the company, a medium size modernization project is typically in the order of 10^6 lines of code with around 10^4 files produced. They typically require 225 man-years efforts: 3 years with a team of 75 people. Further down the spectrum, large scale projects reach the 2×10^8 lines of code and can require around 750 man-years of effort: more than 5 years with a 150 person team, the current largest expected team at the company. Recently, Fresche Legacy estimated a medium sized Synon modernization project for a customer using MDE to require a 180 man-years effort, which is a 20% improvement compared to manually developed projects.

2.2 RPG modernization project

Currenly Fresche Legacy is developing the process of modernizing RPG applications to Java using MDE. RPG as significantly evolved through the years. Originally a procedural language with strong static typing, it used a fixed columns and line length so it could be input into the system with punch cards. The current version used is RPG IV and allows for looser format. Many other functionalities were included to adapt the language to more modern requirements. For example, the code can be structured to take advantage of reuse it may have access to a database, or be designed for sequential data, such as tapes. This raises a challenging problem when modernizing legacy systems in RPG because of the variety of possible implementations and programming idioms that can drastically differ from one RPG application to another.

2.2.1 *Scope prior to collaboration*

After a successful proof of concept with Synon, the company proceeded with migrating RPG-based projects. The major difficulty is the low level of abstraction of the legacy paradigm that needs to be migrated to an object-oriented paradigm.

The RPG project to migrate is a booking system of a travel agency. Following the approach for Synon, an RPG model is extracted from the RPG code base and Acceleo templates are used to generate the target application. This extraction simply ported RPG code to Ecore, as a one-to-one correspondence with no abstraction or alteration of the logic. In this case, the RPG metamodel was hand-written as a refinement of the Generic Abstract Syntax Tree Metamodel (GASTM)[2] because MoDisco does not support RPG. Furthermore, they implemented a parser that extracts RPG code and produces an RPG model that conforms to the RPG metamodel. The parser interfaces with MoDisco code in order to manipulate GASTM elements. In this project, the resulting RPG model is a 16 MB XMI file, containing 145 208 elements. Table 1 summarizes the characteristics of the RPG metamodel. The process, illustrated on the top of Figure 2, is the scope of the project prior to our collaboration.

2.2.2 *Need for collaborating with MDE experts*

Fresche Legacy has been modernizing projects where the target application still relied on frameworks and libraries that emulate the behavior of the legacy application, but now in a modern system. This however hampers enormously maintenance of the new modernized application, since it still requires skills and knowledge of the legacy paradigm, e.g., RPG, which do not transcend to contemporary paradigm, e.g., Java. Therefore, any maintenance task or client change requests requires re-engineering the legacy code. However, all the refactorings and optimizations performed at this level are time consuming, error prone, and dependent on scarce expertise.

2.2.3 *Scope of the collaboration*

Our collaboration with Fresche Legacy aimed at providing a clear abstraction from the legacy system and its logic in order to facilitate re-engineering without prior knowledge of the legacy code. Therefore, we proposed to define a domain-specific language (DSL) that abstracts RPG details and emphasizes on the business logic of the application. An ATL transformation produces a domain-specific model (DSM) conforming to this DSL from the RPG model. Manual refactorings and optimizations are performed on the

[2]http://www.omg.org/spec/ASTM

DSM before Acceleo templates generate the target application. This process is illustrated in the dotted box at the bottom of Figure 2.

3. METODOLOGY

This qualitative case study was performed in order to understand how a company, such as Fresche Legacy, uses MDE technologies without academic expertise influence.

3.1 Participant selection

The participants of this study are engineers of the company who have been working with MDE, specifically on either of the two projects presented in Section 2. This included one senior architect, two on-site developers, and one off-shore developer. They all account for 12 to 20 years of experience in programming and software engineering. Their past programming experience was mainly focused on Java development in modernization, framework development, and system emulation. They have experience with UML modeling, and two of the participants learned UML at the university.

3.2 Data collection

We conducted series of informal meetings, semi-structured interviews, and observations. First, we approached participants with informal conversations and participated as observers in relevant company meetings. Then, we performed semi-structured interviews to collect answers to our research questions. Finally, we followed up with informal discussions to clarify some points brought to light during data analysis. We collected data from informal and observational methods over a period of two months. The formal interviews where held during the last two weeks. Additionally we collected quantitative data to contextualize our observations and interview data. This quantitative data came from code inspection, reports on the projects, models, transformation artifacts and their execution. We also collected overall data to characterize modernization projects, with information such as project scale and the time manual modernization usually takes for those types of projects.

3.2.1 Informal meetings

Informal meetings helped establish contact with the participants and collect background information on the projects. We also inquired on the use of models and tools to complement our observational data. These encounters where held in company's office during normal work hours. At a later stage, these informal meetings focused mainly on clarifying or elaborating on responses given during the interviews.

3.2.2 Semi-structured interviews

The semi-structured interviews where designed in accordance to [10] and [13]. The interview guideline and questionnaire we followed is available online[3]. In principle, interviews should focus more on *what* and *how* questions [10]. However, because we were also interested in the rationale behind the use of MDE technology, we also asked more *why* questions than is usually expected for this type of interview. Interviews were held during normal work hours, in a closed meeting room. They lasted around 30 minutes each. Only the interviewer and the interviewee were present in the room or via skype for the remote participant. Interviewees were free to answer the questions as they saw fit. All interviews were recorded with the interviewee's consent. The interviewer then analyzed the recordings and transcribed in writing the most relevant answers of the script.

[3]http://tinyurl.com/zk5x6uo

3.2.3 Observations

Observations focused on how engineers use models and MDE tools, relying on observational note taking, inspecting the code, testing live executions of the modernization process, and taking note of questions raised about MDE or specific tools during internal company meetings.

3.3 Threats to validity

We are part of the collaborating effort to further and improve the integration of MDE in the company. Also data collection took place after we started the collaboration. Therefore, a threat to internal validity is that there may be some positive bias towards the successful integration of MDE in the company. We made sure to clarify to the participants that any criticism to MDE would be beneficial. We also crosschecked inconsistencies that came up during our observations.

There are several threats to external validity. It is difficult to generalize the findings of this study to other companies in the same industry or even other industries in general, due to the specific characteristics of the projects studied. However, as stated by the participants (see Section 4.1.1) the competition is using MDE, so it is most likely applicable to other companies in the same business domain. Another threat is the low number of participants interviewed. We interviewed all of the individuals engaged in MDE at the company, so within this context it was not at all possible to include more individuals. Hence, we must consider our data as a single data point, in contrast to [11] where many individuals across multiple companies were interviewed.

Finally, there is a threat to the construction of our analysis We mitigated interpretation errors and removed inconsistencies by collecting quantitative information about the artifacts and processes used. We also followed up with punctual meetings to clarify answers and to make sure we interpreted correctly the data.

4. FINDINGS

In following section we report on the analysis we gathered from the qualitative study.

4.1 How did the company get into MDE?

We want to understand how engineers in industry become familiar with an unfamiliar paradigm.

4.1.1 Discovery of MDE

Fresche Legacy discovered MDE while investigating how competing companies approached the problem of automating the

modernization of legacy systems. Some reported using the Object Management Group (OMG) standards and practices, which led to further investigation on the subject. These solutions showed great potential for code generation. This is why the company decided to start using standards and their associated tools to rely on less risky solutions.

However, they felt that compliance to the standards as well as the rigidity of some modeling languages, such as Knowledge Discovery Metamodel (KDM)[4] and GASTM, was either too constraining or not lightweight enough. Therefore, there was a need to customize the languages in order to facilitate re-engineering of the applications. MDE offered them a broader vision than what they could find directly under the OMG umbrella. The senior engineer involved in R&D who discovered about MDE started testing the technology and tools. This then propagated to other R&D engineers.

4.1.2 Self-teaching MDE

To learn about MDE, domain-specific modeling and associated processes, interviewees reported their main source of knowledge to be tool documentations and forums. Google searches on software modernization with MDE led to reading the following articles: [2, 4, 3, 6]. Apart from understanding basic concepts, most learning came from trying and applying small examples in MDE tools. Their familiarity with programming and traditional software engineering drove this pragmatic way of learning MDE and mostly to see if the technology really worked. Driven by the tools, they first noted syntactic differences with object-oriented paradigm which then progressed into capturing more subtle difference with the MDE paradigm.

4.2 How has MDE been integrated?

Integrating a new technology, and even more a new paradigm is an important investment for a company that involves risks at the technical level, but also at the social level.

4.2.1 Natural adaptation to MDE

Given their extensive software engineering experience, all participants viewed MDE as an extension of their development knowledge with new libraries and associated tools, rather than a new paradigm. Therefore, they adapted to MDE very pragmatically as if they were acquiring a new programming language.

> *"I am surprised you ask. Adapting to MDE felt completely natural, just like when I learned Java or Javascript 20 years ago."* (Interviewee 3)

Modeling is very much anchored with traditional programming and software engineering at Fresche Legacy.

4.2.2 MDE for automation, not for abstraction

The primary goal of the projects was to automate the modernization process. For the reasons stated in Section 2.2.2, the quality of the generated code, although functionally correct, is difficult to maintain for a Java programmer.

We observed that the first models created closely mirrored the structure of the Synon and RPG. Therefore, most of the complexity in transforming legacy to modern paradigm is encoded in the code generators. Instead, MDE practitioners would solve this problem using abstraction at the level of models. Given that the modernization problem involves two very different paradigms (RPG and Java), there is little abstraction concepts in common. Thus, participants felt compelled to search for a better level of abstraction, that allows independence from the source paradigm, and facilitates the generation of code and re-engineering on the target paradigm. This is what prompted our collaboration.

> *"I feel it will be easier to generate the code from a DSL than from the abstract syntax tree of RPG."* (Interviewee 1)

4.2.3 Incrementally integrating MDE

Given the commercial goals and deadlines of the industry, adoption of MDE had to be restricted. The risk of time investment needed to learn a new technology may negatively affect the time needed to be spent on clients projects. Also, the risk of completely shifting to MDE without a proof of commercial viability is too high.

The first project was used as proof of concept for migrating the Synon database, because it presented more abstractions than RPG. Given the complexity of RPG, the company decided to integrate MDE incrementally to mitigate risks. They started migrating simpler language elements first (e.g., `Entities`) and then gradually integrated more complex parts of the language (e.g., `Activities`). This allowed each engineer to invest time on MDE as needed, while being able to continue working on other projects. Currently, all engineers that were involved (part-time) with the MDE projects are now fully dedicated to further MDE projects. The integration of MDE at Fresche Legacy is has been continuously positive.

4.3 How MDE tools help the modernization?

Tools play a crucial role in MDE [16]. Therefore, we describe what tools were used, what their purpose and scope of use was, what technical limitations related to the nature of the project were encountered, and what workarounds were need to overcome these limitations.

4.3.1 Reverse engineering

As shown in Figures 1 and 2, the modernization process starts by extracting a metamodel definition of Synon and RPG, as well as a model that conforms to its metamodel representing the specific code. This reverse engineering step was performed using MoDisco [3].

The derived metamodels produced by MoDisco are very large and complex (see Table 1). They are hard to manage, have unclear definitions, and lead to unused classes in the final model. MDE tries to tame this complexity by changing the level of abstraction, which the company was not able to achieve. Furthermore, the translation of some scarce legacy languages, such as RPG, into a metamodel is not well supported in MoDisco. To solve this issue, the developers had to manually evolve the RPG metamodel extracted by MoDisco. They relied on model manipulation functionalities provided by MoDisco and an in-house RPG parser to reverse engi-

[4]http://www.omg.org/technology/kdm

neer RPG code to models. This is why they seized to use MoDisco, though it was a helpful start.

4.3.2 Refinement of the metamodels

The metamodel of Synon did not require any further modification after it was output from MoDisco. Because RPG is not supported by MoDisco, the metamodel of RPG needed to be refined through class inheritance from GASTM. Although this resulted in 477 classes (see Table 1), this metamodel simplified how the precondition of model transformation rules are specified.

The developers used OCLTools[5] for its OCL in Ecore editor that allows the manipulation of metamodels as a textual representation. This allows the developer to surpass the limitations of the base metamodel editors with features such as partial keyword search and batch manipulation of Ecore elements. It is very useful to be able to search for class, attribute and reference names and perform multi-point refactoring (among others) using text operations. Keeping in mind the size of the metamodels, these small features greatly help the productivity of the developer. Nevertheless, they did not use OCL for constraining models.

4.3.3 Code generation

The tool chosen for generating the code in Java and other languages (see Table 2) is Acceleo. While the team was refining and refactoring the templates, Acceleo was producing errors as more elements of the model were being used. Error correction and testing was performed in parallel while developing the templates.

Although non-trivial, the code generation step was noted to be the simplest endeavor for the team. They seamlessly expanded the Java code and added the other artifacts to the project, e.g., HTML.

There was no major issue with this step. The development team noticed a technical problem however. Acceleo reported errors when the templates where updated by another team member, although the templates were correct. A cleanup operation was systematically needed to clear the problem.

4.3.4 Feedback on MDE tools

The overall feeling regarding the available modeling tools, is that they satisfactorily serve their purpose, but lack maturity. They provide no significant additional usage facilities, such as accurate and direct feedback from errors, search and comparison functionalities. This reinforces the need for work in line with [5], where the user is given valuable interactive feedback about transformation errors.

> *"When you understand how these tools work, it's a lot more flexible [than doing it by hand]."* (Interviewee 2)

Another issue is the need for a support service behind tools. They would rather choose a tool that offers support, than investigating for the best tool that lacks support. Because most of the tools used are part of the Eclipse ecosystem, these projects depend heavily on the Eclipse Modeling Framework (EMF) [19]. This is also felt as a lock-down to the Eclipse platform. Any MDE tool outside this ecosystem has a much smaller community, but they feel it would be beneficial to have alternatives.

> *"[...] sometimes the plugins conflict in Eclipse, so you have to get a different Eclipse installation for each aspect of the project. EMF is so bound to eclipse you cannot go to JetBrains or other tools."* (Interviewee 3)

4.4 How this solution scales to larger projects

4.4.1 Environments with performing resources

We noted that the development systems used in the company are quite uniform, with an emphasis on large amounts of RAM: around 16 GB. Depending on the project and the client, production systems may have more resources available. However, some projects must be run on the client's premises, which may not have high performing resources. Therefore, the development environment for modernizing of these legacy systems using MDE at full scale is required to have performing resources. Once the code and process is tested, the migration must still be able to run on less performing resources that clients are restricted to. Nevertheless, the development environments perform satisfactorily for MDE-based modernization projects.

4.4.2 Performance is not a problem

When dealing with such large systems, scalability is usually an issue. However, in the context of this study, scalability of MDE is not. The productivity benefits of automatic modernization are orders of magnitude better than manual modernization (see Section 2.1.2), that the current state of scalability of MDE is satisfactory enough.

> *"But of course, any improvement on performance is welcome."* (Interviewee 1)

The development team is not worried about the performance of current MDE tools. They are so used to very large project artifacts that they were used to manage inherent scalability problems even before MDE. They believe these techniques can transcend to MDE as well. For example, in this incremental process, they separate larger models into smaller ones in order to make it work in environments with less performing resources.

> *"[internally] we can generate a subset of the application without having to parse all the files... we can produce a sub-model directly from the Synon tools, which is much easier for me to work with smaller files."* (Interviewee 3)

As far as model size is concerned, one would expect the usage of CDO[6] to store and access large models. Model scale was not a major limiting factor. They elaborated and adapted a technique using hash maps which was sufficient to increase the performance of (re)loading and searching models.

4.4.3 Robustness of technology

Unlike in manual modernization, there is no guarantee of atomicity when running code generation. In case of external (byzantine) error, we cannot ensure that at least what

[5]http://www.eclipse.org/modeling/mdt/?project=ocl

[6]http://wiki.eclipse.org/CDO

is produced is correct. This makes the execution process quite fragile, in the sense that once it is started it cannot be interrupted or resumed at a later time. This limits the use of laptops when working on clients premises, where the workstation is restricted to unnecessary up times. Pause and resume functionalities, recovery points, running as a service, and on-demand execution are desired when executed automated MDE tools, such as code generation.

5. RELATED WORK

In [6], the authors address the modernization process by translating the legacy code into ASTM. They then raise the abstraction to KDM, allowing for a round-trip development between the two levels of abstraction. The work presented here also starts by translating legacy code to an ASM using MoDisco, but when considering to raising the level of abstraction, there is no reverse relation persevered for tracing.

In [12] the authors refer versioning as a contention point of applying MDE to large scale projects. This was not noted by the participants of our study. The remaining issues raised by [12] do not appear to apply to our case, as by nature of the migration projects, Fresche Legacy already has methods and resources in place to handle scalability that translate to MDE.

The author in [9] discusses re-engeniering, like our participants, although in a different senses. The worry of re-engeniering we observed is on a client by client basis, optimizing patterns of each specific project. However, the author of [9] talks about general strategies of MDE re-engeniering. Still it is possible to place the work reported here in regards to [9] as <D2,P1> (good database with a preserved logic encapsulated with emulation) but the aim is to achieve <D2,P3> (good database with and new high quality logic) within acceptable costs.

Contrary to [11], where the authors state in 5.2.7,

> *"Organizations that consider themselves to be in the business of software engineering appear to find process change, and particularly the adoption of MDE, a bigger challenge than those for whom software development is subsidiary to some other function."*

we found that MDE can be integrated in companies where their main focus is software engineering. This might be related to the nature of the software engineering practiced in the company, as the modernization process requires engineers to think about platform specificities and how to abstract from them. Nevertheless, it shows that the barrier to entry might be more at the managerial level than at the engineering level. Another hypothesis for this discrepancy is that, in the case of Fresche Legacy, the first contact with MDE was autonomous and not through a relation with academia. We also feel that, like in [11] 5.2.2, the choice of project to introduce MDE and the motivation of the people involved is a significant contributing factor to the success of this endeavor.

The conclusions reached by [7] are in line with the responses of our interviewees in that even if manual intervention and adaptation is needed, MDE provides a clear benefit when compared to a fully manual modernization process. The issue of testing is also in alignment with [7], where it is believed that with MDE the greatest part of resources will be testing the applications. One thing not mentioned in [7] is that MDE with DSLs in particular provide an easier way to refactor the system being migrated:

> *"It will enable us to refactor the code directly [...] it will be a lot easier to have a textual representation of the code, in a new language more business oriented, then doing [the refactoring] on the code."* (Interviewee 4)

In [18], the authors refer to the maturity of MDE technology and methods as a condition for adoption. This is still a relevant issue from the feedback we gathered.

We agree with Mohagheghi et al. [14] who state that the cost and risk of adopting MDE is a barrier to adoption, instead, the integration should be incremental.

Our data suggests a divergence from [15] regarding the ease of use of the tools, as our participants found them easy to adapt to. But still the point of lack of maturity and the need for better features for managing complex models is shared with [15].

The paper [1] also reports on a case study with interviews, with about the same number of interviewees. We did not observe any institutional and cultural frictions. The way MDE was being used was very much within the company's usual methodologies, it remains to bee seen if a more profound adoption of MDE would shake the structure of the company.

The authors in [3] discuss about scalability along the same lines as our findings. However, this is not the most pressing issue with MDE according to our participants.

We also noted that none of the studies on MDE adoption in the industry work refer how the companies came to engage in MDE.

6. CONCLUSION

6.1 Summary

In this article we report on how a company in the business of modernizing legacy application started using MDE without any prior contact with academia to provide orientation on MDE. The qualitative study we conducted resulted in interesting findings, some even contradicting previous beliefs or studies on the adoption of MDE in industrial settings.

6.2 Lessons learned

Although our study reports on the situation specific to one company, we feel this is not a isolated case, especially given that competing companies were also looking into model engineering solutions. We report the lessons learned from a company that used MDE without the help of MDE experts.

Starting with standards, such as from the OMG, gives reassurance and is a useful introduction to concepts of a new paradigm. Nevertheless, it is crucial to **customize modeling languages** when focusing on a specific domain, such as re-engineering legacy software.

For a software company it is natural to view MDE technologies as another *programming paradigm*. Although this point of view enables to generate complete applications, the lack of abstraction at the model level forces manual refactoring and optimization at the code level. In contrast, MDE practitioners tend to solve problems by modeling to **abstract away from the code** and perform refactoring and optimization using model transformations. This helps to develop more reusable solutions and is less prone to errors that stem from the technological platform.

Companies in similar situations may realize the need to work on higher levels of abstraction, but still struggle to achieve it on their own. Following these two projects, since our collaboration has started, they understood that a **DSL** would provide the level of abstraction needed to bridge the gap between code models (Synon and RPG) and generated code. The goal of this DSL is to provide engineers responsible for the modernization of legacy systems with an environment that they are familiar with, while allowing them to reason at a level of abstraction higher than RPG code. This DSL can be integrated in the modernization process through the use of **model-to-model transformation**. Like for code generation, these transformations can also be automated to translate platform-specific code models (e.g., RPG models) to domain-specific models isolating the essential information of the business rules for the system being modernized encoded in a DSL.

Adapting to MDE tools and technology was natural for an experienced programmer. These developers were in general satisfied, yet not overwhelmed, with the MDE tools they used. An important concern for such companies is to invest in a tool that will provide some long term return. **MDE tool builders** may want to note that, to address this concern, companies choose a tool based on the presence and quality of an online documentation and active forums. This provides a better adaptation of its developers to the tool, shows that issues will be promptly answered and is a good indicator that the tool will be supported for a long period of time. Surprisingly, **scalability** of MDE is was not on of these concerns: the current benefits are enough to adopt MDE. However, they were surprised that MDE tools are built without using MDE technologies.

ACKNOWLEDGMENTS

This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the Engage program, award RN000434.

7. REFERENCES

[1] J. Aranda, D. Damian, and A. Borici. Transition to Model-Driven Engineering. In *Model Driven Engineering Languages and Systems*, volume 7590 of *LNCS*, pages 692–708. Springer, 2012.

[2] G. E. Boussaidi et al. Reconstructing Architectural Views from Legacy Systems. In *Working Conference on Reverse Engineering*, pages 345–354, 2012.

[3] H. Brunelière et al. MoDisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[4] H. Brunelière et al. Software Modernization Revisited: Challenges and Prospects. *IEEE Computer*, 48(8):76–80, 2015.

[5] J. S. Cuadrado, E. Guerra, and J. de Lara. Quick fixing ATL model transformations. In *Model Driven Engineering Languages and Systems*, pages 146–155. IEEE, 2015.

[6] G. Deltombe, O. L. Goaer, and F. Barbier. Bridging KDM and ASTM for Model-Driven Software Modernization. In *Software Engineering and Knowledge Engineering*, pages 517–524, 2012.

[7] F. Fleurey et al. Model-driven Engineering for Software Migration in a Large Industrial Context. In *Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 482–497. Springer-Verlag, 2007.

[8] A. Gupta. Architected RAD: Tackling the challenges of on demand business. In *The Rational Edge: e-zine for the Rational community*. IBM, 2003.

[9] J.-L. Hainaut et al. *Software Evolution*, book section Migration of Legacy Information Systems, pages 105–138. Springer, 2008.

[10] S. E. Hove and B. Anda. Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. In *Software Metrics Symposium*, pages 10–23. IEEE, 2005.

[11] J. Hutchinson et al. Empirical assessment of MDE in industry. In *International Conference on Software Engineering*, pages 471–480. ACM, 2011.

[12] V. Kulkarni, S. Reddy, and A. Rajbhoj. Scaling Up Model Driven Engineering – Experience and Lessons Learnt. In *MODELS*, volume 6395 of *LNCS*, pages 331–345. Springer, 2010.

[13] S. Merriam and E. Tisdell. *Qualitative Research: A Guide to Design and Implementation*. Jossey-Bass higher and adult education series. Wiley, 2015.

[14] P. Mohagheghi et al. MDE Adoption in Industry: Challenges and Success Criteria. In *Models in Software Engineering*, volume 5421 of *LNCS*, pages 54–59. Springer, 2009.

[15] P. Mohagheghi et al. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering*, 18(1):89–116, 2012.

[16] G. Mussbacher et al. The Relevance of Model-Driven Engineering Thirty Years from Now. In *Model Driven Engineering Languages and Systems*, volume 8767 of *LNCS*, pages 183–200. Springer, 2014.

[17] J. Porter. *Synon developer's guide for the AS/400*. McGraw-Hill, 1995.

[18] M. Staron. Adopting Model Driven Software Development in Industry – A Case Study at Two Companies. In *Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 57–72. Springer, 2006.

[19] D. Steinberg et al. *EMF: Eclipse Modeling Framework*. Eclipse Series. Addison-Wesley, 2nd edition edition, 2008.

[20] J. Yaeger. *Programming in RPG/400*. Duke Press, 1995.