Continuous integration support in modeling tools

Robbert Jongeling, Jan Carlson, Antonio Cicchetti, Federico Ciccozzi
Mälardalen University, Västerås, Sweden
Email: {robbert.jongeling, jan.carlson, antonio.cicchetti, federico.ciccozzi}@mdh.se

ABSTRACT

Continuous Integration (CI) and Model-Based Development (MBD) have both been hailed as practices that improve the productivity of software development. Their combination has the potential to boost productivity even more. The goal of our research is to identify impediments to realizing this combination in industrial collaborative modeling practices. In this paper, we examine certain specific features of modeling tools that, due to their immaturity, may represent impediments to combining CI and MBD. To this end, we identify features of modeling tools that are relevant to enabling CI practices in MBD processes and we review modeling tools with respect to their level of support for each of these features.

1 INTRODUCTION

In this work we couple two concepts: Continuous Integration (CI) and Model-Based Development (MBD). CI refers to a subset of the Agile development practices as described by Martin Fowler [11]. Several empirical evaluations have shown the positive effects of CI on productivity in industrial software development projects [20, 27]. The MBD paradigm holds the promise of increased productivity of development teams by raising the level of abstraction from code to models [24]. The benefits of MBD on productivity have been empirically assessed in industrial settings too [5, 16, 21].

We hypothesize that CI practices can further improve the productivity of MBD. In our research, we aim at identifying impediments to realizing these practices in industry. Eventually, although not in the scope of this paper, we aim at resolving them, thereby contributing to the maturity of collaborative modeling practices.

In this work, we focus on technical challenges towards introducing CI practices in MBD. We examine modeling tools to identify particular features that are commonly underdeveloped and thereby may represent potential impediments to combining CI and MBD. In particular, we answer the following research questions:

(1) What are relevant features for a modeling tool to be able to support CI practices?
(2) To what extent are these features provided in current modeling tools?

In the remainder of this introduction, the concepts of CI and MBD are described in more detail. The rest of the paper is organized as follows. Section 2 outlines related reviews of modeling tools. The first research question is answered in Section 3. In Section 4, the second research question is answered. The results are discussed in Section 5. Threats to the validity of our work are outlined in Section 6 and the conclusions are provided in Section 7.

1.1 Continuous Integration

CI is one of the twelve Extreme Programming (XP) practices [11]. In turn, XP is one of the elements of the software development concepts published in the Agile manifesto [6]. Since then, various practices regarding the frequency and level (e.g. the entire system, a sub-system or an own branch) of software integrations have been developed [19]. The terms Continuous Integration, Continuous Deployment and Continuous Delivery are sometimes used interchangeably [26]. An unclear definition and interchangeable use of the terms may lead to their devaluation [28]. Therefore, we refer to the definition of Continuous Integration, given by Fowler [11], as follows:

Definition 1. Continuous Integration is a collaborative development practice where software engineers frequently, at least daily, integrate their work into a shared repository. After each integration, an automatic build is performed. On successful build, automated tests are performed.

In MBD, this integrated work consists of models and other modeling artefacts in addition to code. This may pose additional challenges, such as differencing on model level, that are not encountered when applying CI practices in conventional software development projects.

1.2 Model-Based Development

We use the term Model-Based Development (MBD) to denote modeling practices in which models are used to capture functionality and possibly to generate code. Rios et al. distinguish five maturity levels of modeling practices [23]. The levels range from ad-hoc to ultimate and describe immature to complete modeling practices. Since our goal is to introduce CI practices in MBD, it does not make sense to consider the first level, which describes immature modeling practices where models are only used by individuals for e.g. design or documentation. Instead, we are interested the more advanced levels of modeling practices where models are used by multiple team members and the eventual code or application derived from the models must be consistent with these models.

2 RELATED WORK

Since the publication of the agile manifesto in 2001, research on combining Agile and modeling practices [4], has been performed. Evidence-based software engineering studies of the field have shown that Agile modeling is not a mature field yet [1, 15]. In particular, these studies identify the need for more reports on Agile modeling practices in industry.

Although Agile modeling is a topic treated in several research articles and CI is named as a needed practice in modeling [2], we have only found few that go into the details of the Agile practice of CI in combination with MBD. García-Díaz et al. identify four problems in applying CI practices in an MBD project [14]. Among these problems, the two most interesting in relation to CI (as we will see later in Section 3) are version control systems for models and incremental code generation. Additionally, they stress the importance of uniform, user-friendly interfaces and the variability of technologies in different phases of the pipeline. They build and evaluate a prototype solution to resolve these identified problems in an MBD project. This solution focuses on modeling approaches where models are used to generate 100% of the code for an application, whereas we consider also modeling approaches where only parts of the code are generated.

Some work has been done towards a build server for MBD [30]. The authors identify the need to support verification and validation of models. Furthermore, they argue the need for build tooling to support a mix of automatic and manual actions.

Recently, early work towards resolving impediments to combining Continuous Delivery and MBD was presented [12, 13]. The authors identify the main technical impediment to be the model-awareness of the integration server. Furthermore, they remark that for all typical MBD activities, tools that can be included in a pipeline are available. Our approach looks at this from a different perspective; we explore modeling tools and then consider the possibilities to introduce CI practices.

Naturally, there are aspects of MBD in general, not just the combination of CI and MBD, that are relevant to its adoption in industry. The impediments may be technical or be related to organizational factors of the software development process. Tooling is one of the aspects preventing a more wide-spread adoption of MBD in industry; another is the lack of clear processes to support development [33]. There are numerous organizational challenges that, if not properly tackled, hinder agility in MBD [29]. Other impediments to the adoption of MBD that are highlighted in literature are the lack of tool interoperability [18], and the steep learning curve for developers [5, 29]. Furthermore, impediments might lie in development processes of companies and the required time and money investments to change these. Technical challenges include poor scalability of the modeling practice in general in large industrial applications of MBD [5]. Related to these are automated test generation and performance of generated code. Finally, Selic mentions challenges regarding the integration with legacy systems, traceability of generated code and the ability to execute models [25]. Nevertheless, we limit the scope of this paper to those aspects relevant for introducing CI practices in MBD.

In megamodeling, a model is created to describe the relationships between all concepts in a modeling project [10]. Since part of the work to enable CI practices is to chart the artefacts in a project and how they are related, a CI pipeline could be seen as a megamodel. In this paper, we do not further explore this relation and instead focus on existing modeling tools that are used in current practice.

3 IDENTIFYING RELEVANT ASPECTS

In this section, we identify aspects of modeling tools that are relevant for enabling CI practices in MBD. We first identify core aspects of CI based on our definition and existing literature. Then, we list particular aspects that realize these general CI aspects in the MBD domain, based on existing MBD literature. We submitted the identified aspects for review to two industrial practitioners of MBD. The resulting aspects are summarized in Table 1.

3.1 Core CI Aspects

In their literature study, Shahin et al. provide an overview of types of tools used to form a pipeline for Continuous Deployment (CD) [26]. The categories are: *Version Control System*, *Code Management and Analysis*, *Build System*, *CI Server*, *Testing*, *Configuration and Provisioning*, and *CD Server*. They note that an implementation of CD does not necessarily include all categories. Furthermore, the set of CI practices can be considered a part of that CD pipeline, where the latter two tool categories (i.e. *Configuration and Provisioning* and *CD Server*) are excluded. Nevertheless, the CD pipeline is a good starting point to find out relevant features for combining CI and MBD.

We now elaborate on each of the proposed categories and their relevance for CI practices in MBD. A Version Control System (VCS) is used to manage different versions of developed artefacts. The artefacts are typically stored in a shared repository, sometimes allowing developers to copy it and work on their own instance, or branch. Integration of code is done into this shared repository. Code management and analysis techniques such as static code (or model) analysis might be employed to improve the quality of artefacts in a CI process. They are however themselves not closely related to the three main activities in the definition of CI: integrating, building and testing. Therefore, we do not include the Code Management and Analysis category. A build system typically combines artefacts to create executables. In the case of MBD, this could mean generating code from models but also keeping the consistency of several related models. CI servers do not perform builds themselves but rather execute builds and automated tests in other tools, the results are combined in a status overview [11]. These automated tests are important to indicate the quality of integrations. In that way, they contribute to increasing the predictability of the amount of work left, one of the purposes of CI.

3.2 MBD Aspects Related to the Core Continuous Integration CI Aspects

We differentiate between three types of projects to which CI can be applied. The first type is traditional software development, no models are used at all. The second type is very mature MBD, where all code is generated from models and no manual coding is performed. The third type concerns less mature kinds of MBD, where code is partially generated and then manually extended to form a complete application. We consider the latter two types in this paper. The inclusion of modeling artefacts in the CI process requires that some parts of integration, testing and building are handled differently. In this section, we identify the specific aspects of MBD that constitute these differences.

3.2.1 Integration. We consider the integration of changes to individual artefacts into the repository and focus on the aspects of model differencing and model merging to facilitate this integration. Integration is thus considered an activity localized to the directly changed artefacts. If multiple artefacts need to be changed to maintain consistency of the models, this is considered as the responsibility of the developer.

In order to integrate models in a shared repository, there must be a way to control their different versions. Zhang and Patel [34] refer to CI in MBD as “Continuous Modeling.” They identify the need to merge frequently, but also note that merge tooling cannot handle many simultaneous changes. Merging of models is also identified as an important aspect to the adoption of modeling tools in general [7, 29]. Alternatively, pessimistic locking is used to avoid merge conflicts by allowing only one developer at a time to make changes to a model or part of a model [3].

3.2.2 Building. There is no direct MBD equivalent of a build system for conventional programming languages. A build system for models requires more steps than a build system for code, since code needs first to be generated from the models and possibly altered or completed by developers. Given the scope of our research, we adapt the previously identified aspect of a build server to include more model-specific actions. Automated code generation is a central part of continuous integration in a modeling context [34]. Furthermore, it is argued that code generators should work incrementally, i.e., that code should be generated only for parts of the model that have changed [14]. But the key elements of building in MBD are the ability to generate code and the ability to synchronize models and code. Therefore, we add only code generation and model discovery as relevant aspects.

For the different types of CI projects, these aspects can have different meanings. In projects with complete code generation, this generation is a task for the build server and is not performed locally by developers. When code is only partially generated, this can be done both locally and on the build server. In case of complete code generation, model discovery is irrelevant, it will never be done since the code is never manually edited. Conversely, in a partial code generation scenario, model discovery can be needed both locally and remotely. In the simplest form, a developer locally makes changes to a piece of generated code and immediately updates the corresponding model too. In a more complex scenario, a developer could only change the code, integrate it and expect the build server to propagate her changes to the model.

3.2.3 Testing. We distinguish three components of testing in MBD: validating the model with respect to syntax, verifying the model with respect to predefined requirements and verifying models after integration (integration testing). The aforementioned continuous modeling practices specify unit and integration testing as important practices to build confidence in the product [34]. Verification and validation of models are identified as crucial features of a build server for MBD [30]. In both cases, "testing" specifically refers to testing the correctness of the created models, it is assumed that correct models yield correct code and thus correct applications. We therefore add model validation and verification, as well as integration testing, as sub-categories of the testing aspect.

Again, some distinctions can be made in testing between the different types of CI projects. In case of complete code generation, testing the models and the generated code should yield the same results. In partial code generation projects, code is the predominantly tested artefact. In both cases, validation of models with respect to syntax is an implicit part of the code generation process, which will not yield correct output for invalid models. This validation can also be a local action, but this is not required for the CI process. It does not prevent the integration of invalid or incorrect models, analogous to traditional software development in which e.g. non-compiling or incorrect code is integrated. In such cases, the builds or tests are expected to fail.

3.2.4 Model-Awareness. In some work about combining CI and MBD, authors have argued the need for more *model-awareness* in tasks related to CI. In case of the build tooling, it is argued that manual actions that are typically performed in MBD should be taken into account and that testing should focus on validation and verification of models [30]. Others argue that the entire pipeline should be model-aware, such that dependencies between artefacts and between jobs in the pipeline that would be lost in textual representations can be discovered [13]. García-Díaz et al. also note lack of model-awareness, particularly in version control systems and code generation [14]. We do not add an aspect *model-awareness* to our list but when discussing the other aspects in modeling tools, we do take into account to what extent their implementation is model-aware.

The need for model-awareness may also refer to the need to synchronize several models when one of them is changed. In our case, this model synchronization is limited to the consistency of models and code, and between several models in a single project. In other cases, the term co-evolution is used to refer to similar activities that also include the synchronization of models and metamodels, or the synchronization of models and model transformations. Since the most used metamodels in the considered tools are UML and SysML, they and the model-to-text transformations (code generators) are typically not changed during a project. We therefore refer to this MBD aspect as "model synchronization."

Extensive support for activities related to model synchronization, such as automatically handling inconsistencies, is required in modeling tools [32]. Model synchronization is also related to code generation and model discovery, i.e., the automatic creation of a model from code. Since the generated code and models can become inconsistent after changes to either. Modeling tools can support this synchronization, e.g. by providing automated impact analysis for changed artefacts, but the process cannot always be automated. Therefore, manual actions could be required during model synchronization, this step is unsuitable for inclusion in automated builds. Since we envision a CI pipeline for models that is automated similarly to that for traditional software development, we consider model synchronization to be a task performed locally by a developer. Consequently, the CI server or build server is not concerned with tasks such as propagating changes to other artefacts. The identified need for support is still relevant, since the developer should be supported in her local work.

3.2.5 Automation. Interoperability of a modeling tool with other tools is an important aspect to consider too [34]. To assess their suitability to cooperate with CI servers, we look at possibilities to run modeling tools in batch mode or call their functionality from the command line. If this functionality exists, it can be used to create a script that automates part of the CI pipeline, including building and testing. Such automation is of crucial importance to the adoption of CI practices in industry.

3.2.6 Summary. The aspects identified in this section are summarized in Table 1. It contains primary aspects (in bold) and secondary, more specific aspects. In Section 4, we evaluate a set of modeling tools with respect to these primary aspects based on their support of the secondary aspects. Notably, not all CI aspects are directly mapped to a single MBD aspect. Rather, the MBD aspects are specific to their domain and target a more specific functionality than the general CI aspects. In the table, the citations refer to literature sources used to identify the relevance of the related aspects.

Table 1: Identified relevant aspects of CI in MBD.

		CI	MBD
Aspects	**Integration**	[26]	[7, 29, 34]
	Model Differencing		
	Model Merging		
	Building	[26]	
	Code Generation		[14, 34]
	Model Discovery		[32]
	Model Synchronization		[12, 32]
	Testing	[26]	
	Model V&V		[30]
	Integration Testing		[34]
	Automation	[26]	[12]
	Building		
	Testing		

4 SUPPORTED ASPECTS

In this section, we introduce the evaluated modeling tools and discuss for each tool how it implements support for the primary aspects in Table 1. We discuss the tools with respect to their support for the relevant aspects of CI in MBD as discussed in Section 3.

Note that the selection of modeling tool(s) depends on more than just the ability to use it in a CI process applied to an MBD project. Conversely, a CI process in MBD depends on more than just the used modeling tool(s), such as the maturity level of the modeling practices. The goal of our work is not to identify the *best* modeling tool for CI, but rather to investigate what impediments in applying CI to MBD projects exist.

4.1 Modeling Tools

The selection of modeling tools was based on their use in industry and on inputs from our industrial partners. We included the four most-used[1] tools in industry as reported by practitioners [18]: Matlab Simulink, Sparx Systems Enterprise Architect, IBM Rational Rhapsody[2], and National Instruments LabVIEW. After discussions with our industrial partners, this list was supplemented with four additional tools that are most relevant to their daily work: NoMagic Magic Draw, PTC Integrity Modeler, OneFact BridgePoint, and Eclipse Papyrus. Most of these tools support UML and SysML, among the most used modeling languages in industry [18]. Exceptions to this are BridgePoint, which supports xtUML (an executable dialect of UML), LabVIEW, which supports their "G" graphical modeling language, and Simulink, which supports modeling in the Simulink language. Most of the tools thus support general purpose modeling languages. The most advanced modeling practice includes the creation of custom Domain-Specific Languages (DSLs). Tools used for that purpose are further away from the state of practice at our industrial partners and therefore not included in this evaluation. An overview of the considered tools is shown in Table 2.

[1]Excluding Eclipse-based tools and in-house tools, since they cannot be specified to a particular tool. They are reported as second and fourth most used respectively [18].
[2]In [18] the reported tool is Rational Modeler, but we include Rational Rhapsody, which can be seen as its successor.

Table 2: Evaluated tools in this review.

Tool	Vendor	Supported Modeling Languages
BridgePoint	OneFact	xtUML
Enterprise Architect	Sparx Systems	UML + SysML
Integrity Modeler	PTC	UML + SysML
LabVIEW	National Instruments	G
Magic Draw	No Magic	UML + SysML
Papyrus	Eclipse	UML + SysML
Rhapsody	IBM	UML + SysML
Simulink	MathWorks	Simulink

4.2 Other Tools

In addition to the modeling tools, a CI pipeline typically also involves Version Control Systems (VCSs) and CI servers. There exist numerous open-source and commercial CI servers, such as Jenkins, Travis, and TeamCity. Some of these allow for a completely custom defined pipeline whereas other tools provide users with a choice between predefined pipelines for some programming languages. Since we aim at using these tools in MBD processes, we are particularly interested in those CI servers that allow the definition of a custom pipeline. Therefore, we will mainly refer to Jenkins in the remainder of this section.

4.3 Tool Evaluations

We evaluated how the selected tools support the primary aspects depicted in Table 1. The evaluations are based on publicly available documentation and research papers about the tools. The results of the evaluation are summarized at the end of this section in Table 3.

4.3.1 Integration. In BridgePoint, version control is difficult to achieve [31]. Automated merging is not supported but the tool does show a visual difference between two versions of a model. This is not necessarily an impediment to introducing CI practices, but in practice it may discourage developers to integrate frequently if every integration potentially requires a large manual effort.

Enterprise Architect (EA) has no integrated support for model versioning but relies on pessimistic locking of packages (parts of models). This system grants user exclusive editing rights on a package, thus preventing conflicts due to simultaneous changes. The tool does support the integration of several third-party version control systems, which can be used to store and manage the history of EA models. Additionally, LemonTree is a third-party project that supports optimistic locking, three-way merging and branching for EA models.

Integrity Modeler contains a built-in service for configuration management. It includes a weak optimistic locking mechanism allowing multiple developers to collaborate on the same artefacts simultaneously. When multiple users are editing the same artefact, the changes of one of them are visible to each of the others in real-time. Alternatively, there is an optimistic locking mechanism available, where these changes are not visible. Then, merges can be performed automatically and their results manually edited to resolve merge conflicts.

LabVIEW includes a tool showing graphical differences between model versions. VCSs, such as Git and SVN, can be used to keep track of different model versions. The differencing functionality of those tools can then be redirected to use the graphical difference available in LabVIEW. Merges can be performed automatically and merge conflicts can be resolved manually.

Magic Draw contains a built-in server for version control, it provides a model repository and supports collaboration through branching and merging. Branching allows multiple developers to work in parallel on the same project. A plug-in is available to support merging at model level. In case branching is not used, concurrent changes directly on the mainline are prevented by means of pessimistic locking. The locks can be acquired at sub-model level, i.e., parts of a model can be locked for editing. The locks can then be released or maintained on commit.

Papyrus can be extended using plug-ins that are part of the Eclipse Modeling Framework (EMF). The Collaborative Modeling initiative provides such plug-ins, in terms of collaboration support for modeling in Eclipse, by using EMFCompare for the detection and merging of changes, EGit for distributed version control and Gerrit for reviews of models. This allows developers to create a branch for a project, make changes and merge them into the mainline while staying on the model level. The included version control system EGit is an implementation of Git, incorporated in Eclipse. EMFCompare shows differences of changes between model versions from several views (graphical, textual, tabular). It can automatically merge changes, or in case of conflicts in three-way merges, allows the developer to choose the version to be integrated.

Rhapsody includes the tool DiffMerge, which can show graphical differences and automatically merge models or projects containing models. In case of any merge conflicts, the developer is shown a graphical difference between the versions and can resolve the conflict by choosing one of the versions. The tool also supports integration of version control systems ClearCase and SVN.

Simulink provides version control support through an integrated SVN instance but can also be used together with Git. This allows a project to be branched and thus models to be edited in parallel. Simulink contains an integrated tool for three-way model merging. The tool automatically merges models and on conflict offers a choice between the remote, base and local change.

Summary. There are three main approaches for model versioning in the evaluated tools. The first, locking, does not scale to large collaborative projects. The second, leaving versioning completely to a VCS, is not feasible because the VCS is typically not model-aware, which is required for merging at model level. Furthermore, line based differencing of the XML representations is not appropriate for models [3]. The third and most feasible versioning approach when introducing CI practices is to enable the integration of a version control tool in the modeling tool, but circumscribing model differencing and merging to the modeling tool itself. Indeed, this level of support is provided by multiple tools: Simulink, Rhapsody, Magic Draw, and Papyrus.

4.3.2 *Building.*

BridgePoint provides integrated support for code generation. This functionality forms the “Translatable” part in “eXecutable Translatable UML” (xtUML). Models in xtUML can be transformed to C, SystemC or C++ using included model compilers. These compilers are open source and can be customized. It is also possible to create new compilers to translate models into different programming languages. Changes to generated code are not propagated back to models. So, there is only support for one-way development and not for the round-trip from models to code and back. When the generated code is a complete application rather than a skeleton or a detached, individual subsystem, this is not necessarily an impediment to introducing CI practices.

In Enterprise Architect, skeleton code can be generated from both Class diagrams and Interface models. More detailed code can be generated from sequence-, activity-, and state machine diagrams. Several languages are supported, including C, C++, C#, and Java. Reverse engineering is also (partially) supported since some UML diagrams can be generated from code. Enterprise Architect includes a development environment where generated code can be edited. This environment also supports typical functionalities of a code editor such as debugging and profiling. Code generation and reverse engineering can be combined and an option exists to keep models and code synchronized. When generated code is updated due to a change in the model, the body of methods is untouched, only their headers are changed such that previous work is not undone. Although Enterprise Architect provides traceability matrices from requirements to models for requirements engineering, impact analysis for changes in models is not supported. Some work has been done on creating impact analysis techniques for any Enterprise Architect model, showing the potential of third party solutions to solve this problem [17].

Integrity Modeler supports code generation from class and state-machine diagrams in several programming languages, including Ada, C++, and Java. The generated code might be complete but usually manual editing is required [22]. The tool includes functionality to keep models and code synchronized in real-time when manually altering the code. Furthermore, it supports impact analysis by letting the user define relationships between different modeling artefacts. These relationships can then be visualized to identify potential model elements that need to be synchronized.

Using additional code generators, C and Ada code can be generated from LabVIEW models. Templates on which the generations are based can be customized. Alternatively, the generated code can be customized after generation. Code generation is a one-way process, where the generated code is expected to be complete with no manual editing required. The generators are designed to produce code that can be integrated in a larger project.

Magic Draw can generate code in several languages (Java, C++, C#). In most cases, code generated from models will be skeletons and thus will be edited by developers to implement complete functionalities. There is also support for reverse engineering; models can be derived from code. Forward and reverse code engineering is managed using *Code engineering sets*. These sets contain model elements for which code is generated and conversely files from which code is reversed to models. In addition, relationships between model components can be defined. These can be visualized in different ways to show the impact of changes on the remaining model artefacts in the project.

Papyrus supports code generation from UML models through plug-ins. There are plug-ins available for the generation of C++ and Java code from UML models, but it is also possible to create custom code generators for other languages. Reverse engineering is supported as part of the Papyrus Software Designer tooling, using it, class diagrams can be generated from Java classes and packages. Using the Papyrus Software Designer plug-in, models and generated code can be synchronized. Changes to the code are then propagated back to the model and changes in the model are incrementally applied to the code.

Rhapsody can generate code in C, C++, Java and, using a specific Rhapsody Developer version, also for Ada. This is done incrementally, i.e., only new code is generated for modified model elements. The generated code can be modified and changes are propagated back to the model. It is also possible to specify code that should not be included in this round-trip, which could be useful for implementation-specific code that is not to be reused in other versions of a product. To see the impact of a change on the other artefacts in the models, Rhapsody supports automated impact analysis. The user configures the analysis by defining, among other things, the types of links to follow and their depth. Given the result of an impact analysis, it is up to the developer to manually co-evolve the impacted artefacts.

The generation of C and C++ code from Simulink models is supported by additional tools that can be integrated in Simulink, such as Embedded Coder and Simulink Coder. Generated code is a complete program, not just skeleton code. Modifying the generated code can be done at the level of the code generators, which can be configured to replace code by custom snippets. This also means that the process of code generation is one-way; there is no support for propagating manual changes in the generated code back to the model. Simulink also contains a facility for automated impact analysis that can predict impacted elements in anticipation of a particular change.

Summary. The basic functionality of generating skeleton code from models is present in each of the considered tools. The detail of the created models dictates whether the entire application or only skeleton code can be generated. This distinction usually influences the functionality regarding synchronization of models and code too. This aspect is usually better supported in tools that just produce skeleton code than in tools that produce complete code and where thus the generated code does not require manual editing. We have seen that some tools contain functionality to assess the impact of changes at model level, but that the implementations still rely mostly on manual actions, which is not ideal in an automated build scenario. We note that this type of synchronization functionality focuses on models and code created in a single modeling tool. In projects where multiple modeling tools are used, more challenges related to the synchronization of the different models can be expected. This is mainly due to the limitation of impact analysis to assess only the impact of changes in models to models created in the same tool.

4.3.3 Testing.

To test models, BridgePoint provides a *verifier*. This functionality forms the "eXecutable" part of xtUML. The verifier can be used to test models without the need to regenerate source code. It executes the model itself and supports placement of breakpoints and inspection of variable values during the simulations. This can be useful for manual testing, but less so in CI settings, where automated testing is preferred.

In Enterprise Architect, models can be simulated and test scripts can be defined to automatically test model elements. In these scripts, unit tests for Java (JUnit) or .NET (NUnit) can be called. A skeleton for these unit tests can automatically be generated from class diagrams. By default, the tool supports the validation of models with respect to the UML syntax, but custom rules can be added. Furthermore, validation and test scripts can be used to automate testing of models, whereas the simulation functionality is mostly meant for debugging.

LabVIEW includes a framework for unit testing. Test cases can be defined in the tool itself by defining input values and expected output values for a specific unit under test. The tests can be executed in isolation or in a test suite. The tool includes a functionality to track tests and the code it covers, automatically providing the developer with code coverage information. In addition to this, models can be validated using static code analysis rules, which can be customized for particular purposes.

Integrity Modeler contains a framework for automated testing. In it, test cases can be defined, triggered and their results viewed. The test cases can also be grouped in sessions, allowing their execution to be automated.

In Magic Draw, models can be validated with respect to predefined constraints or custom created constraints expressed in the Object Constraint Language (OCL). If the validation logic cannot be expressed in OCL, boolean constraints can be defined in Java. Additionally, unit tests can be defined to verify models or integrations. JUnit is used to express test cases that can be executed using the build-in test framework. The framework also provides functionality for checking the created program for memory leaks.

Papyrus models can be validated with respect to predefined soundness constraints. Custom constraints can be defined in OCL. Validations can be performed on an entire model as well as on parts of a model. Warnings or errors are shown in the models themselves after a validation. This validation is a static check, but UML models can also be executed, using the execution engine of the Moka module. This can be used to manually test models, but there is also support for automatic testing. The Papyrus Testing Framework supports the automatic generation of unit tests from UML diagrams. The unit tests can be automatically tested using the JUnit framework.

Before code is generated in Rhapsody, a model-checker can be run to validate the model with respect to predefined and custom defined rules. Such custom rules have to be written in Java and can be used to check both the structural and the behavioral aspects of the model. Included in the tool is also a functionality to simulate models (or animate as is the used terminology for this tool). In addition, the tool can be integrated with other IBM Rational tools for testing (Test RealTime) and quality assessment (Quality Manager). Furthermore, the tool includes a framework for the automatic generation of test cases.

Similar to code generation, there exist additional tools for the validation and testing of Simulink models. Simulink Test is a tool that supports creation and execution of test cases for models. Test cases can be defined to verify the models with respect to functional constraints. It also provides an overview of failed and succeeded test cases, similar to the dashboard of CI tools.

Summary. Most tools contain a unit testing mechanism. Some implement their own and some use existing frameworks such as JUnit. Most tools also include model validation functionality, a check of the well-formedness of the models with respect to the metamodel. Additionally, some tools are capable of simulating models, which is primarily useful for manual debugging. Next, we look at ways to automate builds and tests in the considered tools.

4.3.4 Automation. The BridgePoint editor is based on Eclipse and its model compilers are implemented as Eclipse plug-ins. It is possible to run these from the command line and thus incorporate them as a build step in a CI pipeline. Similarly, the testing functionality incorporated in the *verifier* can be included in an automated process.

Enterprise Architect offers the possibility to create analyzer scripts, these can be used to automate builds, tests, and other functionalities. The scripts can be created in the tool itself and allow for execution of the builds and tests from the command line. The scripts can also be used to specify an output file to contain a generated report on the test results. Furthermore, they can be used to execute the models and deploy the project, but that is out of the scope of our definition of CI.

Automation for CI can easily be achieved in Integrity Modeler using a Jenkins plug-in.[3] The plug-in can detect changes in the built-in repository and can be configured to execute builds after such a detection. Furthermore, it can retrieve the results of automated tests, executed after the build. The availability of the Jenkins plug-in signals a higher level of maturity with respect to CI processes than seen in other tools.

For LabVIEW, command line interfaces are available as open-source.[4] These allow the builds and tests to be executed by a CI server, e.g. Jenkins. The test reports created by the tool can be stored as HTML files and as such be shown in Jenkins [9].

Magic Draw supports extensibility by custom add-ins through its *Open API*. This API also allows the tool to be run in batch mode. This allows command line access to code generation and unit test execution, which makes it suitable to be used in a CI pipeline. Alternatively, Magic Draw can also be integrated in other applications using its OSGi interfaces. Since this construct is Java-based, it is applicable in fewer cases than the generally applicable batch mode construct.

For Papyrus, code generation and model testing functionalities are packaged in Eclipse plug-ins. These are executable from the command line, which can be leveraged to include Papyrus in a CI pipeline, for example by calling these plug-ins from scripts managed by a CI server such as Jenkins. Creating a CI pipeline for conventional Eclipse projects is a common practice, so it is not expected that these particular tools would yield new problems.

Rhapsody offers command line interfaces for code generation and the DiffMerge tool. Using these commands, code can be generated for specific components or for a project containing a number of components. This allows for these tasks to be integrated in a CI pipeline where only models are checked in to the version control system and the application is generated.

It is possible to create a CI pipeline using Jenkins to automatically execute builds and tests in Simulink. Furthermore, the CI tool can be configured to report on the success or failure of the automated tests. Such a process can be created using MATLAB, Git and Jenkins [8].

Summary. With some effort, each modeling tool can be included in a CI pipeline. We have seen some examples of ready-made pipelines including some of the discussed tools. As long as the different approaches to automation can still be executed from a pipeline, there should be no impediments regarding combining automation for multiple modeling tools in one pipeline.

4.3.5 Evaluation Summary. Table 3 summarizes the evaluations by scoring the different aspects in each tool as mature, standard, or immature. We define the thresholds for these scores, based on the identified aspects in Section 3, as follows. For integration, an immature level of support is considered an approach that does not allow versioning of models, but e.g. utilizes pessimistic locking. A standard support would be one where models can be merged. A mature level of support is considered when the tool also supports the visualization and resolution of conflicts. For building, an immature level of support would be one where no code generation is possible. Code generation is considered standard support, while mature support includes back-propagation of code changes to models. For testing, immature level of support only includes syntactical validation of models. Standard support involves also verification of models using model testing. Support for testing is considered mature when it includes unit tests for code and models. For automation, tools are considered to provide immature support if they do not feature explicit hooks to incorporate them in a CI pipeline. A standard level of support includes the possibility to run the tool in batch mode and perform some actions. We require the presence of a command-line API to grant mature level of support for automation.

5 DISCUSSION

We have summarized the answers to our research questions by listing relevant features for a modeling tool to be able to support CI practices in Table 1 and by summarizing the extent to which these features are present in current modeling tools in Table 3.

[3]https://wiki.jenkins.io/display/JENKINS/PTC+Integrity+Plugin Last access: June 4, 2018
[4]https://github.com/JamesMc86/LabVIEW-CLI, retrieved: June 1, 2018

Table 3: Aspects as supported by tools, scored by -, ○, or +, depicting immature, standard, or mature support respectively.

	Tools							
Aspects	BridgePoint	Enterprise Architect	Integrity Modeler	LabView	Magic Draw	Papyrus	Rhapsody	Simulink
Integration	-	-	○	+	+	+	+	+
Building	○	+	+	○	+	+	+	○
Testing	○	+	○	+	○	+	+	○
Automation	-	○	+	+	○	○	+	+

Regarding integration, most of the considered tools provide support for differencing and merging at model level. They provide representations in various formats of model differences and allow developers to resolve merge conflicts at model level. The storage of models and change history is usually managed by a VCS such as SVN or Git. For building, all tools provide some form of code generation. Nonetheless, there is a great variability in the maturity of what the environments can do after the code is generated. Some tools automatically keep code and models synchronized whereas in others code generation is a one-way operation. The most challenging aspect seems to be the synchronization of different models, for which most of the tools include some level of change impact analysis support. Another challenge is the synchronization of models when multiple modeling tools are used in the same software project and combined in the same pipeline. Testing is supported by each of the tools, but with different maturity levels. Finally, it is possible to automate at least parts of the build and testing processes for each of the tools. For some of the modeling tools such automated approaches are available, whereas for others it would require custom configuration.

From the evaluations of the modeling tools, we did not find any theoretical obstacle in introducing CI practices in MBD. Some tools already have fairly good support for CI, and by cherry picking features from other tools, they could be even more suitable for CI practices. In fact, in Section 4.3.4 we have mentioned some applications of CI in each of the tools Integrity Modeler, LabVIEW, and Simulink. On the other hand, we foresee challenges in model synchronization and automation in projects involving multiple modeling tools.

6 THREATS TO VALIDITY

In drafting both the list of relevant aspects and the list of considered tools there is an amount of subjectivity and possible bias. Tools and aspects may be omitted or be listed but less relevant. Incompleteness of the list of aspects was a threat to the validity of this work, since we aim to find impeding aspects. If crucial aspects were not included, we could not find the corresponding problems in the evaluated tools. To limit this risk, we gathered core relevant aspects by investigating existing literature on the topics of CI and MBD; furthermore, the lists were informally validated by two practitioners from industry with experience in MBD and CI. Similarly, the selection of the tools contains a bias towards UML and SysML. Other languages or modeling paradigms may yield different impediments.

Other threats are related to our chosen research methodology. The evaluations are based on publicly available documentation and research papers. This means that we did not experiment with the tools themselves to assess the aspects. The advantage of this approach is that we avoid defining a scenario to evaluate the tools, which may not fit all tools or may accidentally favor some of them. On the other hand, the inherent threat of this approach is that it does not highlight possible issues only visible when using the evaluated tools in practice.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we identified relevant aspects of modeling tools to support CI practices. We then evaluated eight modeling tools and assessed their levels of support for each of the aspects. In the evaluated tools, we have seen different maturity levels of support for the considered aspects. Overall, we found some challenges, but no insurmountable impediments to introducing CI practices in MBD.

The next step of our research consists in a set of interviews with MBD practitioners working at different MBD maturity levels and in different companies. Practitioners will be asked to share their views on introducing CI practices in MBD and their views on the impediments in their context. These interviews may uncover hidden technical aspects that did not arise in this work, or bring up other, non-technical impediments, such as those briefly mentioned in Section 5.

8 ACKNOWLEDGEMENTS

The authors would like to thank the industrial partners for their input in discussions about this work. This work is part of a project supported by Software Center.[5]

REFERENCES

[1] Hessa Alfraihi and Kevin Lano. 2017. The Integration of Agile Development and Model Driven Development - A Systematic Literature Review. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*. SCITEPRESS, 451–458.

[2] Hessa Alfraihi and Kevin Lano. 2017. A Process for Integrating Agile Software Development and Model-Driven Development. In *Proceedings of MODELS 2017 Satellite Event: FlexMDE*. 412–417.

[3] Kerstin Altmanninger, Martina Seidl, and Manuel Wimmer. 2009. A Survey on Model Versioning Approaches. *International Journal of Web Information Systems (IJWIS)* 5, 3 (2009), 271–304.

[4] Scott W Ambler. 2003. Agile Model Driven Development is Good Enough. *IEEE Software* 20, 5 (2003), 71–73.

[5] Paul Baker, Shiou Loh, and Frank Weil. 2005. Model-Driven Engineering in a Large Industrial Context—Motorola Case Study. In *LNCS 3713*. Springer, 476–491.

[6] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, and others. 2001. Manifesto for Agile Software Development. (2001). http://agilemanifesto.org

[7] Francis Bordeleau, Grischa Liebel, Alexander Raschke, Gerald Stieglbauer, and Matthias Tichy. 2017. Challenges and Research Directions for Successfully Applying MBE Tools in Practice. In *Proceedings of the 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 338–343.

[8] Andy Campbell. 2015. The Other Kind of Continuous Integration. (2015). https://blogs.mathworks.com/developer/2015/01/20/the-other-kind-of-continuous-integration Retrieved: 2018-05-14.

[5]www.software-center.se

[9] Fredrik Edling. 2013. Using LabVIEW in a Continuous Integration Environment. (2013). ftp://ftp.ni.com/pub/branches/northern_region/nidays2013/presentations/sw_cag_using_lv_in_cont_integr_environment.pdf Retrieved: 2018-06-01.

[10] Jean-Marie Favre. 2004. Towards a Basic Theory to Model Model Driven Engineering. In *3rd Workshop in Software Model Engineering, WiSME*. Citeseer, 262–271.

[11] Martin Fowler. 2006. Continuous Integration. (2006). https://martinfowler.com/articles/continuousIntegration.html

[12] Jokin Garcia. 2018. Continuous Model-Driven Engineering. (2018). https://modeling-languages.com/continuous-model-driven-engineering/ Retrieved: 2018-05-14.

[13] Jokin Garcia and Jordi Cabot. 2018. Stepwise Adoption of Continuous Delivery in Model-Driven Engineering – Extended Abstract. *DEVOPS* (2018).

[14] Vicente García-Díaz, Jordán Pascual Espada, Edward Rolando Núnez-Valdéz, G Pelayo, B Cristina Bustelo, and Juan Manuel Cueva Lovelle. 2016. Combining the Continuous Integration Practice and the Model-Driven Engineering Approach. *Computing and Informatics* 35, 2 (2016), 299–337.

[15] Sebastian Hansson, Yu Zhao, and Håkan Burden. How MAD are we? Empirical Evidence for Model-driven Agile Development. In *Proceedings of XM 2014, 3rd Extreme Modeling Workshop*, Vol. 1239. 2–11.

[16] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical Assessment of MDE in Industry. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*. IEEE, 471–480.

[17] Melanie Langermeier, Christian Saad, and Bernhard Bauer. 2014. Adaptive Approach for Impact Analysis in Enterprise Architectures. In *International Symposium on Business Modeling and Software Design*. Springer, 22–42.

[18] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2016. Model-Based Engineering in the Embedded Systems Domain: an Industrial Survey on the State-of-Practice. *Software & Systems Modeling* 17, 1 (2016), 91–113.

[19] Torvald Mårtensson, Daniel Ståhl, and Jan Bosch. 2017. Continuous Integration Impediments in Large-Scale Industry Projects. In *Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 169–178.

[20] Ade Miller. 2008. A Hundred Days of Continuous Integration. In *Agile*. IEEE, 289–293.

[21] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A. Fernandez. 2013. An Empirical Study of the State of the Practice and Acceptance of Model-Driven Engineering in Four Industrial Cases. *Empirical Software Engineering* 18, 1 (01 Feb 2013), 89–116.

[22] David Norfolk. 2015. *PTC Integrity Modeler — a Standards-Based Tool for Systems and Software Engineering*. Technical Report.

[23] Erkuden Rios, Teodora Bozheva, Aitor Bediaga, and Nathalie Guilloreau. 2006. MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. In *Proceedings of the European Conference on Model Driven Architecture-Foundations and Applications*. Springer, 78–89.

[24] Douglas C Schmidt. 2006. Model-Driven Engineering. *IEEE Computer* 39, 2 (2006), 25.

[25] Bran Selic. 2003. The Pragmatics of Model-Driven Development. *IEEE software* 20, 5 (2003), 19–25.

[26] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous Integration, Delivery and Deployment: a Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access* 5 (2017), 3909–3943.

[27] Daniel Ståhl and Jan Bosch. 2013. Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study. In *The 12th IASTED International Conference on Software Engineering (Innsbruck, Austria, 2013)*. 736–743.

[28] Daniel Ståhl, Torvald Mårtensson, and Jan Bosch. 2017. Continuous Practices and DevOps: Beyond the Buzz, What Does It All Mean?. In *Software Engineering and Advanced Applications (SEAA), 2017 43rd Euromicro Conference on*. IEEE, 440–448.

[29] Stavros Stavru, Iva Krasteva, and Sylvia Ilieva. 2013. Challenges of Model-driven Modernization-An Agile Perspective.. In *MODELSWARD*. 219–230.

[30] Henrik Steudel, Regina Hebig, and Holger Giese. 2012. A Build Server for Model-Driven Engineering. In *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*. ACM, 67–72.

[31] Nenad Ukić, Pál L Pályi, Marijan Zemljić, Domonkos Asztalos, and Ivan Markota. 2011. Evaluation of Bridgepoint Model-Driven Development Tool in Distributed Environment. In *Workshop on Information and Communication Technologies conjoint with 19th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2011*.

[32] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. 2008. Challenges in Model-Driven Software Engineering. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 35–47.

[33] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2013. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 1–17.

[34] Yuefeng Zhang and Shailesh Patel. 2011. Agile Model-Driven Development in Practice. *IEEE software* 28, 2 (2011), 84–91.