Computing and Informatics, Vol. 35, 2016, 299–337

COMBINING THE CONTINUOUS INTEGRATION PRACTICE AND THE MODEL-DRIVEN ENGINEERING APPROACH

Vicente García-Díaz, Jordán Pascual Espada
Edward Rolando Núnez-Valdéz, B. Cristina Pelayo G-Bustelo
Juan Manuel Cueva Lovelle

University of Oviedo
Department of Computer Science
C/ Calvo Sotelo
Oviedo, Spain
e-mail: garciavicente@uniovi.es

Abstract. The software development approach called model-driven engineering has become increasingly widespread. The continuous integration practice has also been gaining the importance. Some works have shown that both can improve the software development process. The problem is that the model-driven engineering is still a very active research topic lacking its maturity, what translates into difficulties in optimal incorporation of the continuous integration practice in the process. We present an experience report in which we show the problems we have detected in a real project and how we have solved them. Thus, we increase the productivity of development and the non-technical people are able to modify already deployed applications. Finally, we incorporate an evaluation that shows the benefits of the proposed union.

Keywords: Model-driven engineering, continuous integration, business users, domain-specific language, experience report

Mathematics Subject Classification 2010: 68U01, 68N01

1 INTRODUCTION

A common problem in computer science is the growth of software development complexity due to customer demand for more features and fewer errors [26]. Furthermore, due to continuous advancements in the technology, it has become necessary to utilize software in multiple domains and professional areas. This leads to problems such as development teams becoming experts in one particular area, necessitating an adjustment period as the team tackles new challenges in other professional areas. Additionally, there are also many technology platforms indicating that development could be greatly optimized if we could reuse not only part of the source code that is generated every day for different platforms, but also the expertise acquired by ourselves or others in a specific domain [11]. Thus, to perform the increasingly complex software development we consider hiring more staff. However, increasing our production capacity through the industrialization of software, as seen in many other sectors, may be a better solution than increasing the volume of developers [40].

Nevertheless, software engineering continually offers new tools that, when properly used, can help in the difficult task of developing software complying with the triple constraint of a project management (scope, time and cost) that is cited in numerous sources such as Frame [20] or the Project Management Institute (PMI) [51]. Thus, a relatively new software development approach called Model-Driven Engineering (MDE) [33] (a.k.a. Model-Driven Development or Model-Driven Software Development) has appeared.

MDE raises the level of abstraction of the traditional languages (e.g., C++ and Java), allowing the use of concepts close to the domains of the problems. The practice of Continuous Integration (CI) [3], which enables greater control of the project status and identifies problems and errors early in the development cycle, is also on the rise. Works such as Volter and Stahl [73] or Duvall et al. [18] have shown that both tools improve the productivity of software development and the quality of the final products, although a lack of research still remains.

The motivation for this work has its origin in a previous application of MDE [22], which minimized the development time of food traceability systems tailored to the specific needs of each customer. However, during the completion of that work we noted the absence of tools for using CI in the development process through model-driven initiatives. This observation prompted us to propose the following two tasks that could have been helpful for both the development team and the customer:

- Integrate model-driven projects: the development team has been unable to successfully carry out the practice of CI because the work was a model-driven development rather than a code-driven one.
- Allow domain experts to modify software systems: changes in models representing the domain of the problem required the regeneration of all the artifacts and the redeployment of the application; therefore, it was impossible for customers to modify software systems by themselves without any help from technical staff.

However, these tasks were not completed successfully. The overall aim of this work is to present a relation that combines CI and MDE, which is called Model-Driven Continuous Integration (MDCI). To achieve MDCI, we created a prototype for quantitatively evaluating the benefits of the approach. Thus, the two main sub-objectives of this work are as follows:

- Integrate model-driven projects: the goal is for different members of the development team to be able to properly integrate model-driven projects in a distributed and continuous way, as they do for traditional developments. This objective would have many benefits, including risk reduction, bug fixes, better relationships with customers, higher morale for the development team, improved estimations, immediate availability of the latest version of code, improved collaborative capabilities of the team, and cost reduction[1].
- Allow domain experts to modify software systems: the goal is for different experts in a domain to be able to change software models created for that domain in a distributed way, causing applications to change dynamically and transparently to the users of modeling tools. Thus, experts in the knowledge domain would be allowed to modify the deployed applications themselves to meet customer needs. This need has already been explored in works such as Volter [72]:

  > *You want to develop application logic at a higher level of abstraction than the level provided by the programming language use, for example because you want domain experts to program.*

  Due to the use of CI, the present study goes a step further; allowing actions, such as deploying the system, managing the version history of system models, generating change reports, notifying third parties, or creating tests, to be performed in a manner that is transparent to domain experts.

The remainder of this paper is structured as follows: Section 2 presents an overview of MDE and CI concepts; Section 3 discusses some problems found in achieving our objectives; Section 4 shows the proposed solution that led to the MDCI prototype; Section 5 offers a quantitative evaluation through three case studies; finally, Section 6 indicates our conclusions and outlines the future work to be done.

2 STATE OF THE ART

2.1 Model-Driven Engineering

The history of software development is inevitably related to different generations of programming languages, for instance, machine language, assembly language, procedural languages, object-oriented languages and, according to some authors, aspect-oriented languages, the latest generation of languages until the appearance of MDE.

[1] Fowler, Martin. *Continuous Integration* `http://martinfowler.com/articles/continuousIntegration.html` (February 10, 2012)

Each new generation of languages increases the level of abstraction of their concepts, thereby increasing the productivity of development. This is because, in addition to using terms closer to the way humans communicate, one can use more sophisticated instructions. The last great leap for increasing productivity and quality of software development via increases in the level of abstraction was the occurrence of MDE.

MDE is a paradigm that elevates models to first-class citizens in the field of Software Engineering [37]. It is based on the separation of the functionality of the system and the development of such a system for a specific platform, that is, it seeks to clearly separate the analysis and design of the programming. Models are used to achieve this goal [58]. A model is a set of formal elements that describes something that is being developed for a specific purpose; these elements can be analyzed using several methods [41]. From these models, transformations are typically performed to automatically generate artifacts in a lower level of abstraction (e.g., source code of a programming language such as Java or C#, documentation, tests, etc. [73]).

According to Selic [59], there are two types of complexities in the software development. First, essential complexity is inevitable and is directly related to the problem to be solved. Second, arbitrary complexity is due to the tools and methods used during development. MDE serves to decrease the arbitrary complexity, raising the level of abstraction and avoiding lexical, syntactic and semantic problems with different programming languages that exist and will exist in the future.

The use of MDE also has a cost. It is not always profitable for a company to make the extra effort required to use models, generally through the use of a textual or graphical Domain-Specific Language (DSL) [5]. Thus, some authors recommend the use of MDE in the so-called software product lines [71] and give, as a reference point, three developments (or even less) [57] to obtain a positive Return On Investment (ROI). Some works indicate that the reason why there is still a lack of adoption by industry is a poor tool support. However, other works say that problems have to do with social and organizational factors [75] to achieve greater adoption and there is a need for a progressive change to adopt MDE successfully [29].

2.2 Continuous Integration

CI is a software development practice by which different members of a development team integrate their work frequently to get a full or partially full work, usually once per day [28]. Each integration is performed with an automatic build of software, generally accompanied by unit, quality, accessibility or coverage tests, to detect errors as quickly as possible. It also can generate reports to show the results of each integration.

The main advantage of this practice is the reduction of integration problems and greater cohesion in software development, thus increasing software quality and reducing risks [18]. Authors such as McConnell [39], Olsson [49] or Ebert et al. [19] have also noted many advantages. With regard to costs, there are many economic data justifying the use of CI. Thus, Boehm said that the cost of removing a software defect grows exponentially at each stage in which it was not detected [7]. In addi-

tion, other studies have confirmed this theory. In Sharpe [60], it was explained how the cost of repairing a bug is a dollar when the software is being developed. After integration with the rest of code, its correction will cost more than a hundred dollars and when the software has already been distributed, the correction will cost thousands of dollars. Meanwhile, a study by the National Institute of Standards and Technology (NIST) has shown that software bugs make the United States spend $ 60 million each year [66]. NIST also has discovered that almost 80 % of the total projects cost is intended to correct their faults.

CI is a practice recommended by many software methodologies. For example, it was picked as one of the 12 original practices of Extreme Programming (XP) [3] and is part of the Unified Process (UP) recommendations [30]. It is like a member of the development team that is responsible for monitoring the source code, compiling each change, testing the code and notifying the team of any problems that occurred during the process [53]. In the absence of real case studies validating the benefits offered by CI, works like Ståhl and Bosch [64] have appeared exploring the benefits from projects in a large company.

Finally, it is important to note that some authors such as Olsson [49] have also mentioned the disadvantages of CI, though they are almost negligible in relation to the benefits offered. These disadvantages are mainly the possible need for a CI-specific server or the need to train developers to upload the code correctly. In addition, it is difficult to adopt CI with legacy systems since those systems usually are not properly designed to be integrated with some tools on which continuous integration relies (e.g. unit or integration tests) to provide feedback on the construction of the system. However, this would not be a problem in this work because when a system is developed using an MDE approach it is usually re-engineered [54] or started from scratch [21].

Figure 1 shows an overview of how CI is carried out:

1. The idea is that different members of a development team work against a repository managed by a Version Control System (VCS) [62] which facilitates collaborative and distributed works.
2. The CI tool is waiting for changes in the application repository.
3. There can be different types of triggers that make the CI tool check the state of the repository (e.g. a time interval, an exact date, the success of a compilation of another program).
4. When changes in the repository are detected, various tasks are undertaken to build artifacts through automated scripts by using tools such as Ant[2]:
   - During the compilation phase libraries or applications are generated.
   - During the testing phase tests on the source code or on the application are conducted.

[2] The Apache Software Foundation. *The Apache Ant Project* `http://ant.apache.org/` (December 19, 2012)

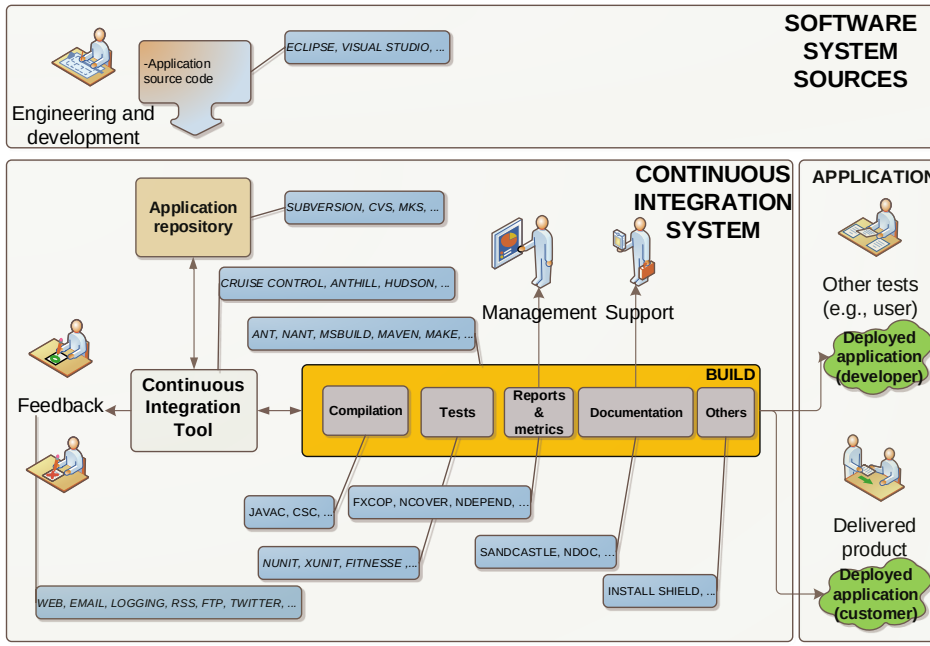


Figure 1. Continuous Integration

- During the reporting and metrics phase studies are tracked. For example, the percentage of code coverage is monitored.
- During the documentation phase help files and documentation from the comments made in the code are created.
- There may be other steps such as the construction of an installer for the application.

5. If there is any mistake or problem during artifact generation, the tool would report to the CI responsible using some system (e.g. Web, email, RSS). This mistake or problem would be corrected as soon as possible, thus re-launching the CI process.

6. If generation of the artifacts works properly, the output of the reports and metrics phase could be sent to the management team and the output of the documentation phase could be sent to a hypothetical support team.

7. Assuming that the construction has been successful, the software is deployed in a computer where even more tests could be run since not all tests can be automated (e.g. some usability tests).

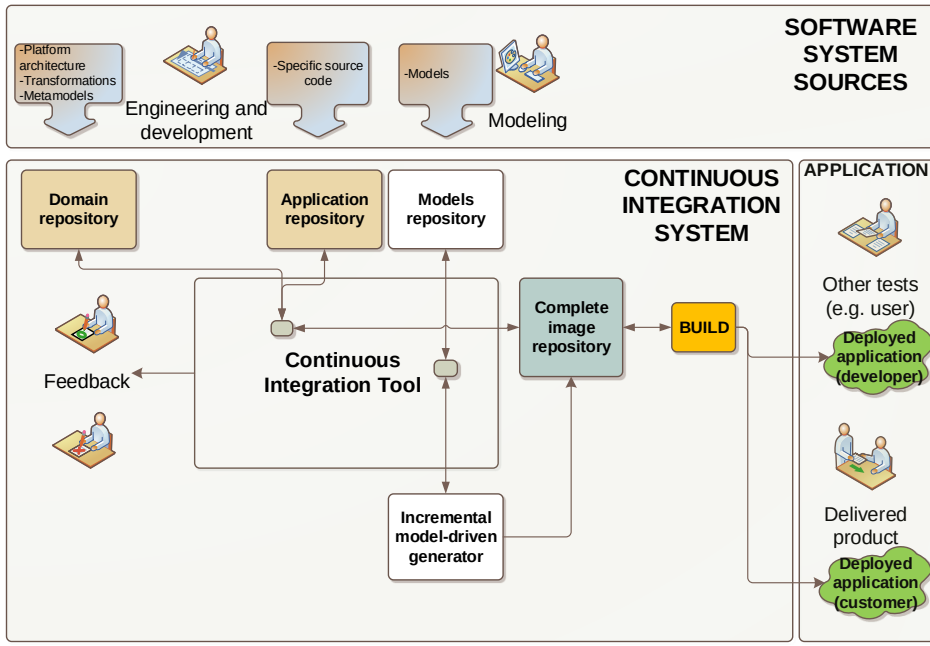


Figure 2. Model-Driven Continuous Integration

3 PROBLEMS IN ACHIEVING THE OBJECTIVES

On the one hand, CI tools are becoming increasingly popular as companies become more aware of the many benefits of successfully integrating code, such as risk and cost reduction [18].

On the other hand, companies want to reduce costs and create quality products; they do not want to *reinvent the wheel* during every development. For this reason, the model-driven development approach is gaining popularity [23].

In this paper, we create an association between MDE and CI that allows for study of the direct relationship between the two concepts. Figure 2 shows an overview of how MDCI is carried out. Two differences from a traditional case could be as follows:

- Although CI tools often provide support for multiple repositories, usually only one is used or needed. We use three repositories, one of them specific for models, since MDE is usually performed using a distinction between the following three types of artifacts [68]:

  1. artifacts common to the entire product family [14] that can use a shared repository for all the products from the same family,

2. artifacts that are specific to a particular product and therefore use their own repository, and
3. artifacts that are models used to generate other artifacts and therefore also use their own model-oriented repository [48].

- An incremental model-driven generator will be used. It will be integrated with the CI tool to generate artifacts when necessary. This will be done using extension points of a large number of CI tools.

Current MDE technologies are not fully capable of allowing CI tools to function efficiently in conjunction with model-driven development. The difficulties are mainly due to the fact that MDE is still under assessment and its maturity has still to be achieved. In the following section, we outline some of the problems encountered and the necessary changes that were necessary to develop an integrated approach. The work is divided into partial objectives to facilitate tracking of progress and identification of benefits. There are still few studies that relate MDE to CI. For example Szabo and Chen [65] mention the need for them to use a CI server in future versions of their model-driven approach. Rumpe [55] explains that we should be able to organize our model-driven project in a way that the components can be developed independently, but ideally with a CI tool. However, we have not found practical works with a proposal to make a real integration between MDE and CI.

3.1 Selection of an MDE Initiative

It is possible to develop software systems using the MDE paradigm through the use of similar, yet technologically different, initiatives. For example, Model-Driven Architecture (MDA) [43] is based on the use of standards. Other initiatives, such as Software Factories (SFs) [24], are more productive due to the benefit of a greater integration with the target platform and the possibility of using a higher level of abstraction [68], but generally do not use standards.

The first issue we found was that version control systems intended for models and generators of MDE artifacts, which can be used in a CI process, use many different technologies. Thus, we wanted to use standards to promote portability, reusability and interoperability, yet we also wanted to benefit from using other approaches. Therefore, we need to reach a compromise that favors the use of standards while allowing the use of DSLs that are not based on standards. It is important because even when MDE is becoming increasingly important, immaturity in MDE due to for example lack of quality and integrated toolsets [69] makes the way to be followed in the future unclear.

Through the use of this strategy most available model-related tools, both standard and non-standard based, could be used.

3.2 Version Control Systems for Models

As software is developed, it is very common that different versions of software are deployed on different computers while developers are working simultaneously on further improvements, updates or corrections of errors. Therefore, it is very important to have a fully localized version history so that the contents of each version are clear. Moreover, for large software developments, teamwork is essential and requires a system capable of managing the changes made by different members of the development team in a transparent manner; keeping a record of who made each change. VCSs [62] have evolved greatly over the years to meet this need.

It is difficult to apply VCSs to models because there are still some unsolved problems [2, 44, 52]. For example, current VCSs do not properly detect when there is a new version of a model due to its internal graph structure [34]. In fact, there is no approach to compute the differences between graph-like models yet. As a result, no general model version control system exists today [42]. Thus, CI tools working with current VCSs are not suitable for working with models. It is necessary to use a specific tool for models; that is a Model Version Control System (MVCS) to make working with CI tools more efficient. Some other problems related to MVCS (e.g., unsupportive conflict resolution or inflexible VCS), although very important and interesting, are out of the scope of this work.

A MVCS would reduce the number of times the CI tool identifies false changes or no changes in input models manipulated by users of modeling tools. This feature would allow the CI tool to run scripts only when necessary.

3.3 User-Friendly and Uniform Interface

User interfaces are very important in today's software and are the focus of numerous research studies. The goal of these studies is to find a way to display information in an effective and user-friendly way [61]. Similarly, CI tools typically incorporate a module or dashboard that provides the user with feedback regarding the ongoing background integration activities.

By default, CI tools are not designed to work with model-driven artifact generators. However, most CI tools can run command-line applications, allowing the use of generators with command-line support to easily obtain artifacts. In this way, we could not directly interpret the generated output in a user-friendly and homogeneous manner. Also, feedback from generations could not be distributed by the CI tool in an appropriate manner. To achieve a good user experience, we need to integrate the model-driven generator and the CI tool using extension points.

The benefit of this approach would be that feedback from the execution of the generator would be attainable in a user-friendly manner. Thus, we may distinguish, in a user-friendly and uniform interface (CI tool), whether or not the operation has been successful, what artifacts have been generated, and whether the generator provided any relevant information. The CI tool could then display information using graphical interfaces or notify those responsible for incidents.

3.4 Incremental Generation of Artifacts

Traditional incremental compilers, such as the incremental Java compiler that is built into the Eclipse Platform[3], provide the following benefits that could be extrapolated to MDE:

- Avoid having to rebuild the whole application with each change, which prevents lost time and increases efficiency.
- Allow almost instantaneous re-compiling when code changes are small.
- Reduce the granularity of compilation units while keeping the semantics of the language.
- Avoid complete recompiles guided by the developer, which leads to a more interactive development environment because the developer can learn from errors almost instantaneously.

However, current model-driven generators require that each artifact is always regenerated, except for very limited cases with specific tools, such as OptimalJ [15]. To combat this issue, we need to make the generators work incrementally, generating only the minimum artifacts necessary for each use.

The benefit of using this incremental system in a model-driven way is a reduction in the number of artifacts that are regenerated after modifications in the models. It is important to note that the higher the number of artifacts regenerated, the greater the probability of having to recompile code, redo tests, or re-analyze the output. In other words, the higher the number of artifacts regenerated the greater the impact on the deployed system. Some works such as Schaefer [56] talk about the importance of incremental model-driven developments, although this issue is currently being investigated and there is still a clear lack of final tools.

4 PROPOSED SOLUTION

We encountered several difficulties implementing the proposed architecture mainly due to the lack of appropriate MDE tools. The work completed to carry out the objectives is discussed below. This work is divided into four partial goals that will allow a clearer overview of the work.

4.1 Selection of an MDE Initiative

Works like Kelly and Tolvanen [32] or Cook [16] have shown that using only MDA it is difficult to build complete software solutions for problems beyond a certain level of complexity. However, although other initiatives such as SFs try to create complete solutions, they pay the price of having to focus on a specific architecture, losing the portability and interoperability offered by MDA [73].

[3] The Eclipse Foundation. `http://www.eclipse.org/` (January 14, 2013)

To solve this problem, we draw upon our previous work. In that study, we proposed TALISMAN MDE (TMDE), which is novel in its use of a mixture of principles to achieve maximum productivity with maximum possible interoperability, portability and reusability. For current problems, we believed that the most reasonable approach was a mix of the two main existing proposals, MDA and SFs, in which we could generate a fixed part of the system using best practices for one specific platform and generate other parts of the system through the four-layer MDA architecture [43] from a DSL, perhaps of a higher level of abstraction. This mix was viewed as the way to achieve the best possible features using the minimum amount of source code. To achieve this hybrid system, we used bridges between metamodels, as evaluated in Tolosa [67]. In Bezivin et al. [9, 10], further details and examples of bridges between metamodels can be seen. These bridges were used so that in the future someone could easily migrate the system to another platform. See García-Díaz et al. [21] for further information.

The proposed solution can be seen as a way to avoid having to restrict the use of certain tools. For example, if the development team designs a DSL using the DSL Tools [17], then the system would make a transformation from the metamodel used by the tool to the standard metamodel, that is, the Meta-Object Facility (MOF) [50], leaving the rest of the prototype intact.

4.2 Version Control Systems for Models

Although there are several proposals that address the need for version control systems for models, such as Murta et al. [44] or Altmanninger [1], there is no complete software that can implement these proposals. To adapt CI systems to MVCS, there are two main issues that must be addressed. First, there is no complete implementation of a functional MVCS, and second, there is obviously no CI tool that offers support for MVCSs.

To achieve the aims of this work, it was necessary to have a MVCS. This need led to the use of a prototype or to the development of a tool that simulates the behavior of the proposals made by the most relevant authors in this area. The idea, based on the problems cited in Oliveira et al. [48], was to decide in each commitment of the sources the unit of versioning (UV)[4] that would yield the maximum flexibility in testing. According to Oliveira et al., choosing the UV in each case is very interesting from a semantic point of view because for every problem in each particular application domain we may be interested in increasing or decreasing the granularity with which it is decided whether a change in a model can be seen as a new version of that model. The importance here is to make testing and code generation only when strictly necessary for each case, thus reducing the impact of changes. Of course, the granularity with which it performs depends on many factors such as the ability of the developers or the time available to adjust the settings, but the idea would still be trying not to redo things if they are not strictly necessary. Models are stored in

[4] The UV is the smallest unit used to consider that there is a new version in the model

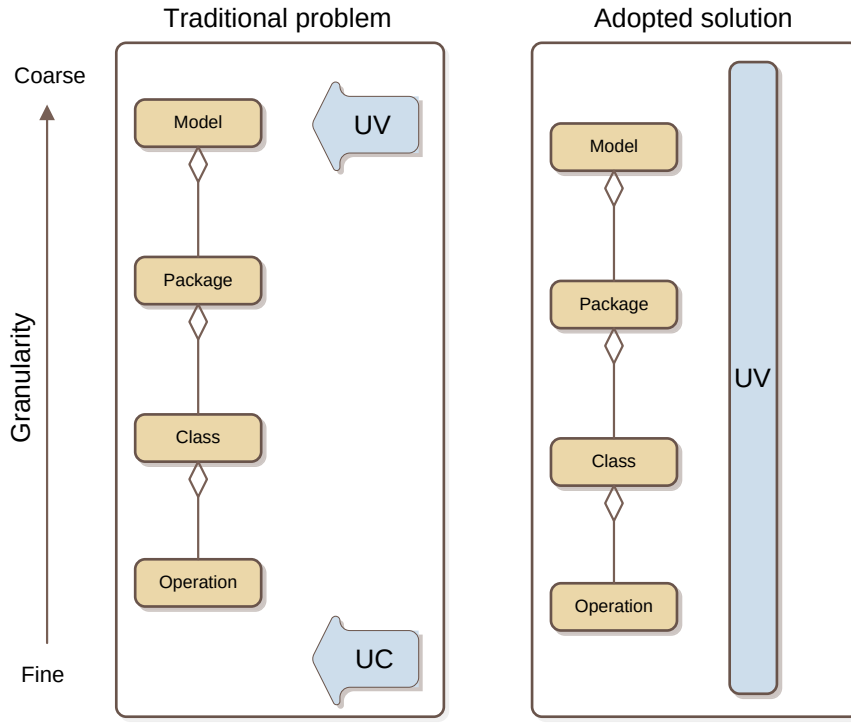


Figure 3. Unit of versioning and unit comparison for models

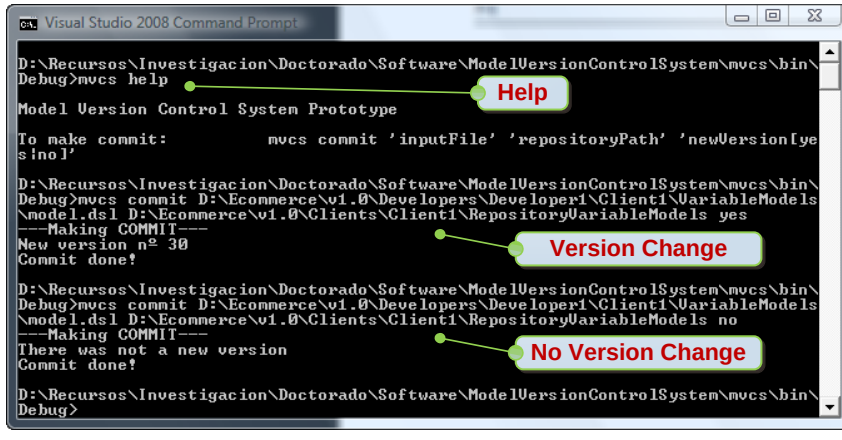


Figure 4. Version control system for models (prototype)

a repository that is checked by the CI tool. Thus, when the tool detects a new version of the model, it launches the build step. Figure 3, based mainly on the ideas of Oliveira et al. [48], shows that with the developed prototype we can freely configure the UV for each case. Thus maximum interaction with CI tools is achieved. That way we take the control of exactly when a new version of a model is introduced in the repository, being essential to perform comparable experiments. The proposed software meets only the minimum requirements for building and testing the final prototype and should not be used for other purposes. In addition, there are many open research lines related to version control systems that are outside the scope of our work. For example, we did not consider the unit of comparison (UC)[5], which is treated in Reiter et al. [52], because it is oriented on collaborative work when using optimistic source management. With the optimistic approach the MVCS does not lock the files when a user is making changes on them. For this reason those systems are more complex, since they need to compare files and perform merges when necessary to prevent inconsistencies. That speeds up the development process and encourages developers to take greater responsibility for the source code. Figure 4 shows examples of how the developed prototype works.

The MVCS prototype of this study correctly identifies new versions of a model, avoiding both false positives and false negatives. Thus, this MVCS provides a total control when changes to a model will cause the CI tool to start a new software construction phase. To allow the CI tool to work with the developed prototype, we have also developed a plug-in called *ModelVersionControl*.

To achieve the goals of the prototype, it was necessary to integrate the CI practice and the MDE development approach. This required two types of tools; a CI tool and a model-driven generator. Although there are many other tools that could have been used, we decided to use CruiseControl[6] for CI because it is a widely used tool with great support from the Open Source community. We also decided to use the Eclipse Modeling Project (EMP)[7] [25] for the MDE generator because it is the set of utilities that is closest to the Object Management Group (OMG)[8] standards.

4.3 User-Friendly and Uniform Interface

There is great interest in achieving uniform and easy to understand user interfaces. For example Neron et al. [46] emphasized achieving homogeneous user interfaces for heterogeneous programs to minimize syntactic complexity for bioinformatics applications. In addition, there is a great deal of scientific work that aims to show how to

[5] The UC is the smallest unit used to consider that there is a conflict between different changes in the model

[6] ThoughtWorks. `http://ccnet.thoughtworks.com/` (November 19, 2012)

[7] Specifically, we have relied on the tools that formerly gave rise to the openArchitectureWare (oAW) tool, now part of EMP

[8] Object Management Group. `http://www.omg.org/` (January 13, 2013)

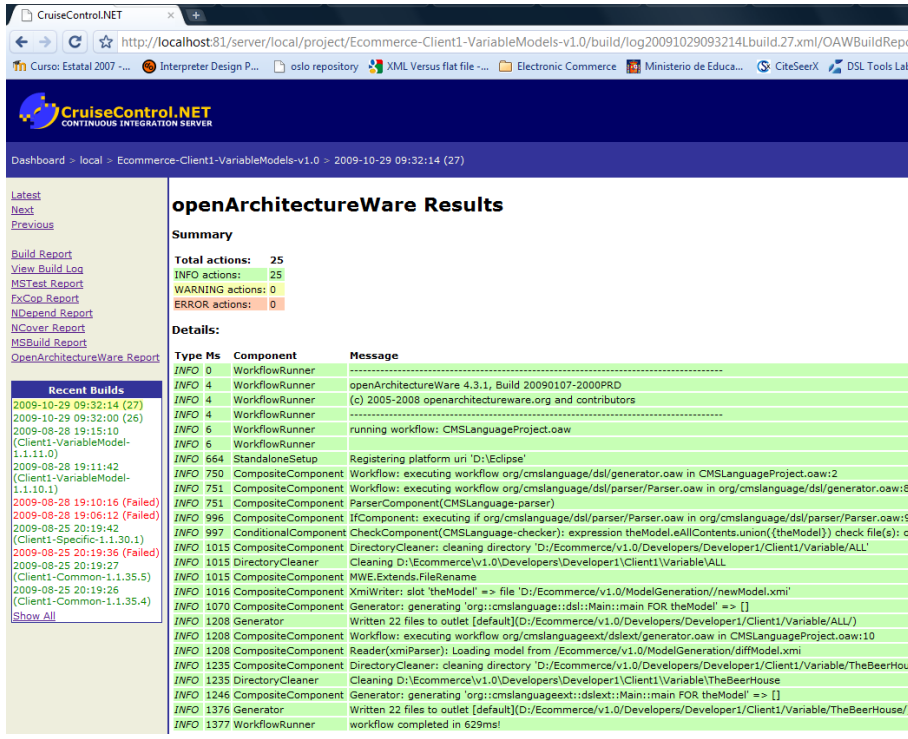


Figure 5. Feedback obtained in the generation of artifacts

build user-friendly interfaces or how to create a user-friendly interface for a specific knowledge domain (e.g., [47, 63, 74]).

Figure 5 shows how the CI tool displays feedback obtained from the artifact generator in our prototype. In this case, the feedback taken from the generator shows different types of messages: 1) information, 2) warning and 3) errors produced during generation. All the messages have a text, the name of the component that produces the message and the timespan in which the message was produced. On the left side it is possible to see all the other tools that are integrated with the CI tool and therefore can display information in addition to the results of the latest integrations made. The main idea is to use the CI tool interface to provide feedback evenly, as it would with any other feature already integrated by default. For example, the CruiseControl version we used (CruiseControl .NET or CCNET) offers native support for utilities such as NCover[9] or FxCop[10]. In this case, it is easy to provide uniform feedback because it only requires the use of the extension

[9] Gnoso. *NCover* `http://www.ncover.com/` (January 6, 2012)

[10] Microsoft Corporation. *FxCop* `http://msdn.microsoft.com/en-us/library/bb429476(VS.80).aspx` (November 6, 2012)

capabilities of CCNET to read and interpret the output provided by the generator.

In other words, it was almost only necessary to create a style sheet that processed the contents to be shown from the output format of the generator.

4.4 Incremental Generation of Artifacts

There are many model-driven generators of artifacts. Some of them are commercial and others are distributed freely; however, we found deficiencies in all of them.

- Tools with incremental generation features are integrated and very specific. An example is objectiF[11], a tool that only regenerates parts that are directly modified in the model. The shortcomings of objectiF are that it only supports changes in design time and does not take into account how a change in one part of the model could affect other parts.
- Other tools generate artifacts according to a predetermined input pattern. That means that they do not take into account the evolution of the models. For example, that is how the oAW tools work, by default.
- No tools were found to be capable of processing two input models that conform to the same metamodel, calculating the differences between the models, representing the differences in a suitable way, and generating artifacts based on their differences. For example, the proposal of Cicchetti et al. [13] is intended to represent changes, and the EMF Compare [70] is intended to compare models.

To incrementally build artifacts, it is essential to know the differences between two versions of the same model to generate the corresponding artifacts based on these differences. The problem of determining the differences, and indirectly the correlation between models, is inherently complex and involves three main steps [8]:

1. calculate differences between the models,
2. represent these differences in a format that is easy to manipulate by computer systems, and
3. visualize the differences in a human-friendly format.

The first and most crucial step, that is, calculating the differences between models, is a difficult task that has no single-best solution because it depends on the specifics of each particular problem [35]. There have been many studies related to this task. A powerful tool, called EMF Compare [70], is already included in EMP for these purposes. EMF Compare is specifically designed to compare models within the EMP, using four generic heuristics (Name, Type, Relation and Content) that provide solutions with high reliability.

[11] microTOOL. *objectiF* `http://www.microtool.de/objectiF/` (August 25, 2012)

The visualization of differences is not important with respect to incremental generation because this type of task does not require human intervention. However, the representation of differences is an important step. To avoid losing the essence of the models, our proposal is based on common features that any representation of differences between models should have [12] and is based on the technique proposed in Cicchetti et al. [13]. Given two models that conform to the same metamodel, the difference between them generates another model that conforms to another metamodel that was derived from the first metamodel. The upper right part of Figure 6 reflects this idea whose explanation can be seen in detail in Section 3 of Cicchetti et al. [12]. Thus, in Cicchetti et al. [13], the aim was to reflect three possible types of differences between a model and a previous version of that model. These differences were the possible deletion, addition, or modification of an element.

For the purposes of completing the proposed work, we developed several plugins, such as *MetamodelExt*, that can create an extended metamodel from a given metamodel in which there are new metaclasses from each original metaclass. These metaclasses are inherited from the originals and can represent the changes between two models. *ModelDiff* can represent the changes between two models that conform to the same metamodel, thus creating another model that conforms to the extended metamodel created by *MetamodelExt*. The models arising from the changes will be the first-class artifacts in the build process allowing all the elements that have been updated to be captured in a single file.

4.5 Prototype Overview

Figure 6 shows the main work that was completed in this study. The work is divided into four sections, one for each issue we addressed. To begin, we use models created with any modeling tool that could undergo a transformation if necessary. Next, the version control system for models manages the input models and detects whether there is a new version of models. Then, the CI tool also detects it causing the generation of artifacts. Feedback from the generation is shown, using a user-friendly and homogeneous interface, compared to other tools used during the process. Finally, using various available tools, such as EMF Compare, along with software specifically developed for this purpose, artifacts are built incrementally. The idea is to compare models using the EMF Compare algorithms and represent the differences using the Cicchetti et al. [13] proposal as a basis. The Xpand[12] engine is used to perform model to text transformations and to generate the incremental software.

[12] The Eclipse Foundation. *Xpand* `http://wiki.eclipse.org/Xpand/` (December 16, 2012)

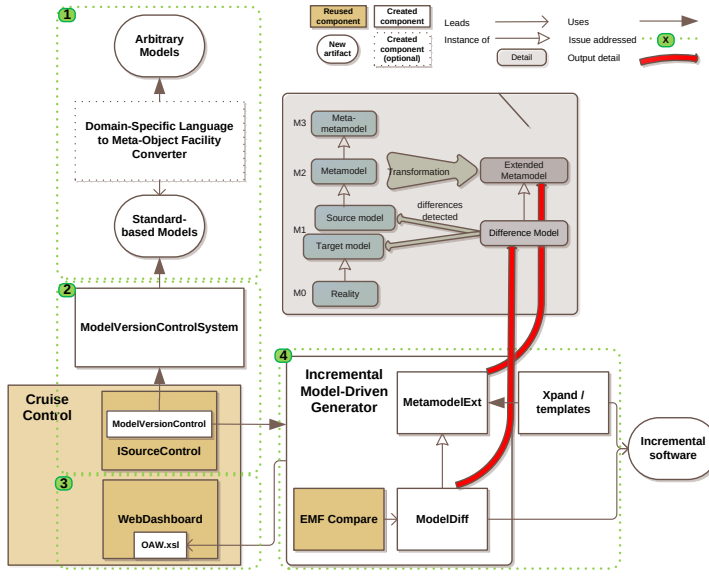


Figure 6. Prototype overview

5 QUANTITATIVE EVALUATION

When we finished the prototype, we conducted three case studies to assess the potential advantages gained by working with MDCI. The tests were designed to compare the results of the proposed method to the other methods of the study, with special interest in the traditional MDE approach, in terms of time and resources needed to develop software. For this purpose, numerical values were derived from changes in the models that guided the construction or modification of the software. These parameters corresponded to the following factors:

1. the number of generated artifacts after a change in a model,
2. the size of such artifacts,
3. the generation time for the artifacts, and
4. the deployment time in the production environment.

We used these parameters because they are numerical values that indicate areas in which the prototype may offer improvements. Thus, if the value of these parameters is low, the changes will have less impact on the final system. For example, if the number and size of the generated artifacts is very small and they have to be distributed through a network to be deployed in a final system, then the time they will take would be lower than if the number and size of the generated artifacts were greater. The same applies to the time parameter. A very small time may mean that the system impact is so minimal that end users would not even notice a loss of

service. This list of parameters is not exclusive, and other parameters may be used in future studies.

Table 1 shows the different possibilities for making a development under the MDE approach. As suggested by prior studies, such as Karlesky and Williams [31], many software developers do not currently know what a CI is, and many of those who do know what a CI is do not use CIs because working with the current tools can be quite complex. Thus, current MDE developments are usually performed with technique 1 (Table 1). However, as discussed below, the use of CI with MDE (technique 2) is very beneficial for development. The completion of the prototype for this study also allowed for an evaluation of the benefits arising from the use of techniques 3 and 4. Finally, technique 5 represents the use of MDCI.

Technique	Description
1	Only MDE
2	MDE and continuous integration
3	MDE, continuous integration and incremental generation of artifacts
4	MDE, continuous integration and model version control system
5	MDCI. Equivalent to the union of MDE, continuous integration, incremental generation of artifacts and model version control system

Table 1. Development possibilities with MDE

The first evaluation was conducted from a case study based entirely on a real application called The Beer House[13]. This is a complete starter kit to create content management systems (CMSs) and electronic commerce sites. This application can be downloaded and used freely. Furthermore, some prior studies have used it as an example (e.g., [4, 6, 38]). Therefore, we have an example of a complete and fully functional application made with the ASP.NET MVC framework[14] using the C# language and the SQL Server database management system. To complete the case study, during CI, various actions with the generated artifacts were performed using tools like NCover, FxCop, MsTest[15], NDepend[16] or NDoc[17].

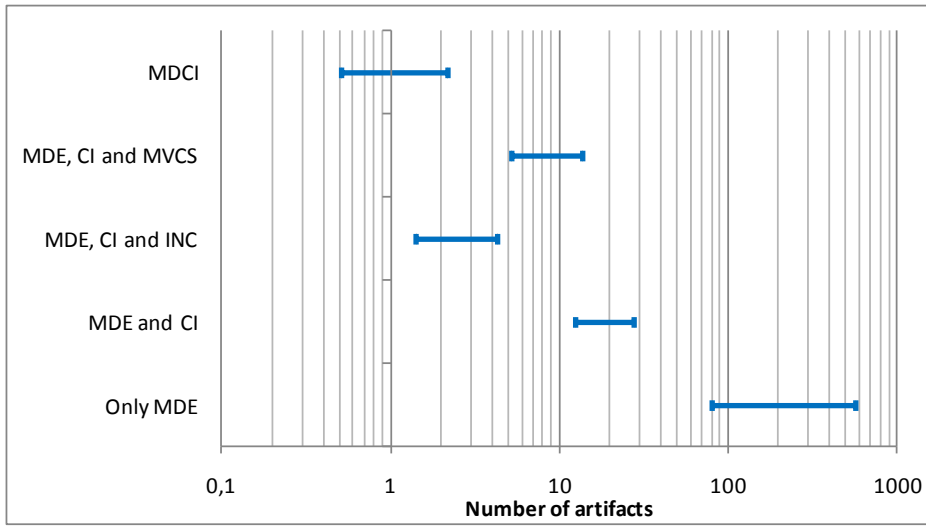
[13] *TheBeerHouse: CMS & E-Commerce Starter Kit* `http://www.codeplex.com/TheBeerHouse/` (January 14, 2013)

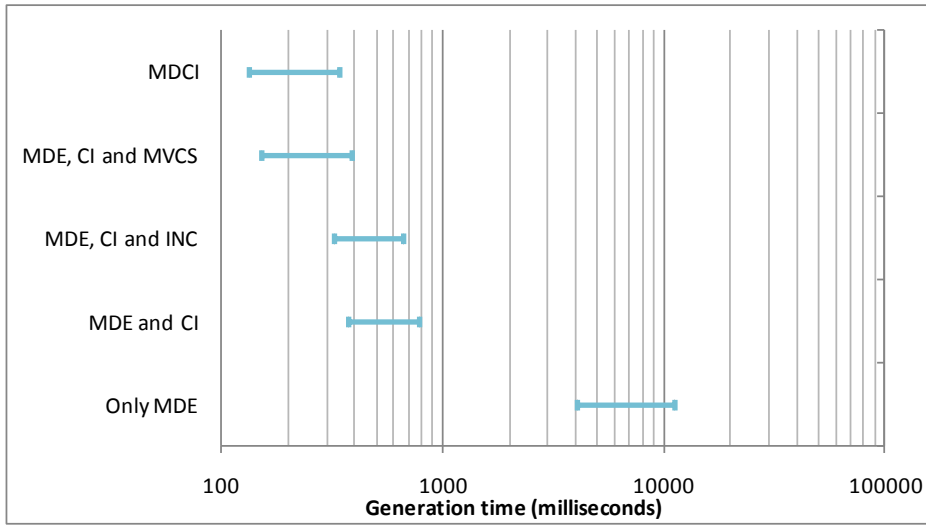
[14] Microsoft Corporation. *ASP.NET MVC* `http://www.asp.net/mvc/` (January 10, 2013)

[15] Microsoft Corporation. *MsTest* `http://msdn.microsoft.com/en-us/library/ms182486.aspx` (November 29, 2012)

[16] SMACCHIA. *NDepend* `http://www.ndepend.com/` (November 6, 2012)

[17] *NDoc* `http://ndoc.sourceforge.net/` (November 8, 2012)





The second case study was based on a real application called Simple WebSite Software (SWS)[18], which aims to provide what is necessary to build simple Web applications with a small amount of effort. SWS is created using the PHP[19] language with HTML and CSS style sheets to control the look and feel of the site. During continuous integration, files generated in each case were introduced into a Web server. This was time consuming during the deployment phase.

[18] Gliedt, Terry and the University of Michigan. *Simple WebSite Software* `http://phpsws.sourceforge.net/` (May 9, 2012)

[19] The PHP Group. *PHP* `http://www.php.net/` (January 7, 2013)

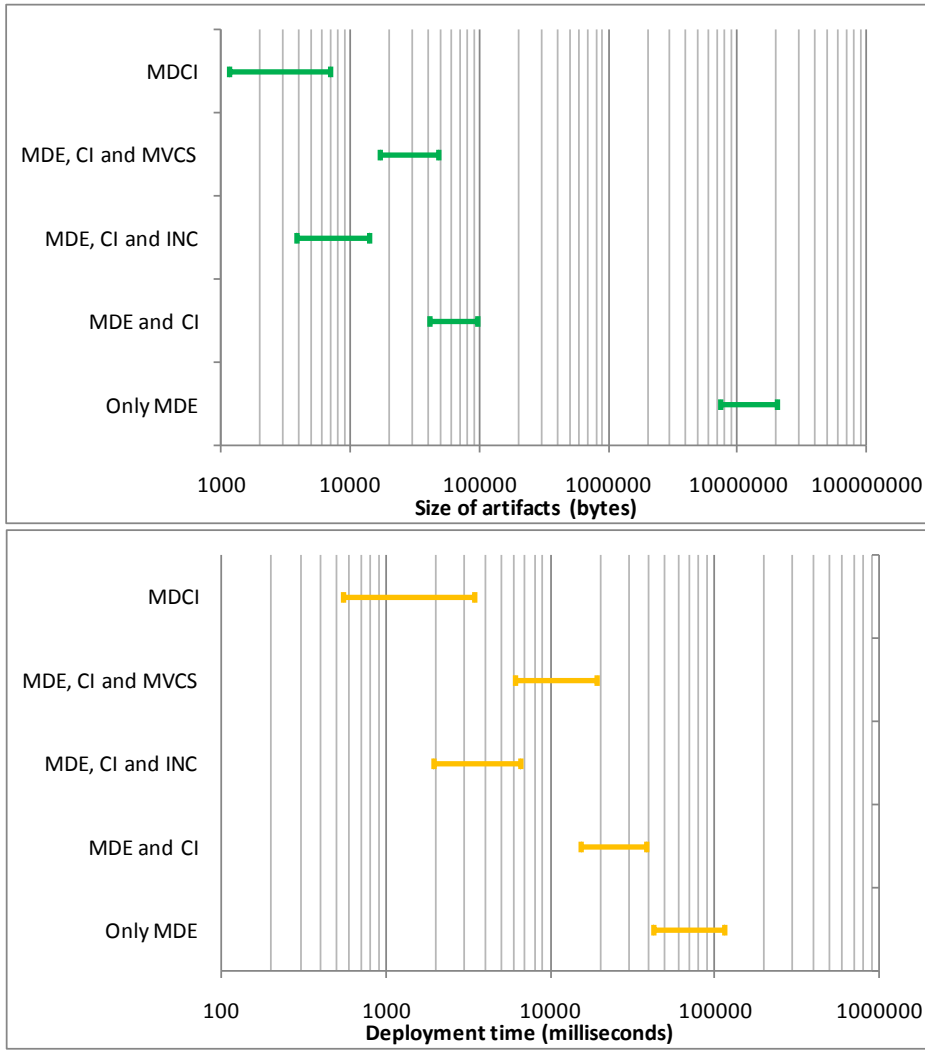


Figure 7. Confidence intervals of average values ($\alpha = 0.05$)

The third case study was based on a real application called Tweeterati[20], which is a Twitter[21] client developed using Windows Presentation Foundation [45] technology. As in the second case study, files were introduced into a Web server during integration. In this case, because the code was interpreted rather than compiled, we had to compile the regenerated assemblies when required.

[20] Singh, Arunjeet. *Tweeterati* `http://tweeterati.codeplex.com/` (April 6, 2012)
[21] Twitter. `http://www.twitter.com/` (January 19, 2013)

Figure 7 shows an interesting result inferred from the three case studies, that may be taken as a basis for further experiments or forecasts. We obtained confidence intervals of average values (with 95 % confidence) that could be obtained in the future for new case studies focusing on the number and size of the artifacts, the generation time and the deployment time. Obviously, this figure would change if more case studies were added, but after three case studies, the figure serves as a preliminary estimation of the future trend. Table 2 shows the MDCI improvement factor, which was always greater than 14 %.

Techn.	Num. arti.	Siz. arti.	Gen. time	Dep. time
1	357 216	2883 329	32 854	33 557
2	1 268	13 599	2 283	11 225
3	2	2	1 946	1 928
4	634	6 799	1 147	5 633
5	1	1	1	1

Table 2. MDCI improvement factor based on the technique

For clarity and due to word count limitations, the following sub-sections refer only to the first case study. However, as Figure 7 suggests, the results can be extrapolated to the other case studies because they had very homogeneous results.

5.1 Domain-Specific Language

To carry out the example, we built a DSL specially designed to perform the configuration tasks of the CMS. Thus, depending on user input, the generator created different configuration files and various SQL scripts that completed the application data load in the corresponding database. Thus, it was a typical example of an application of MDE, in which a user without programming knowledge was able to program in a given domain by using a DSL specific to his or her needs. Figure 8 shows, for reasons of size, only a small fragment of metamodel used for building the DSL. On it, it is possible to see the main concepts of the language such as *BasicConfig*, *UsersConfig* or *SectionsConfig*. Each of the main elements has a set of sub-elements used to configure the different aspects that the system has.

The complete metamodel acts as the abstract syntax for the DSL created using Xtext [27] as the development platform. Figure 9 shows a fragment of the grammar used to build the concrete textual syntax of the DSL. From the grammar, textual keywords for the language and the underlying metamodel can be inferred. Please, see Koster [36] to know the details of how the technologies used to implement the DSL work.

5.2 Tests Performed

To achieve the case study objectives, we made a number of sequential changes in the input model used for generating the artifacts. These changes included additions,

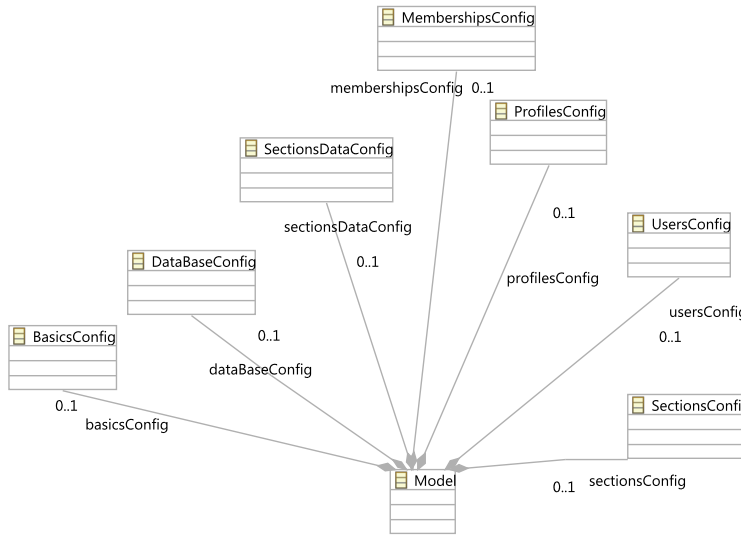


Figure 8. Capture of part of the metamodel used for the DSL

modifications or deletions of users, user profiles, articles, elements per page, categories of items, support for languages, polls, product shipments, payment methods, newsletters, and departments. It is important to note that we completed the same tests for each technique described in Table 1 to collect analyzable and comparable data.

5.3 Results

The evolution of the model led to the acquisition of 150 results for the five different techniques. The results table created has the following columns:

1. the number of test performed,
2. the technique used according to Table 1,
3. if the UV is such that it should detect that it has created a new version of the input model,
4. the number of generated artifacts,
5. the size of the sum of the artifacts generated in bytes,
6. artifact generation time in milliseconds and
7. the deployment time for the artifact production machine in milliseconds.

This table with the values is very large so in the next section we use graphics to facilitate the understanding of the results to the reader.

```
Model:
    "BASICS: " basicsConfig = BasicsConfig
    "RESOURCES: " resourcesConfig = ResourcesConfig
    "DATA BASE: " dataBaseConfig = DataBaseConfig
    "MEMBERSHIPS: " membershipsConfig = MembershipsConfig
    "PROFILES: " profilesConfig = ProfilesConfig
    "ROLES: " rolesConfig = RolesConfig
    "USERS: " usersConfig = UsersConfig
    "LOCALIZATIONS: " localizationsConfig = LocalizationsConfig
    "SECTIONS CONFIG: " sectionsConfig = SectionsConfig
    "SECTIONS DATA: " sectionsDataConfig = SectionsDataConfig
    "SMTP: " smtpConfig = SmtpConfig
    "WEB SERVICES: " webServicesConfig = WebServicesConfig;

BasicsConfig:
    "Basics:"
        "applicationName = " applicationName = STRING
        "contactForm = " contactForm = STRING;

ResourcesConfig:
    "Resources:"
        "title = " title = STRING
        "modulesCssFile = " modulesCssFile = STRING
        "siteCssFile = " siteCssFile = STRING
        "welcomeMessage = " welcomeMessage = STRING
        "copyrightMessage = " copyrightMessage = STRING;
```

Figure 9. Capture of part of the grammar used for the DSL

5.4 Analysis of Results

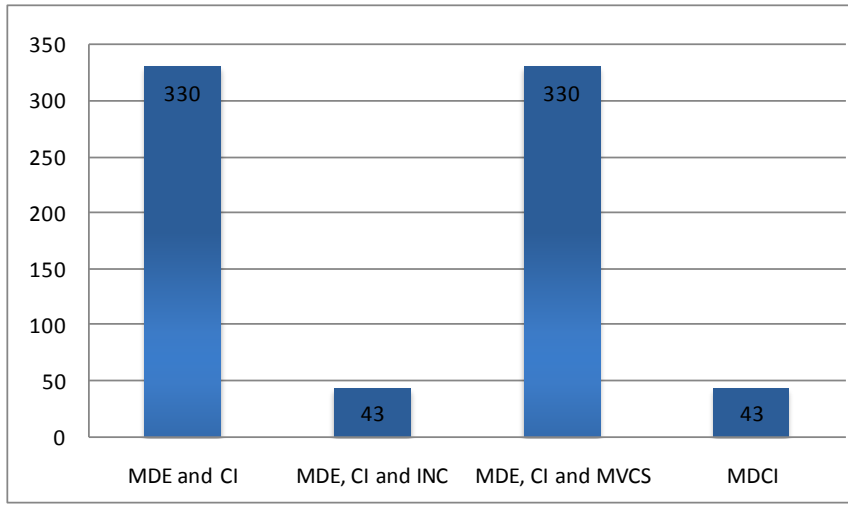
5.4.1 Number of Artifacts

The first graphs show the number of generated artifacts, depending on the input parameters. Figure 10 a) shows that when the CI tool identifies that there has been a change in the model, relevant artifacts will be generated depending on the technique used. In this case, the techniques that worked the best were 3 and 5 because they incorporated the incremental generation of artifacts.

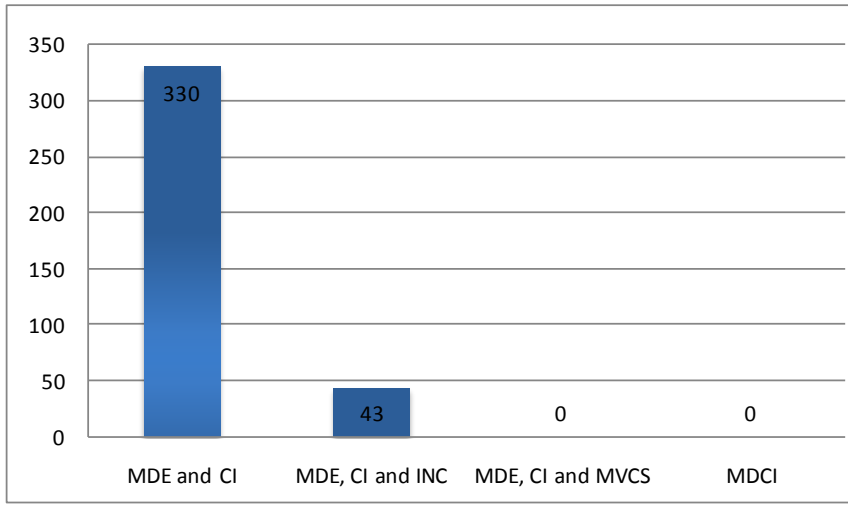
Using a MVCS (techniques 4 and 5), we detected that changes made by the user of the modeling tool did not cause a new version of the software to be generated, thus unnecessary regeneration of artifacts was avoided (Figure 10 b)). That may happen for example when the model is modified because someone thinks that it is better the way he or she thinks but it turns out to be only a syntactic change and not a semantic one. In those cases, the generator should do nothing because even when the models are different, changes do not involve a variation of the concepts.

Technique 5 behaved the best, under both criterion above, thus it is clear that with this technique the smallest number of artifacts will be generated (Figure 10 c)).

Figure 10 d) shows the number of artifacts that have been regenerated using traditional techniques instead of CI with MDE. The scale used in the graphic is much higher than in the previous cases. The reason is because working only with MDE we take into account that all the artifacts are manipulated, copied, uploaded, etc. whenever there is a change in the input model. However, with the other alternatives

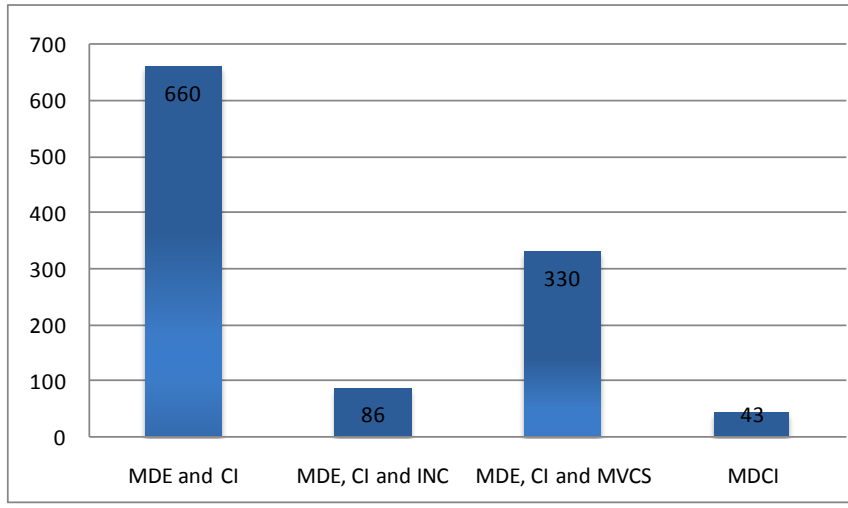


a)

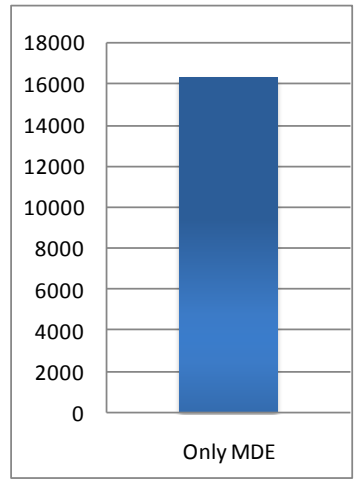


b)

we are working with the union of MDE and CI. Thus, even if there is no incremental generation of artifacts or the version control system is not specific for models, only specific related artifacts are treated. However, with a traditional approach, it is usually necessary to generate new artifacts from models and put them together with the other items, either specific for a customer or for the entire product family. Although, of course, this is not always the case and it only happens in the worst case, we only try to illustrate that using only MDE is a more hands-on approach.

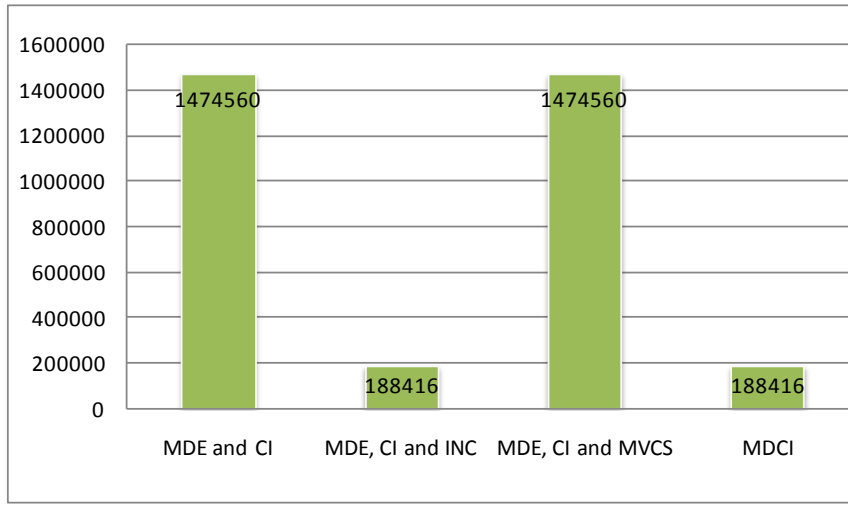


c)

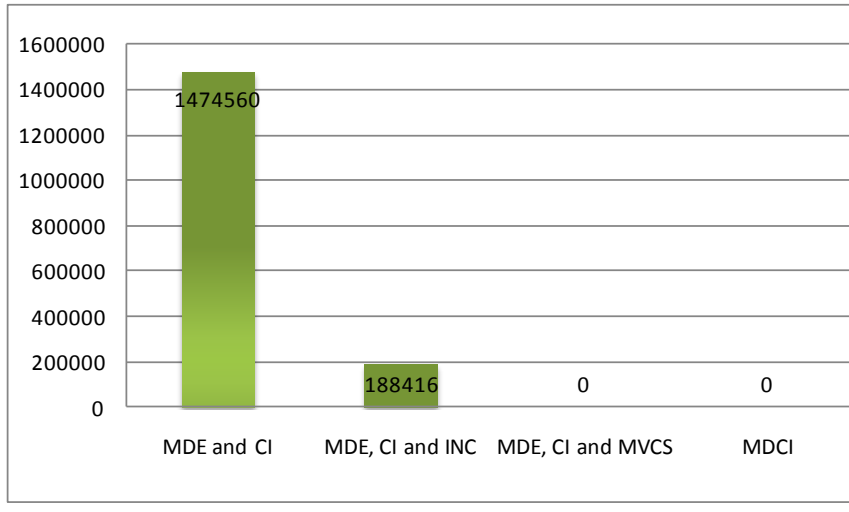


d)

Figure 10. Number of artifacts; a) Number of artifacts generated with version change in the model, b) Number of artifacts generated without version change in the model, c) Total number of artifacts generated, d) Total number of artifacts generated (without continuous integration)



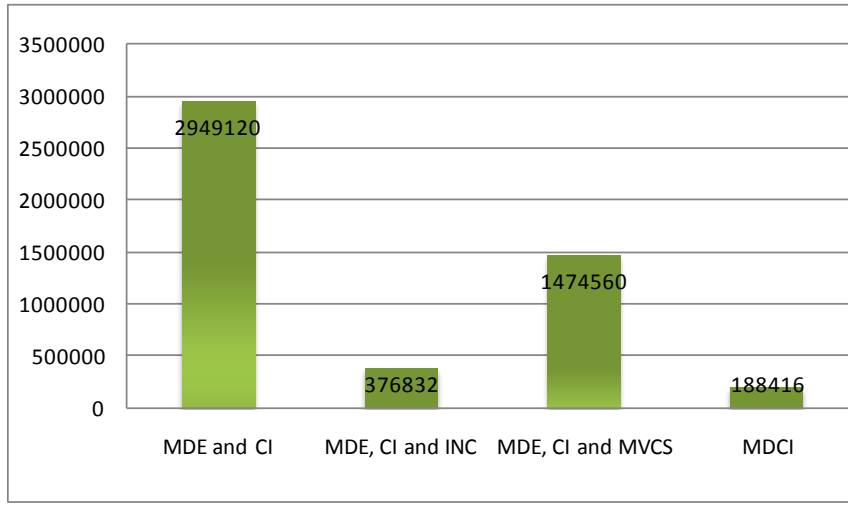
a)



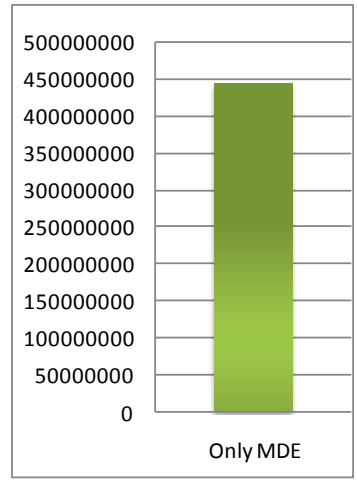
b)

5.4.2 Size of Artifacts

The next graphs show the total size of the generated artifacts, in bytes, depending on the input parameters. Figure 11 a) shows that when the CI tool identified that there had been a change in the model, artifacts were generated. As previously mentioned, techniques 3 and 5 worked the best. The results regarding the size of the artifacts in other cases (Figures 11 b), 11 c) and 11 d)) were also directly proportional to the number of artifacts in this case study.

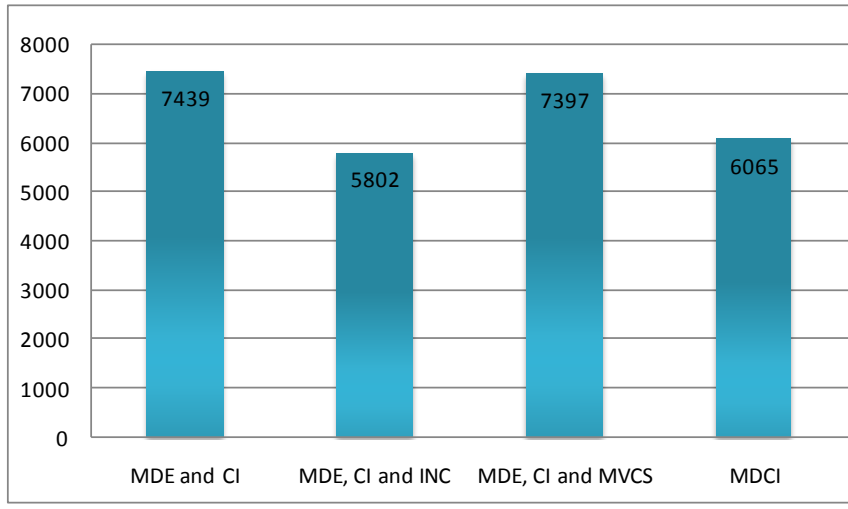


c)

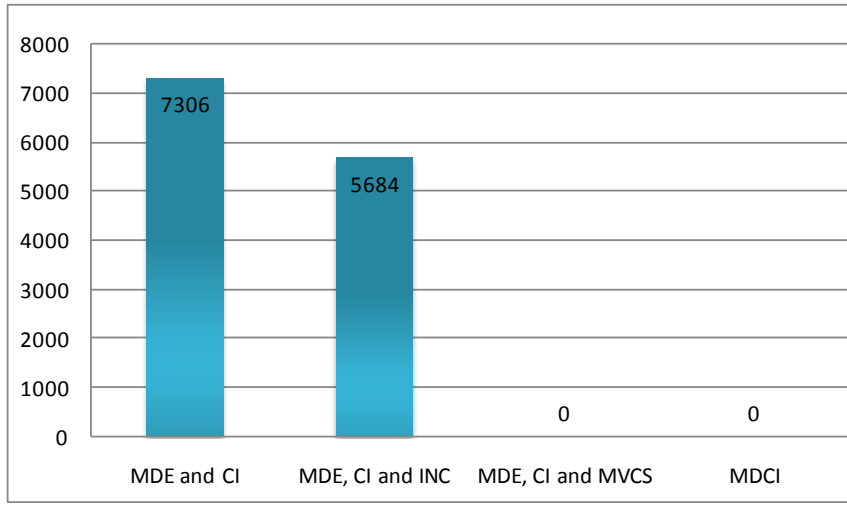


d)

Figure 11. Size of artifacts; a) Size of artifacts (bytes) with version change in the model, b) Size of artifacts (bytes) without version change in the model, c) Average size of artifacts (bytes), d) Average size of artifacts (bytes) (without continuous integration)



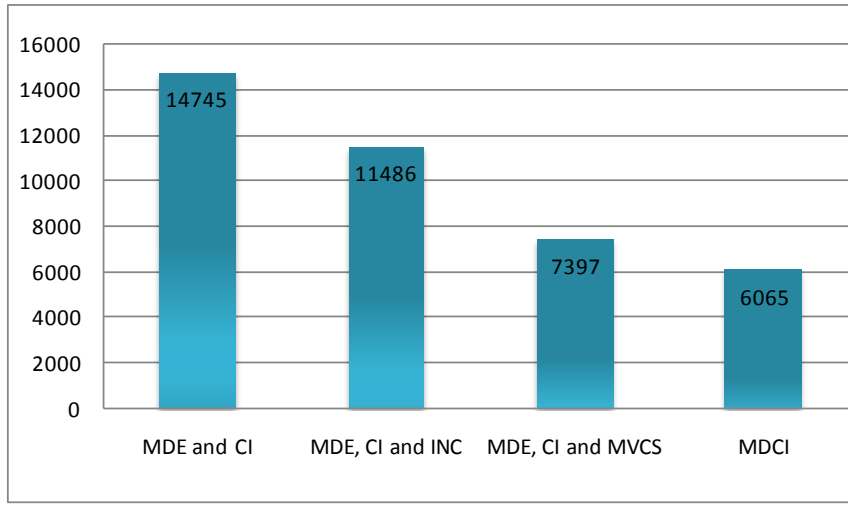
a)



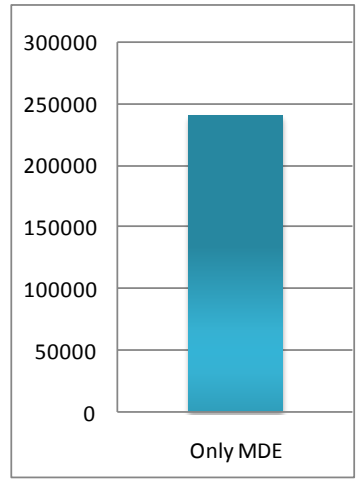
b)

5.4.3 Generation Time

The graphs in the third group show the time needed, in milliseconds, to generate the artifacts when the input models are changed. The results did not vary much as a function of this parameter (Figure 12 a)). We also observed that the results for techniques 3 and 5, while very similar, were not identical because it is highly unlikely that probabilistically different runs on a machine will result in exactly the same times. Figures 12 b), 12 c) and 12 d) show the other values obtained.

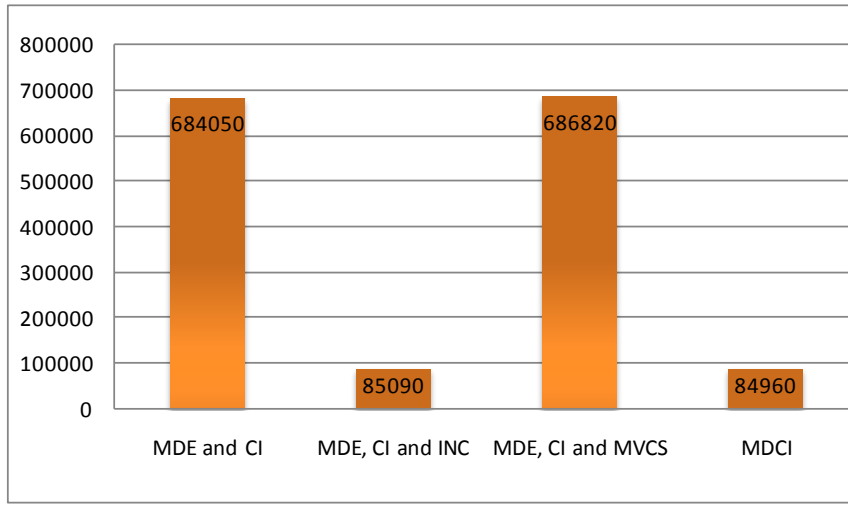


c)

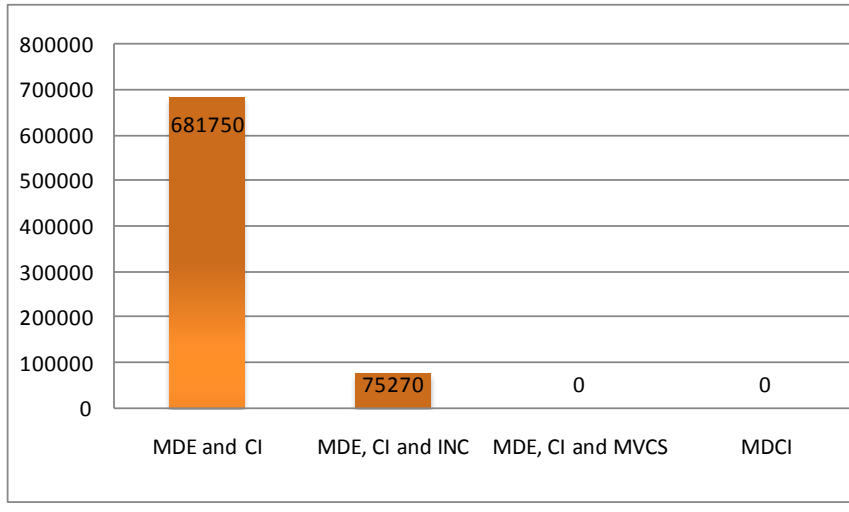


d)

Figure 12. Generation time; a) Generation time (ms) with version change in the model, b) Generation time (ms) without version change in the model, c) Average generation time (ms), d) Average generation time (ms) (without continuous integration)



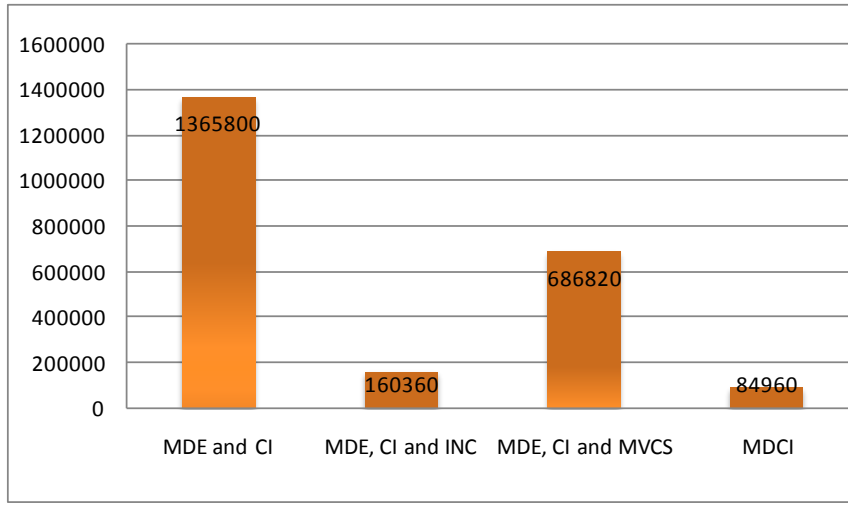
a)



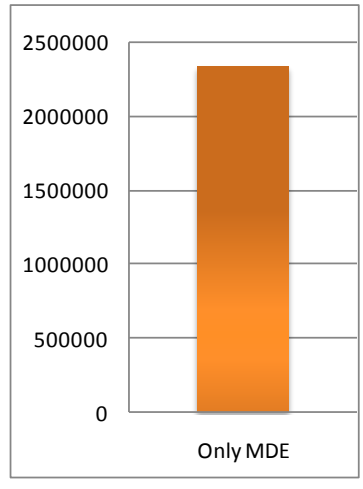
b)

5.4.4 Deployment Time

The graphs in the fourth group show the time needed, in milliseconds, to deploy the artifacts when the input models are modified. Figures 13 a), 13 b), 13 c) and 13 d) show the obtained output.



c)



d)

Figure 13. Deployment time; a) Deployment time (ms) with version change in the model, b) Deployment time (ms) without version change in the model, c) Average deployment time (ms), d) Average deployment time (ms) (without continuous integration)

5.4.5 Summary of Results

We have shown that technique 5 (MDCI) was the best in all the tests for the four output parameters. Additionally, the traditional technique performed the worst. Moreover, by incorporating CI in development, results are improved. Incorporating a MVCS or an incremental generator also improved the results, creating a better tool in either case.

6 CONCLUSIONS AND FUTURE WORK

With the goal of creating a prototype that met the objectives, we have presented various problems encountered. These problems included the following:

1. selection of an MDE initiative,
2. version control systems for models,
3. user-friendly and uniform interface, and
4. incremental generation of artifacts.

Finally, a quantitative evaluation was carried out, which suggested various numerical advantages of using MDCI rather than other types of development strategies.

In this study, our key research objectives were achieved. The prototype integrates model-driven developments in a more appropriate way. Thus, it allows domain experts to modify software systems without any help from technical staff. Traditionally, although CI has been used in a development, domain experts could not participate for themselves in the evolution of the system. By bridging the gap between MDE and CI, the domain experts that understand the semantics of a DSL [68] can also make changes in the system without many technical skills. We believe that we have fulfilled our two main objectives as follows:

- Integrate model-driven projects: with the incorporation of the continuous integration, model version control systems and incremental code generators, it is possible for different members of the development team to be able to properly integrate model-driven projects in a distributed and continuous way, as they do for traditional developments, but working in a model-driven development rather than a code-driven one. The numerical values of the results also suggest the benefits of integration reducing times for testing and generation of artifacts.
- Allow domain experts to modify software systems: with the incorporation of the continuous integration, model version control systems and incremental code generators, it is possible for different experts in a domain to be able to change software models (languages with a high level of abstraction are used) created for that domain in a distributed way, causing applications to change dynamically and transparently to the users of modeling tools. The continuous integration tool is responsible for orchestrating the entire process. Thus, changes in models representing the domain of the problem does not require the regeneration of

all the artifacts and the redeployment of the application; allowing customers to modify, in many cases, software systems by themselves without any help from technical staff.

Although some benefits have been suggested, there is still much work to be done. Based on the lessons learned in this work, we are able to describe a MDCI process that will answer the typical process related questions such as who, what, where, when, and why. In addition, the inputs and outputs, the methods, and the measurements could be described. It would also be useful to define a procedure based on this process.

The tests conducted may be sufficient to suggest the advantages provided by MDCI, but it would be interesting to conduct more case studies in different fields with the aim of performing quantitative and qualitative analyses on the differences between using traditional methods and using more advanced features like CI, MDE, MVCS, or MDCI. Other interesting topics would be the analysis of how to optimize and ensure quality of development, or how to measure the potential ROI for the industry.

Finally, it should be noted that to implement the prototype presented in this work, the simultaneous interaction of about 30 software components was required. Some of these applications were built specifically for this work. However, previously existing tools have very different origins and are designed for very heterogeneous purposes. Setting up all the tools to work properly was a very difficult task that required people with expertise in diverse technologies. Thus, it would be interesting to avoid or reduce the effort of users by standardizing the interaction between the tools.

Acknowledgements

This work has its origins in the research conducted by the University of Oviedo and Link Nmd Servicios Industriales S.L. under contract No. FUO-EM-120-07 – Software Development for the Realization of Traceability. We thank the anonymous reviewers for their constructive comments, which helped us to greatly improve the manuscript.

REFERENCES

[1] Altmanninger, K.: Models in Conflict – Towards a Semantically Enhanced Version Control System for Models. In: Giese, H. (Ed.): MoDELS Workshops. Springer, Lecture Notes in Computer Science, Vol. 5002, 2008, pp. 293–304.

[2] Altmanninger, K.—Kappel, G.—Kusel, A.—Retschitzegger, W.—Seidl, M.—Schwinger, W.—Wimmer, M.: AMOR – Towards Adaptable Model Versioning. 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with Models'08, 2008.

[3] Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, October 1999.

[4] Bellinaso, M.: ASP.NET 2.0 Website Programming: Problem – Design – Solution. Wrox, 2006.

[5] Bentley, J. L.: Little Languages. Communications of the ACM, Vol. 29, 1986, No. 8, pp. 711–721.

[6] Berardi, N.—Katawazi, A.—Bellinaso, M.: ASP.NET MVC 1.0 Website Programming: Problem – Design - Solution. Wrox, 2009.

[7] Boehm, B. W.: Software Engineering Economics. Springer, New York, USA, 2002.

[8] Brun, C.—Pierantonio, A.: Model Differences in the Eclipse Modeling Framework. UPGRADE, The European Journal for the Informatics Professional, Vol. 9, 2008, No. 2, pp. 29–34.

[9] Bézivin, J.—Brunette, C.—Chevrel, R.—Jouault, F.—Kurtev, I.: Bridging the Generic Modeling Environment and the Eclipse Modeling Framework. Proceedings of the International Workshop on Software Factories (OOPSLA '05), 2005.

[10] Bézivin, J.—Hillairet, G.—Jouault, F.—Kurtev, I.—Piers, W.: Bridging DSL Tools and the Eclipse Modeling Framework. Proceedings of the International Workshop on Software Factories (OOPSLA '05), 2005.

[11] Caldiera, G.—Basili, V. R.: Identifying and Qualifying Reusable Software Components. Computer, Vol. 24, 1991, No. 2, pp. 61–70.

[12] Cicchetti, A.—Ruscio, D. D.—Pierantonio, A.: A Metamodel Independent Approach to Difference Representation. Journal of Object Technology, Vol. 6, 2007, No. 9, pp. 165–185.

[13] Cicchetti, A.—Ruscio, D. D.—Pierantonio, A.: An ATL Based Implementation to Support a Metamodel Independent Approach to Difference Representation. Technical report, Dipartimento di Informatica, Università degli Studi dell'Aquila, 2007.

[14] Clements, P.—Northrop, L.: Software Product Lines – Practice and Patterns. Addison-Wesley, 2002.

[15] Compuware. Compuware OptimalJ: How Model-Driven Development Enhances Productivity. Technical report, Compuware Corporation, 2005.

[16] Cook, S.: Domain-Specific Modeling and Model Driven Architecture. In: Frankel, D. S., Parodi, J. (Eds.): The MDA Journal: Model Driven Architecture Straight From The Masters, Ch. 5. Meghan-Kiffer Press, Tampa, 2004.

[17] Cook, S.—Jones, G.—Kent, S.—Wills, A. C.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley, 2007.

[18] Duvall, P.—Matyas, S.—Glover, A.: Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley, 2007.

[19] Ebert, C.—Parro, C. H.—Suttels, R.—Kolarczyk, H.: Improving Validation Activities in a Global Software Development. Proceedings of the 23rd International Conference on Software Engineering (ICSE '01), IEEE Computer Society, 2001, pp. 545–554.

[20] Frame, J. D.: The New Project Management: Tools for an Age of Rapid Change, Complexity and Other Business Realities; Electronic Version. Jossey-Bass Business & Management Series. Jossey-Bass, Newark, 2002.

[21] García-Díaz, V.—Fernández-Fernández, H.—Palacios-González, E.—G-Bustelo, B. C. P.—Sanjuan-Martínez, O.—Lovelle, J. M. C.: TALISMAN MDE. Mixing MDE Principles. Journal of Systems and Software, Vol. 83, 2010, No. 7, pp. 1179–1191.

[22] García-Díaz, V.—Fernández-Fernández, H.—Palacios-González, E.—G-Bustelo, B. C. P.—Lovelle, J. M. C.: Intelligent Traceability System of Cabrales Cheese Using MDA TALISMAN. In: Arabnia, H. R., Mun, Y. (Eds.): International Conference on Artificial Intelligence (IC-AI 2008), CSREA Press, 2008, pp. 578–584.

[23] Gitzel, R.—Korthaus, A.—Schader, M.: Using Established Web Engineering Knowledge in Model-Driven Approaches. Science of Computer Programming, Vol. 66, 2007, No. 2, pp. 105–124.

[24] Greenfield, J.—Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming (OOPSLA '03), ACM, 2003, pp. 16–27.

[25] Gronback, R. C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. 1st ed. Addison-Wesley, March 2009.

[26] Groth, R.: Is the Software Industry's Productivity Declining? IEEE Software, Vol. 21, 2004, No. 6, pp. 92–94.

[27] Haase, A.—Völter, M.—Efftinge, S.—Kolb, B.: Introduction to openArchitectureWare 4.1.2. Proceedings of TOOLS EUROPE 2007 – Objects, Models, Components, Patterns, July 2007.

[28] Herbsleb, J. D.—Grinter, R. E.: Splitting the Organization and Integrating the Code: Conway's Law Revisited. Proceedings of the 21st International Conference on Software Engineering (ICSE '99), ACM, 1999, pp. 85–95.

[29] Hutchinson, J.—Rouncefield, M.—Whittle, J.: Model-Driven Engineering Practices in Industry. 2011 33rd International Conference on Software Engineering (ICSE), IEEE, 2011, pp. 633–642.

[30] Jacobson, I.—Booch, G.—Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, 1999.

[31] Karlesky, M.—Williams, G.: Mocking the Embedded World: Test-Driven Development, Continuous Integration, and Design Patterns. Embedded Systems Conference Silicon Valley, 2007.

[32] Kelly, S.—Tolvanen, J.-P.: Domain-Specific Modeling: Enabling Full Code Generation. John Wiley & Sons, 2008.

[33] Kent, S.: Model Driven Engineering. Proceedings of the Third International Conference on Integrated Formal Methods (IFM '02), Springer, London, UK, 2002, pp. 286–298.

[34] Khuller, S.—Raghavachari, B.: Graph and Network Algorithms. ACM Computing Surveys, Vol. 28, 1996, No. 1, pp. 43–45.

[35] Kolovos, D. S.—Di Ruscio, D.—Pierantonio, A.—Paige, R. F.: Different Models for Model Matching: An Analysis of Approaches to Support Model Differencing. Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM '09), IEEE Computer Society, 2009, pp. 1–6.

[36] Koster, V.: Implementation and Integration of a Domain Specific Language with OAW and Xtext. MT AG, Ratingen, 2007.

[37] Kramer, M. E.—Klein, J.—Steel, J. R.: Building Specifications as a Domain-Specific Aspect Language. Proceedings of the Seventh Workshop on Domain-Specific Aspect Languages, ACM, 2012, pp. 29–32.

[38] Love, C.: ASP.NET 3.5 Website Programming: Problem – Design – Solution. Wrox, 2009.

[39] McConnell, S.: Code Complete. Second Edition. Microsoft Press, Redmond, WA, USA, 2004.

[40] Mcilroy, D.: Mass-Produced Software Components. Proceedings of the 1st International Conference on Software Engineering, Garmisch Partenkirchen, Germany, 1968, pp. 88–98.

[41] Mellor, S. J.—Clark, A. N.—Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software, Vol. 20, 2003, No. 5, pp. 14–18.

[42] Meyers, B.—Vangheluwe, H.: A Framework for Evolution of Modelling Languages. Science of Computer Programming, Vol. 76, 2011, No. 12, pp. 1223–1246.

[43] Miller, J.—Mukerji, J.—Belaunde, M.—Burt, C.—Cummins, F.—Dsouza, D.—Duddy, K.—Kaim, W. E.—Oya, M.—Soley, R.—Watson, A.: MDA Guide, v1.0.1. Technical report, Object Management Group, 2003, http://www.omg.org/docs/omg/03-06-01.pdf.

[44] Murta, L.—Correa, C.—Prudencio, J. G.—Werner, C.: Towards Odyssey-VCS 2: Improvements over a UML-Based Version Control System. Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models (CVSM '08), ACM, 2008, pp. 25–30.

[45] Nathan, A.: Windows Presentation Foundation Unleashed (WPF) (Unleashed). Sams, Indianapolis, IN, USA, 2006.

[46] Néron, B.—Ménager, H.—Maufrais, C.—Joly, N.—Maupetit, J.—Letort, S.—Carrère, S.—Tufféry, P.—Letondal, C.: Mobyle: A New Full Web Bioinformatics Framework. Bioinformatics, Vol. 25, 2009, No. 22, pp. 3005–3011.

[47] Nguer, E. M.—Spyratos, N.: A User-Friendly Interface for Evaluating Preference Queries over Tabular Data. Proceedings of the 26th Annual ACM International Conference on Design of Communication (SIGDOC '08), ACM, 2008, pp. 33–42.

[48] Oliveira, H. L. R.—Murta, L. G. P.—Werner, C.: Odyssey-VCS: A Flexible Version Control System for UML Model Elements. Proceedings of the 12th International Workshop on Software Configuration Management (SCM '05), ACM, 2005, pp. 1–16.

[49] Olsson, K.: Daily Build. The Best of Both Worlds: Rapid Development and Control. Technical report, Swedish Eng. Industries, 1999.

[50] OMG. Meta Object Facility 2.0. Technical report, OMG, 2005. http://www.omg.org/mof/.

[51] PMI. A Guide to the Project Management Body of Knowledge, Third Ed. Project Management Institute, 2005.

[52] Reiter, T.—Altmanninger, K.—Bergmayr, A.—Kotsis, G.: Models in Conflict – Detection of Semantic Conflicts in Model-Based Development. Proceedings of 3rd International Workshop on Model-Driven Enterprise Information Systems (MDEIS-2007), in conjuction with 9th International Conference on Enterprise Information Systems, 2007.

[53] Richardson, J.—Gwaltney, W.: Ship It! A Practical Guide to Successful Software Projects. Pragmatic Bookshelf, 2005.

[54] Ruiz, M.—España, S.—Pastor, Ó.—González, A.: Supporting Organisational Evolution by Means of Model-Driven Reengineering Frameworks. 2013 IEEE Seventh International Conference on Research Challenges in Information Science (RCIS), IEEE, 2013, pp. 1–10.

[55] Rumpe, B.: Towards Model and Language Composition. Proceedings of the First Workshop on the Globalization of Domain Specific Languages, ACM, 2013, pp. 4–7.

[56] Schaefer, I.: Variability Modelling for Model-Driven Development of Software Product Lines. Proceedings of Fourth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), 2010, pp. 85–92.

[57] Schmid, K.—Verlage, M.: The Economic Impact of Product Line Adoption and Evolution. IEEE Software, Vol. 19, 2002, No. 4, pp. 50–57.

[58] Seidewitz, E.: What Models Mean. IEEE Software, Vol. 20, 2003, No. 5, pp. 26–32.

[59] Selic, B.: MDA Manifestations. The European Journal for the Informatics Professional (UPGRADE), Vol. 9, 2008, No. 2, pp. 12–16.

[60] Sharpe, R.: Building a Better Bug-Trap. The Economist. Science Technology Quarterly, 2003.

[61] Shneiderman, B.: Designing the User Interface Strategies for Effective Human-Computer Interaction. SIGBIO News, Vol. 9, 1987, No. 1, 6.

[62] Spinellis, D.: Version Control Systems. IEEE Software, Vol. 22, 2005, No. 5, pp. 108–109.

[63] Spinellis, D. D.: The Information Furnace: User-Friendly Home Control. Proceedings of the 3rd International System Administration and Networking Conference (SANE 2002), 2002, pp. 145–174.

[64] Ståhl, D.—Bosch, J.: Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study. The 12th IASTED International Conference on Software Engineering, 2013.

[65] Szabo, C.—Chen, Y.: A Model-Driven Approach for Ensuring Change Traceability and Multi-Model Consistency. 22nd Australian Software Engineering Conference (ASWEC), IEEE, 2013, pp. 127–136.

[66] Tassey, G.: The Economic Impacts of Inadequate Infrastructure for Software Testing. Technical report, National Institute of Standards and Technology, 2002.

[67] Tolosa, J. B.: A New Approach for Meta-Model Interoperability through Transformation Models. Master's Thesis, University of Oviedo, 2009.

[68] Tolvanen, J.-P.: Domain-Specific Modeling in Practice. Technical report, Metacase, 2008.

[69] Tomassetti, F.—Torchiano, M.—Tiso, A.—Ricca, F.—Reggio, G.: Maturity of Software Modelling and Model Driven Engineering: A Survey in the Italian Industry. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012), IET, 2012, pp. 91–100.

[70] Toulmé, A.: Presentation of EMF Compare Utility. EclipseCon, 2007.

[71] van der Linden, F. J.—Schmid, K.—Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer, July 2007.

[72] Völter, M.: A Catalog of Patterns for Program Generation. Eighth European Conference on Pattern Languages of Programs (EuroPloP 2003), 2003.

[73] Völter, M.—Stahl, T.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons, June 2006.

[74] Wade, J.: Practical Guidelines for a User-Friendly Interface. Proceedings of the International Conference on APL (APL '84), ACM, 1984, pp. 365–371.

[75] Whittle, J.—Hutchinson, J.—Rouncefield, M.—Burden, H.—Heldal, R.: Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? Model-Driven Engineering Languages and Systems. Springer, Lecture Notes in Computer Science, Vol. 8107, 2013, pp. 1–17.

Vicente García-Díaz is Lecturer in the Computer Science Department of the University of Oviedo. He has got his Ph.D. in computer engineering from the University of Oviedo. His research interests include model-driven engineering, domain specific languages, technology for learning and entertainment, project risk management, software development processes and practices. He has graduated in Prevention of Occupational Risks and is a Certified Associate in Project Management through the Project Management Institute.

Jordán Pascual Espada is a research scientist in the Computer Science Department of the University of Oviedo. He has got his Ph.D. in computer engineering from the University of Oviedo. He has got his B.Sc. in computer science engineering and his M.Sc. in web engineering. He has published several articles in international journals and conferences and he has worked in several national research projects. His research interests include the Internet of Things, exploration of new applications and associated human computer interaction issues in ubiquitous computing and emerging technologies, particularly mobile and web applications.

Edward Rolando Núnez-Valdéz has got his Ph.D. in computer engineering from the University of Oviedo. He has got his M.Sc. and a DEA in software engineering from the Pontifical University of Salamanca and his B.Sc. in computer science from the Autonomous University of Santo Domingo. He has participated in several research projects. He has taught mathematics and computer science at various schools and universities and has worked in software development companies and IT Consulting as an IT consultant and application developer. He has published several articles in international journals and conferences. His research interests include object-oriented technology, web engineering, recommendation systems and modeling software.

B. Cristina Pelayo G-Bustelo is Lecturer in the Computer Science Department of the University of Oviedo. She has got her Ph.D. in computer engineering from the University of Oviedo. Her research interests include object-oriented technology, web engineering, eGovernment, modeling software with BPM, DSL and MDA.

Juan Manuel Cueva Lovelle graduated as a mining engineer from Oviedo Mining Engineers Technical School in 1983 (Oviedo University, Spain). He has got his Ph.D. from the Polytechnic University of Madrid (1990). From 1985 he has been Professor in the languages and computers systems area at the University of Oviedo, and he is the ACM and IEEE voting member. His research interests include object-oriented technology, language processors, human-computer interface, web engineering, modeling software with BPM, DSL and MDA.