



BRCode: An interpretive model-driven engineering approach for enterprise applications

Anderson Oliveira, Vinicius Bischoff, Lucian José Gonçalves*, Kleinner Farias, Matheus Segalotto

University of Vale do Rio dos Sinos (UNISINOS), 950, Unisinos Av. – Postal Address: 93.022-000, São Leopoldo, Rio Grande do Sul, Brazil



ARTICLE INFO

Article history:

Received 2 June 2017

Received in revised form 22 November 2017

Accepted 9 January 2018

Available online 5 February 2018

Keywords:

Interpretive MDE

Industry

Case study

Productivity

Profitability

ABSTRACT

Many model-driven engineering (MDE) approaches have been proposed in recent studies. They claim to improve software quality and productivity by raising the abstraction level at which developers work. However, they often fall short of what was expected in terms of productivity, profitability, and Return on Investment in real-world scenarios. This article proposes BRCode, which is an interpretive MDE approach for fast-changing enterprise applications. A case study that involves the development of an Enterprise Resource Planning (ERP) system enabled data collection based on 34 realistic scenarios in a Brazilian company. This evaluation compared BRCode with a generative MDE (genMDE) approach. Our results show that (1) genMDE required 93.75% more effort; and (2) genMDE and BRCode led to financial gains in 48% and 70% of the cases, respectively. On average, genMDE led to financial losses in most cases, while BRCode roughly tripled financial gains; (3) BRCode had an ROI of 1.54, compared to 0.07 for genMDE, which represents a difference of 93.37%. The results were encouraging and show the potential for using BRCode to support software production companies in the turbulent business environment.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

The development of enterprise applications has occurred in increasingly unstable business environments in industry. Usually, complex and fast-changing customer requirements, pressures of shorter development cycles, and rapidly advancing information technologies are ever-present characteristics of most mainstream projects [1]. In this turbulent context, software production companies seek to use generative or interpretive model-driven engineering (MDE) approaches for improving software quality and developer productivity [2,3]. In [4], Hailpern and Tarr highlight that these benefits might be achieved by raising the abstraction level at which developers work.

Generative and interpretive MDE approaches are functionally equivalent. Generative MDE approaches (genMDE) aim at transforming model-to-model or model-to-code, while interpretive MDE approaches focus on interpreting models or meta-data to produce run-time applications [2]. Both approaches claim to improve software quality and productivity by raising the abstraction level at which developers work. However, they often

fall short of what would be expected in terms of productivity, profitability, and Return on Investment in real-world scenarios [5]. Still, little has been reported about their effectiveness in mainstream projects in industry. Even worse, current MDE approaches have not provided stable architectures for supporting software products in increasingly turbulent business environments. This instability goes beyond the limits of inconvenience to developers and customers. The rework that is caused by constant changes leads to a cost increase, which is often not cheap.

Today, countless works report that the greater the number of changes, the greater the likelihood that flaws and defects may arise in software products [6]. Usually, the entire software architecture must be changed to accommodate changes that would not affect core requirements of an enterprise application. Despite this, MDE approaches have been adopted, without empirical evidence about their benefits or side-effects. For example, these approaches may fail to increase the capability of the development team to develop, ensure quality, and release enterprise systems more quickly, while ensuring low development cost.

* Corresponding author.

E-mail addresses: andersonmo@edu.unisinos.br (A. Oliveira), viniciusbischoff@edu.unisinos.br (V. Bischoff), lucianj@edu.unisinos.br (L.J. Gonçalves), kleinnerfarias@unisinos.br (K. Farias), msegallotto@edu.unisinos.br (M. Segalotto).

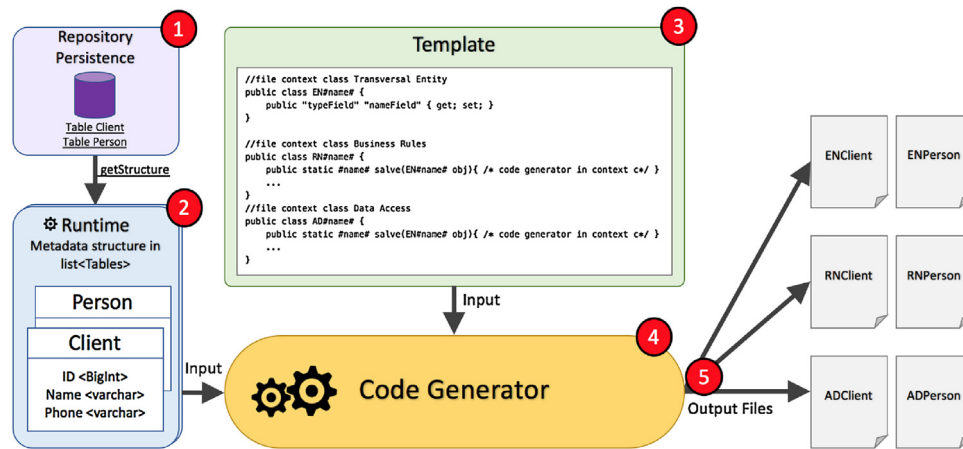


Fig. 1. The generative MDE approach that was adopted by the company.

This article, therefore, proposes BRCODE, which is an interpretive MDE approach for fast-changing enterprise applications. Software developers benefit from using the BRCODE approach typically when performing development and maintenance tasks, such as rendering data-driven user interfaces, elaborating architectural design, and delivering fast, low-cost enterprise applications. For this, BRCODE provides a set of key features that leverage the productivity of software developers by reusing ever-present features in enterprise applications. For example, by automatically rendering user interfaces from meta-data, developers might invest more time designing databases and studying the business rules more carefully.

A case study that involves development of an Enterprise Resource Planning (ERP) system enabled data collection based on 34 realistic scenarios in a Brazilian company. Our results show that (1) genMDE required 93.75% more effort; (2) genMDE and BRCODE led to financial gains in 48% and 70% of the cases, respectively. On average, genMDE led to financial losses in most cases, while BRCODE roughly tripled financial gains; (3) BRCODE had an ROI of 1.54, compared to 0.07 for genMDE, which represents a difference of 93.37%. The obtained results are encouraging and show the potential for using BRCODE to support software production companies in volatile business environments. Moreover, this study is the first to perform an empirical study that compares generative versus interpretive MDE approaches in real-world settings. The empirical knowledge and insights that are generated may serve as a basis for improving the current MDE approaches.

The remainder of this article is organized as follows. Section 2 describes the generative MDE approach that is used in our case study. Section 3 introduces the BRCODE and describes its architecture and main features. Section 4 describes how the proposed approach was evaluated through an empirical study in industry. Section 5 presents the collected results. Section 6 contrasts this study with the current literature. Section 7 presents some conclusions and discusses future studies.

2. Generative MDE approach

MDE approaches [4] aim at shifting the development focus from code to models. Existing MDE approaches are based on *code generation* or *model interpretation* [2]. Approaches of the first type use models and transformers to represent high-level concepts of abstraction and carry out sets of transformations, respectively. The transformers can convert abstract models into less abstract models (i.e., model-to-model transformations) or directly into source code that might be compiled (i.e., model-to-text

transformations) [7]. The source code that is generated from models might cover most of the platform's complexity and configuration details. Consequently, software developers end up concentrating their time on problems that are related to business rules, for example. MDE approaches of the second type use runtime interpretation to produce software that conforms to the model. Typically, these MDE approaches produce an interpreter, which is embedded in the application.

We will mainly focus our attention on the description of the generative MDE approach that was adopted by the Brazilian company in which BRCODE was evaluated (Section 4). To facilitate understanding of how this technique was used, Fig. 1 presents an overall scheme that shows its main elements, which are described as follows:

- Repository:** Software developers produce a data repository based on the customer's requirements and create domain models from these requirements. These models are pivotal to creating the database of the application that is under development. The *Repository* can be seen Fig. 1(1).
- Runtime:** The application is created based on meta-data that are present in the database. Operations, such as CRUD¹ (Create, Read, Update, and Delete), and business rules are defined. Transformers generate tables at the database and data-access objects to manipulate these tables. The *Runtime* component is shown in Fig. 1(2).
- Template:** Software developers can personalize user interface components, such as data masks, combo boxes, styles, object positions, effects, data fields, and database calls. They can also configure how the application manages the information inputs and transforms the retrieved data into a readable format for the clients of the application. The *Template* component is exhibited in Fig. 1(3).
- Code Generator:** The application structure is generated based on the template and run-time inputs. For this, the code generator converts these inputs into source code. This component, which is shown in Fig. 1(4), plays a central role in supporting the generative MDE approach that is used in our case study (discussed in Section 4).
- Output Files:** These files are responsible by representing the generated files throughout the generative process. The application structure is generated, including the entity, business rules,

¹ The four basic functions of persistent storage: create, read, update, and delete (as an acronym CRUD).

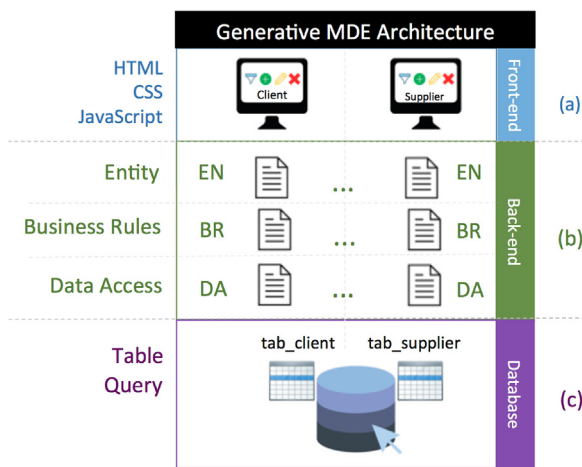


Fig. 2. Application architecture that was produced with the generative MDE approach.

and presentation layer. The software developers can configure how additional functions will be generated. This component is shown in Fig. 1(5).

The generated source code produces enterprise applications based on the architecture exhibits in Fig. 2. The *front-end layer* (Fig. 2(a)) represents the client side, where the client interacts with the produced application. The technologies that are used in this layer include HTML, CSS and JavaScript. The *back-end layer* (Fig. 2(b)) consists of the server side, where the software components that are responsible for implementing the business rules, entity and data access are found. The *database layer* (Fig. 2(c)) contains the tables and queries that are used.

As the source code of the entire enterprise application is not fully generated in generative MDE approaches, software developers need to manually edit it (as required) so that the desired final source code can be obtained. Even though this approach has been adopted to increase the productivity of the development team via code generation, manual addition has been demonstrated to be a time-consuming and error-prone task in practice.

The enterprise, where the case study (Section 4) was performed, adopted this generative MDE approach with the objective of reducing time to market and automating the development process. However, this need to edit manually represented a critical shortcoming from the perspective of software developers in the context of realistic scenarios in industry, where time is short, and budget is tight.

For example, software developers had to apply a great effort to implement simple features, such as the personalization of views, the implementation of the service calls, and the validation of the data on the client side. These are features that are commonly requested by customers. This resulted in (1) problems that are not related to the domain problems, which required rework on the software development; (2) difficulties in predicting the deployment date; and (3) investments in third-party tools for improving the productivity. Consequently, this led to an increase in the number of software developers, frequent extensions in the time of quality assurance, which caused delays in deploying the application, and a significant reduction in the company's profitability.

Moreover, the source code of enterprise applications is not generated completely, as previously mentioned. That is, developers often need to manually implement and edit the generated source code, which is a time-consuming and error-prone task. In this context, some problems have already been identified and reported in the current literature. In [5], Hutchinson and colleagues

highlight a set of negative effects that may be due to the adoption of generative MDE approaches, which are related to the following:

- **Maintenance:** The time that is required to maintain the software may be increased due to the need to keep models/code in sync, and the generated code may be difficult to understand;
- **Productivity:** Even though the effort to develop code may be reduced by automatic code generation, the effort to develop computer-readable models and implement model transformations is still relatively high; and
- **Portability:** Although the effort to migrate to a new platform can be reduced by simply applying a new set of transformations, the effort that is required to develop new transformations or customize existing ones is still quite high.

Still, the development of enterprise applications requires diverse skills and knowledge from the problem domain, business rules, process and programming issues. It becomes difficult to recruit an application developer who has the required knowledge at low cost. The proposed approach addresses precisely this question, as it is as pragmatic as possible of an approach, rather than a fundamentalist approach, to increase the productivity of the software developers.

3. The proposed approach

This section presents BRCODE (Brazilian Code), which is an interpretive MDE approach for supporting the development of fast-changing enterprise applications. Section 3.1 gives an overview of the development process that is adopted in our cases studies. Next, Section 3.2 describes the proposed approach. Section 3.3 introduces the key features of BRCODE. Then, Section 3.4 presents the architecture of BRCODE. Finally, Section 3.5 discusses details about implementation aspects.

3.1. Overview of the development process

Fig. 3 presents an overview of the adopted development process, along with the proposed approach. This process was used with both MDE approaches to build enterprise applications. Each step of this process is described as follows:

- **Step 1: Requirement definition and specification.** This step aims at defining and specifying the customer's requirements (Fig. 3(1)). The application scope is determined by defining business rules, goals, constraints, and high-level operational concepts. Software developers create a domain model to represent the main concepts that are found in the problem space. Based on this diagram, software developers generate the application database.
- **Step 2: Choice of MDE approach.** The development team chooses which MDE approach will be used in the project: the generative MDE approach or the interpretive MDE approach (Fig. 3(2)). As described in Section 2, the generative MDE approach generates source code from domain models that are created by software developers. The generated source code follows the Microsoft Application Architecture² (see Fig. 1). This architecture is a standard enterprise software architecture and was chosen by the company's software architects. If the interpretive MDE approach is chosen, software developers need to create the domain model of the application to be developed. Developers register the modules, the interface, and the business rules in the database.

² NET Application Architecture: <https://www.microsoft.com/net/learn/architecture>.

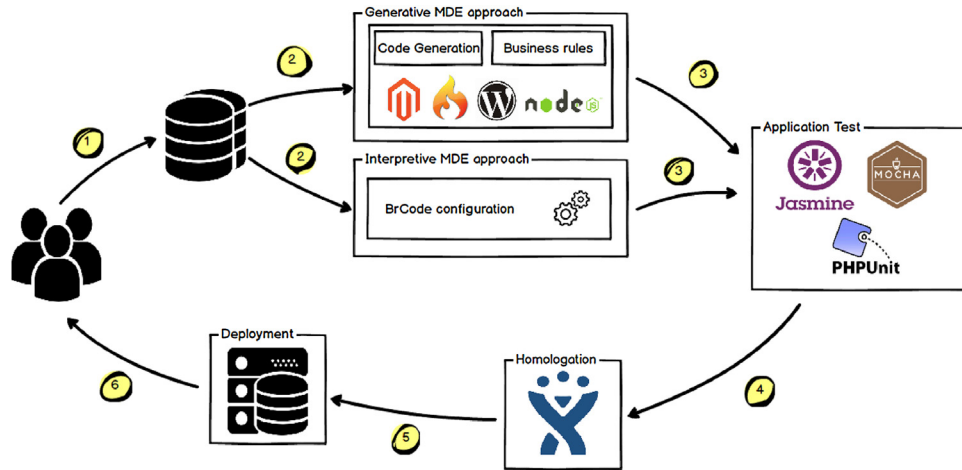


Fig. 3. Overview of the development process.

- **Step 3: Application test.** In this step, the software developer tests the application (Fig. 3(3)). Three kinds of tests are performed: unit, integration and system tests. Jasmine, PHPUnit, and Mocha are the main technologies that are used to test the developed enterprise applications.
- **Step 4: Homologation.** This step aims at publishing the software that is developed in a production environment (Fig. 3(4)). We check whether the developed enterprise applications meet customer requirements in a realistic production environment. After the application has been approved, users test it in a realistic production environment. This beta test is intended to detect possible faults that were not detected in the test phase, as well as to obtain a customer evaluation.
- **Step 5: Deployment.** After obtaining the initial acceptance by the customer, the application needs to be prepared for deployment (Fig. 3(5)). This step aims at performing activities and using technologies, such as a version control system, to deploy the application. Version control is used to store and track changes to the enterprise application files.
- **Step 6: Delivery.** This step involves the delivery of the developed application to the customer (Fig. 3(6)). The application is evaluated by the customer in the production environment. If any problem is identified, it will be reported to the development team.

3.2. Interpretive MDE approach based on stable architecture

The proposed approach identifies concepts that remain unchanged in the realms of the solution problem (e.g., style of software architecture) and the domain problem (e.g., customer

requirements), and then shapes the enterprise application architecture based on them. In this way, we avoid incorporating unstable concepts into the enterprise application architecture [8], which might cause changes that cut across the architectural components. That is, minor changes might cause an endless cycle of architecture refactoring [6].

In this sense, BRCODE consists of a pragmatic way of using concepts that remain unchanged in domain models to automatically build enterprise applications, thereby avoiding human interventions as much as possible; this strategy is commonly used in generative approaches. The instances of this domain model generate meta-data, which might be used to personalize customer interfaces, implement web services, and validate data in the user interfaces. These key features of BRCODE are described in Section 3.3. All applications that are developed using BRCODE follow the structure that is shown in Fig. 4 and discussed as follows:

1. **Database:** The software developers produce the application's database based on the domain model that was created according to the application requirements. The defined meta-data, e.g., database scheme, are formatted and interpreted throughout all application layers (Fig. 4(1)).
2. **Meta-data:** The developers generate meta-data based on the domain model that was created according to the customer requirements. They can define meta-data in the database, which are used throughout all layers of the application (Fig. 4(2)). For example, suppose an entity *person* has an attribute *name*. If a person should have a name, then this information can be found in the meta-data, e.g., *name.isNull = false*. This information serves to set up several parts of applications that are under

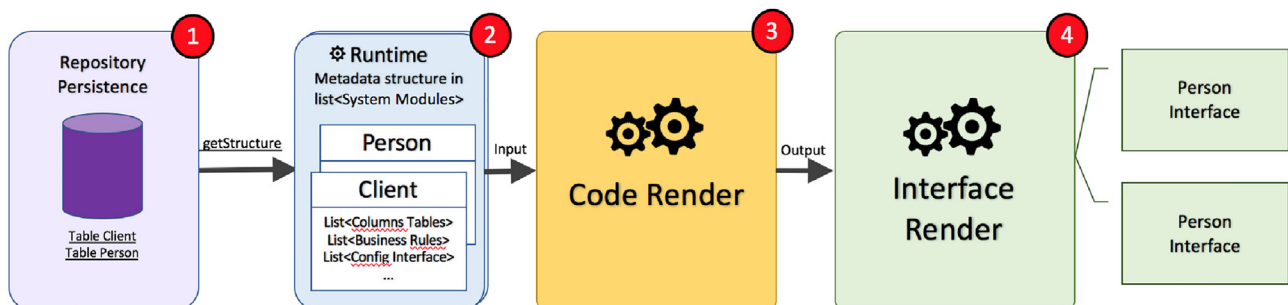


Fig. 4. Structure of the enterprise applications that are developed using BRCODE.

development, such as the style of the web page, business layers, data masks, and data types.

3. **Code renderer:** This component contains entities that remain unchanged and operations for manipulating these entities, which are organized through a stable architecture (Fig. 4(3)).
4. **Interface renderer:** This component renders views on the client side (Fig. 4(4)). The views are not static; that is, they are rendered based on the client requests. The views are rendered according the configurations that are present in the database. Developers set these view characteristics in the first step, during database definition.

3.3. Key features of BRCode

Based on our practical experience, we have identified several features that were common to all business applications that were developed in our company. In addition, we determined that most of the development effort and customer's change requests targeted such features, and that the quality of the source code was directly impacted by time-consuming, error-prone tasks of developers in changing such features. Thus, we understood that increasing the productivity and the quality of the applications would be achieved through the automatic execution of the development and maintenance of such features. The key features that are provided are listed as follows:

1. **Authentication and authorization feature.** Two closely related simple and easy-to-configure concepts are at the heart of security for enterprise applications: authentication and authorization. Authentication consists of the process of acquiring users' credentials and using those credentials to check the user's identity. Authorization is the process of enabling an authenticated user to access resources or functionalities of the enterprise application. Applications that are developed using the BRCode are fully supported by the authorization and authentication mechanisms. In addition, the BRCode assures that the authentication process must always precede to authorization one, and still allows the authentication of users as being anonymous. Usually, enterprise information systems let anonymous users connect and use the application resources. In this case, the BRCode authenticates the users as being anonymous. The technique allows the definition of an access profile and access permissions based on the profile in an easy way. Therefore, developers with no knowledge of how to implement authorization and authentication become capable of developing applications with complete access control.
2. **Mobile-friendly, mobile-optimization and responsive design for enterprise applications.** Several concepts that are related to User Experience (UX) are still unclear to software developers, such as mobile-friendly application, mobile optimization and responsive design. Even worse, they do not know: how to implement enterprise applications that support such concepts, where mobile first design fits in, or even what the best design strategy is for a particular enterprise application. Today, mobile-friendly enterprise applications are often created just as a reduced version of the enterprise application that is viewed on a desktop. Even though these smaller versions can be functional for mobile users, they end up not being as user friendly as they could be since they were designed for desktop users. When the enterprise applications are centered on mobile users upfront and, therefore, are better optimized and more responsive, the application resources (i.e., its functionalities) tend to be used more frequently. In contrast, when the use interfaces are not user-friendly, the use of enterprise applications on mobile devices may become prohibitive. The proposed approach provides mobile-friendly, mobile-optimized and responsive design for enterprise applications. That is, all generated applications support smart phones, tablets and desktops. Another issue that is encountered is mobile optimization. Although all mobile-optimized interfaces are mobile-friendly, not all mobile-friendly interfaces are mobile-optimized. With this in mind, our approach provides mobile-optimized applications by generating user interfaces for smaller screens up front, as opposed to just reducing their content later. That is, our mobile-optimization approach targets mobile users, rather than desktop users. This retargeting avoids the user interfaces being just shrunken-down versions of the desktop interfaces.
3. **Multilingual and easy-to-customize approach for enterprise applications.** Today, enterprise applications need to be adapted for users from different cultures or regions, or with different languages. However, this adaption is a time-consuming task, as developers need to make several adaptations to the source code manually for each new language. The BRCode has a built-in internationalization mechanism, which allows the enterprise applications to have internationalization support. This means that developers do not need to code; they just need to customize the application's terms for a particular language.
4. **Composite dashboard based on widgets.** Enterprise applications often need to provide dashboards, which display strategic information for decision makers. However, the fundamentalist model-driven development approaches overlook the creation of dashboards. Consequently, developers end up manually creating monolithic dashboards based on data that are stored in the database server. For this, a full-stack developer needs to know what data should be retrieved from the database, which architectural components should be created or modified to obtain these data, and which UI (User Interface) components must be used to display the retrieved data. With the BRCode, an inexperienced developer can easily create a dashboard by aggregating a set of widgets. Each widget is automatically generated for exhibiting data regarding a particular entity that are stored in the database. This results in a composite dashboard that is formed by widgets, instead of a monolithic dashboard that consumes data on the server. The developers who are using the BRCode only need to specify what data (of a particular entity) should be displayed in the dashboard.
5. **CRUD operations, filter and pagination resource.** The BRCode provides the four basic functions that are related to data persistence: create, read, update, and delete (namely, CRUD). For this, sophisticated UI components are used to facilitate viewing, searching, changing and deleting information. The BRCode's UI components are based on the front-end component library Bootstrap. In addition, the functions of filter and pagination are also supported at large. Users can filter data in tables, reapply a filter to obtain updated data, or even clear a filter to exhibit all of the data again. The pagination resource offers a lightweight directive that covers the listing of large data sets and the enabling and disabling of UI components properly. The BRCode automatically generates all these functions.
6. **Smart cache to boost performance.** The interpretive MDE approach analyzes and interprets software models at run time without generating any source code; instead, templates are sent as metadata to the application that interprets them. The interpretation in this MDE approach does not require any manual implementation to be executed; instead, a set of predefined generic requirements are configured in the metadata and executed at run time. The BRCode's components are designed to transpose the meta-data structure, which is responsible for storing the characteristics of the software requirements. As BRCode works with the interpretive MDE approach, its components are modeled to transpose the

meta-data structure, which is responsible for storing the characteristics of the software requirements. The interpretation of these models can negatively affect execution time performance. However, this problem has been bypassed in BRCODE through a smart cache structure. In [9], the author supports that this problem can be mitigated using cache. BRCODE caches strategic sectors of meta-data structures by sharing interpreted meta-data structures at run time. In practice, the performance gain has been significant. After the first user accesses a functionality, BRCODE interprets and generates the cache file, and the next users of the same functionality benefit from the shared cache. Another feature of the BRCODE cache is that its algorithm has the ability to detect when changes occur in the base structure of the meta-data (identifying changes in requirements), clear the cache of previously interpreted structures that have become obsolete after the changes, and keep the structures up to date based on the meta-data base model. BRCODE has a positive characteristic in relation to the results that are reported in [2]. It is a widely known problem that .NET does not support software updates and caches without restarting its processes. This is because the code in the .Net architecture needs to be restarted when changes to its code update the GAC (Global Assembly Cache), which is a machine-wide code cache that is found in each computer where the common language run time is installed. BRCODE has been developed in .Net. However, it is possible to update the software at run time without shutting down the system or even restarting processes on the server. This feature is possible because of its intelligent caching system, where it handles interpreted data that do not affect the code structure of the .Net application.

3.4. Architecture of BRCODE

All enterprise applications that are produced using BRCODE are based on a multi-layered architecture. As shown in Fig. 5, the proposed architecture is composed of five layers. This architecture consists of a client-server architecture in which the presentation, business rule processing, and data management functions are physically separated in the front-end and the back-end. Fig. 6 shows how this client-server architecture is structured in terms of the front-end, back-end and database, and clarifies the differences between the generative MDE and BRCODE architectures. The front-end layer runs on the client side, while the back-end and database run on the server side. Each layer that is shown in Fig. 5 is described as follows:

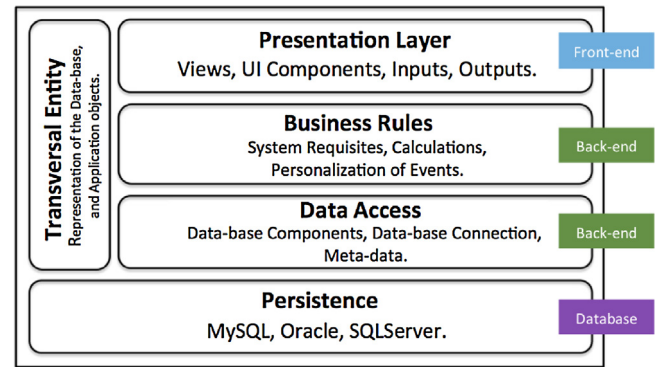


Fig. 5. The proposed architecture which is composed of five layers.

- 1. Presentation Layer:** This layer is responsible for formatting information and delivering it to users. The presentation layer contains responsive views that are created through the interpretation of the meta-data that are received from the layer below. An example of a feature of this layer is the rendering of meta-data for multiple views, e.g., smart phone, tablet and desktop applications.
- 2. Business Rules Layer:** This layer is implemented by core elements of BRCODE and focuses on business rules that are often found in enterprise applications, such as filtering and searching. Invariant and algorithms stay in this layer. Some of the business rules are implemented directly in the database through triggers, procedures, functions, constraints or meta-data about business operations according to business needs. The internal schema will be the product that is generated by the mapping that is performed in the conceptual schema for a specific syntax, which, in this case, is Structured Query Language (SQL). Some of these are generic and involve the maintenance of concepts of business, such as inclusion, exclusion, change, and research of a concept of the business model.
- 3. Transversal Entity Layer:** This layer is cross-cutting because it contains functions of common interest for the presentation, business, and data layers. It is responsible for managing application objects between these layers. These objects contain functions for delivering data collections from one layer to another.
- 4. Data Access Layer:** This layer performs the system's data access operations. It separates the business rule layer from database access operations. In addition, a smart cache reduces the

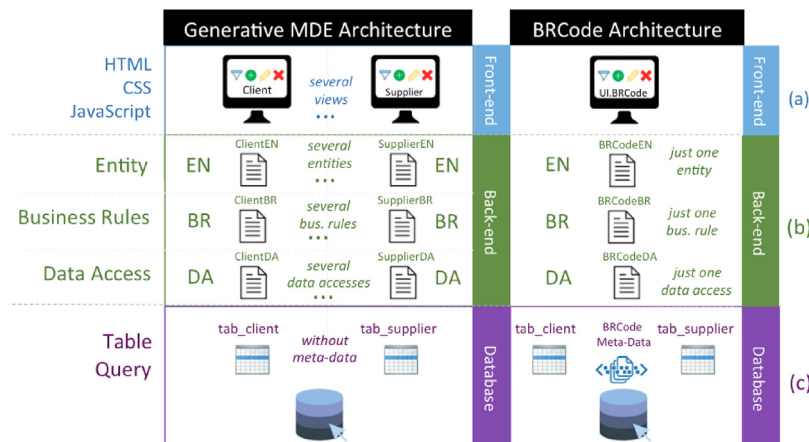
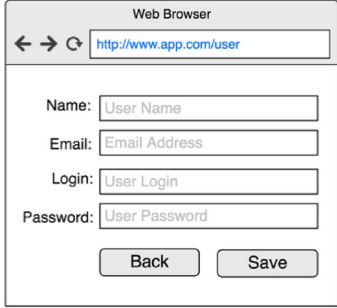


Fig. 6. Generative MDE and BRCODE architectures.

User Form



Generative MDE Approach

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4 <form action="/userForm.aspx" runat="server">
5 <div>
6   Name: <input type="text"/><br>
7   Email: <input type="text"/><br>
8   Login: <input type="text"/><br>
9   Password: <input type="password"/><br>
10  <button type="button" onclick="back()">Back</button>
11  <button type="button" onclick="save()">Save</button>
12 </div>
13 </form>
14 </body>
15 </html>

```

Interpretive MDE Approach

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4 <brCode:ListFields id="lstFields"/>
5 <button type="button" onclick="back()">Back</button>
6 <button type="button" onclick="save()">Save</button>
7 </body>
8 </html>

```

Fig. 7. An example of User Form, implemented using the generative MDE approach and the interpretive MDE approach.

connections and transactions between the application and the database. The data in the cache are checked and updated as new settings are specified.

5. **Data Layer:** This layer is responsible for storing system data. This layer also manages meta-data configuration. In this way, this layer concentrates the meta-data of the business rules, the interface, and the access permissions of users.

Fig. 6 shows the BRCODE architecture in comparison to the generative MDE architecture. The main differences are discussed as follows:

- **Front-end.** In the generative MDE approach, developers need to manually create a user interface for each new entity or functionality, or even change them separately according to change requests. In Fig. 6(a), two interfaces were created: one for *Client* and one for *Supplier*. For example, if a ZIP code has to be added to the *Supplier* interface, the developer will have to change HTML, CSS, and JavaScript code. Suppose that every customer must have an address. There are two options for implementing this requirement: The first is to change the client interface with the address attributes. The second is to create a new interface. Again, in both cases, the developer would have to manipulate HTML, CSS and JavaScript code. In the generative MDE approach, most intangible concepts (e.g., client and supplier) and concepts that are unlikely to remain stable over time are generated and created manually. Thus, maintaining and evolving code becomes an activity that consumes time and is prone to errors. In contrast, in the BRCODE approach, such concepts are represented in an abstract way so that the source code can remain stable over time. In Fig. 6(a), *UI.BRCODE* represents a stable component. Instead of having two interfaces, there is a single component that will render the *Client* and *Supplier* interfaces at run time according to the meta-data of the entities.
- **Back-end.** Both approaches have classes that concern domain entities, business rules, and data access objects. In the generative MDE approach, developers need to manually create entities, business rules and data access objects for each new use case, or change them to implement change requests. In Fig. 6(b), there is an entity, a business rule and a data access object for implementing functions that are related to *Client* or *Supplier*. In addition, stable concepts are defined so that any change can be made. Developers do not need to create domain entities, business rules, and data access objects when new use cases need to be implemented or change requests are required. This

stability is achieved through the following abstractions: *BRCODEEN*, *BRCODEBR*, and *BRCODEDA*. These three abstractions are stable concepts that remain unchanged over time. They interpret meta-data that are organized in a data structure, such as *HashMap* or *List*. Thus, change requests cause changes in the meta-data, thereby avoiding any modification to the code.

- **Database.** Databases are located on servers and provide meta-data for the application. In Fig. 6(c), the presence of meta-data is the main difference between the generative MDE approach and BRCODE.

3.5. Implementation aspects

The technologies that were used followed performance, modularity and usability requirements. BRCODE was implemented using the Visual Studio development platform, the C# language, and ASP.NET — an open-source web framework for building modern web apps and services with ASP.NET.³ We used ASP.NET because it creates web applications based on HTML5, CSS, and JavaScript that are simple, fast, and can scale to millions of users [10]. The databases Microsoft Sql Server, MySQL and Oracle were used.

Moreover, the usability of the user interfaces was achieved by separating content (HTML), its presentation (CSS 3), and functions (JavaScript). In addition, we used the most popular front-end component library, which is called Bootstrap, to build responsive, mobile-first projects on the web. Using Bootstrap, we could quickly prototype user interfaces and build mobile-friendly, mobile-optimized and responsive designs for Web enterprise applications. For this purpose, BRCODE makes use of many features of Bootstrap, including Sass variables and mixins, a responsive grid system, extensive prebuilt components, and powerful plugins that were built on jQuery⁴ — an open-source cross-platform JavaScript library that is designed to simplify the client-side scripting of HTML.

Fig. 7 presents an illustrative example of user interfaces that were implemented using the generative MDE approach (Section 2) and the BRCODE (Section 3). There is a clear difference between them. First, 15 lines of code were required to implement the *User Form* using the generative MDE approach, whereas just 8 lines

³ <https://www.asp.net/>.

⁴ <https://jquery.com/>.

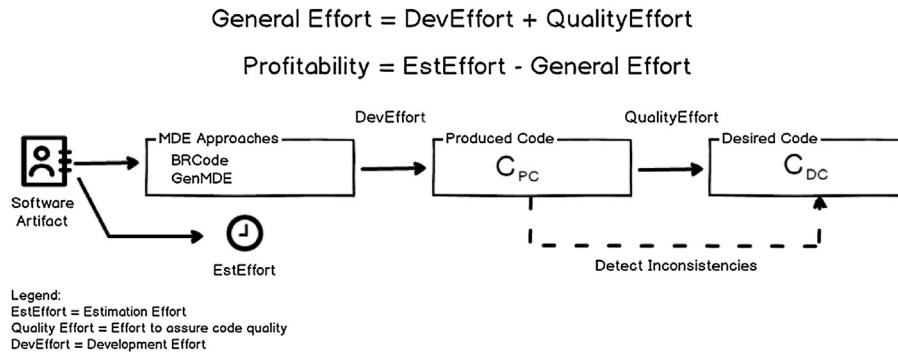


Fig. 8. Variables related to productivity and profitability.

were required using the interpretive MDE approach, i.e., BRCODE. The generative MDE approach required four lines (lines 6–9) to create the labels and input the texts that are found in the User Form. In contrast, the interpretive MDE approach required only one line (line 4). In line 4, the element *brCode:ListFields* consists of the view components that are used to render all labels and input texts based on the received meta-data.

If changes are requested, developers will invest more effort by altering many lines of code. For example, if three new fields were requested (such as first name, last name and phone) in this interface, developers would have to manually change the interface code. However, this change would not cause any change in the interface that is implemented using the proposed approach. This stability in the face of change requests helps to reduce maintenance effort and increase code quality. In the following section, we report a case study in industry for comparing the generative MDE approach and the BRCODE.

4. Case study

This section presents the main decisions that underlie the case study that was carried out to evaluate the proposed technique. In particular, we are concerned with investigating the impact of the proposed approach (Section 3) on the development effort and financial gains. Section 4.1 introduces the objective and the explored research questions. Section 4.2 explains how the analyzed variables were quantified. Section 4.3 presents the evaluation procedures that were defined to achieve the study objective and the research questions that were formulated. All these methodological steps are based on the well-known guidelines that are presented in [11], which describe how to design and conduct empirical studies.

4.1. Objective and research questions

This study seeks to investigate the effects of the proposed technique on two variables: the development effort and the financial gains. These effects are explored from the perspective of developers in the context of the development of Enterprise Information Systems in industry. The objective of this empirical study is stated based on the GQM template [11], as follows:

Analyze MDE approaches
for the purpose of investigating their effects
with respect to productivity and profitability
from the perspective of software developers
in the context of software development in industry.

4.2. Variables and quantification methods

This section describes the dependent and independent variables that are measured in our evaluation.

Independent variable. It indicates which MDE approach is used and can assume one of two values: generative MDE (genMDE) and interpretive MDE (BRCODE).

Dependent variables. This study investigates the effects of genMDE and BRCODE on the following variables.

- **General Effort (GenEffort):** It describes the amount of effort (in hours) that is required to produce the desired code. It is composed of the effort that is needed to develop (DevEffort) plus that to perform quality assurance (QualityEffort). Fig. 8 shows an overview of the variables that are related to productivity. DevEffort and QualityEffort are better described as follows:

- **DevEffort:** It consists of the effort that is invested by the developers in analyzing the customer's requirements, creating the software design, implementing this design, and testing and deploying the produced software. A software system will only be delivered when its quality is good. Therefore, testing is a fundamental part of development. As shown in Fig. 8, DevEffort represents the effort that is expended in using a generative or interpretive MDE approach to produce the source code of the application. Regardless of which MDE approach is used (generative or interpretive MDE), the software development process is error-prone, as developers manipulate the code manually. Thus, the produced source code ends up requiring future improvements after deployment (see Fig. 3). Code with problems is known as *produced code* (C_{PC}).
- **QualityEffort:** It consists of the effort that is invested in implementing all improvements that are detected by the customers, to produce the desired code (C_{DC}) from C_{PC} . These improvements can range from changing the color of a UI component to changes in the performance of running a feature.

It is important to note that every project that was contracted with the company had a period of 6 months of support. During this period, customers could request improvements at no extra cost. Note also that if the effort that is spent to assure quality is low, then the general effort tends to be low as well. If the DevEffort is low, but produces source code with several improvements, then the effort to be invested in fixing such problems tends to be high; hence, QualityEffort tends to assume a high value.

- **Profitability (Profit):** It represents the financial gains of the company per project. The profit is calculated based on the estimated effort (EstEffort) and GenEffort. Fig. 8 shows the

equation that is used to compute *Profit*. The effort estimate was calculated based on function points, regardless of the type of MDE approach that was used in our case study (described in Section 4.3). A function point can be briefly defined as a unit of measurement that expresses the amount of functionality an enterprise application provides to a user. Based on the equation in Fig. 8, financial gains rise when *EstEffort* is higher than *GenEffort*. That is, *Profit* represents the total effort that is saved. In this work, the profit is quantified in hours.

- **Profitability Rate (ProfitRate):** The ratio of the number of systems that generate profits in a project to the total number of systems that are implemented in a project. The value of *ProfitRate* ranges from 0 to 1.
- **Return on Investment (ROI):** It indicates the amount of time that is needed to recover the investments in terms of hours. Eq. (1) presents the ROI equation and the involved variables. ROI can be applied to evaluate a project or a module in terms of hours.

$$ROI = \frac{EstEffort - (DevEffort + QualityEffort)}{(DevEffort + QualityEffort)} \quad (1)$$

4.3. Evaluation procedures

Context. We performed a case study at a software production company in Brazil. The company specifies, develops and tests enterprise information systems for construction companies, such as ERP (Enterprise Resource Planning) systems. The company was contracted to develop an ERP system that contains 17 modules, using Microsoft's Visual Studio development platform and the C# language. The technologies that were used followed portability requirements, using the .NET framework. Initially, the system was implemented using the generative MDE approach (described in Section 2) in 2015. The development team (in both approaches) was formed by three developers (with two years of experience with software development) and a software architect (with 7 years of experience). In 2017, the company was hired again to modernize the system by producing responsive web applications. At this time, the 17 modules of the ERP system were redeployed from scratch using BRCODE. A new development team with same level of experience was allocated, which consisted of three developers and a software architect. Since the development teams were different, the learning effect was discarded. Both development teams versioned the developed systems using the version control system SVN.⁵ In addition, all information regarding the progress of project management activities was recorded in a software project management web application called RedMine.⁶

Quantitative analysis. We used descriptive statistics to identify trends and analyze the distribution of the collected data. Tables and Box-plot were employed to graphically illustrate the results. The presence or lack of trends and patterns acted as a driver for further investigations. In particular, we aimed at introducing and presenting interesting aspects of the collected data, rather than inferring correlation or producing a probabilistic formulation. In addition, we analyzed possible outliers and, when acceptable, they were removed. Outliers are extreme values of the measured variables that may influence the conclusions of the study [11]. In our study, the outliers that were identified did not represent extraordinary exceptions; hence, they were not removed.

Table 1

Descriptive statistics of the general effort.

Variable	Method	SD	Min	25th	Med	75th	Max	Mean	% Diff
General effort	genMDE	40.77	4.5	17.5	32.5	59	163	43.76	93.75%
	BRCODE	2.08	1	1	1	4	7	2.74	

SD: standard deviation, Min: minimum, Med: median, Max: maximum, Diff: difference.

5. Results

This section presents the results that were obtained from the analysis of the collected data. Section 5.1 presents the general effort results. Section 5.2 presents the results that concern the profitability. Section 5.3 presents the results about the profitability rate. Finally, Section 5.4 presents the results that concern the Return on Investment. The study data can be found in Table A.6 (in Appendix A).

5.1. General effort

Table 1 and Fig. 9 present the descriptive statistics and box plot, respectively, of the general effort.

The values of the general effort were measured in minutes. These statistics help us pinpoint the central tendencies, spreads of values around them, and the difference between the means. The general effort analysis involved an examination across cases of a single variable, focusing on three characteristics: the distribution, the central tendency, and the dispersion.

As previously mentioned (in Section 4.3), 34 modules were developed: 17 using the generative MDE approach and 17 using the BRCODE. The central tendency was calculated using the two most commonly used statistics: the mean and the median. The most interesting result is that *the general effort was, on average, approximately 43.76 and 2.74 min using the generative MDE approach and the BRCODE, respectively*. This means that the BRCODE's general effort is only 6.25% of that of the generative MDE. That is, 93.75% of the general effort in the generative MDE

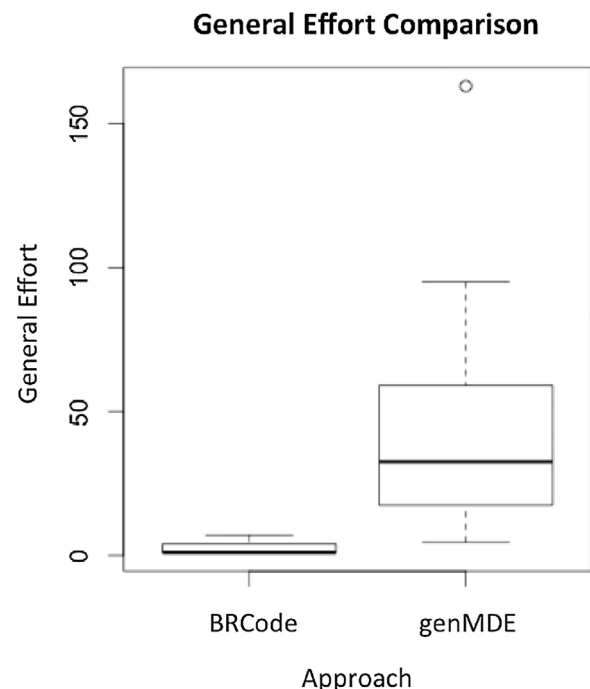


Fig. 9. General effort.

⁵ <https://subversion.apache.org/>.

⁶ <http://www.redmine.org/>.

approach represents extra effort that needs to be applied. The median measures corroborate to this result: 40.77 and 2.08 min. This implies that the general effort that is required for development using the BRCODE was very low compared to the generative MDE approach. Hence, if software production organizations, where resources and time are usually tight, were to use the BRCODE approach, they would significantly decrease the development effort.

Table 1 shows that software developers usually applied more effort to developing a module using the generative MDE approach. The times that were spent were significantly higher compared to the BRCODE approach. For example, the max column shows that developers spent 163 min using the genMDE, whereas developers invested only 7 min using the BRCODE.

In addition, we used the standard deviation and the 25th and 75th percentiles, which were computed to determine the dispersion of the data around the reported tendency. The general effort in the generative MDE approach tended not to concentrate around the central tendency; instead, it spread out over a large range of values. In BRCODE, this spread did not exist.

Indeed, at 40.77 and 2.08 min, the standard deviation measures indicated that the *general efforts were notably different using the generative MDE approach and the BRCODE*. This finding may help software analysts better understand the general effort that is required in using the generative MDE approach and the BRCODE. Today, software analysts make decisions essentially based on their judgment, and check whether the actual general development effort agrees with the expected value (or not).

5.2. Profitability

This section explores the effects of MDE techniques on the profitability variable. Table 2 shows the descriptive analysis of the profitability (profit). Fig. 10 shows the descriptive analysis of the profitability (profit). Fig. 10 presents the box plot. The average profit of the genMDE is a negative value. This is because the applied effort was higher than the estimated effort, thereby causing bias in the time that developers spent. Fig. 10 shows that the data of BRCODE are upper-skewed for values that are greater than zero. Furthermore, the median is located on the top of the box plot, which indicates that values concentrate in that region. In contrast, the median for genMDE is zero, i.e., the half of values converge to profits of time.

5.3. Profit rate

Fig. 11 shows the results on the profit rate, which refers to the ratio of the number of modules with profit and the total number of modules. The profit rate indicates the percentage of modules that resulted in profit. The BRCODE has a profit rate of 0.7 (70%) and the generative MDE approach resulted in a profit rate of 0.48 (48%). This means that the majority of modules that were developed in the generative MDE approach did not return a profit.

5.4. Return on Investment (ROI)

Table 3 presents the descriptive analysis of the ROI. The collected data highlight that the ROI that was obtained using the

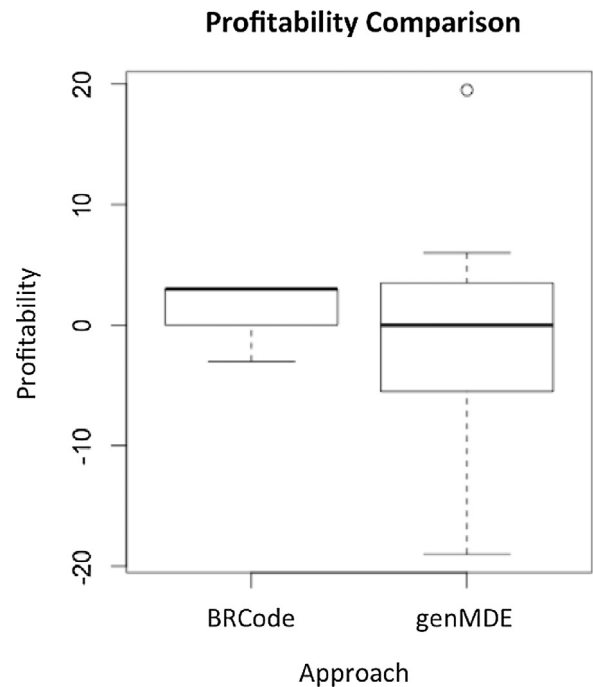


Fig. 10. Profitability.

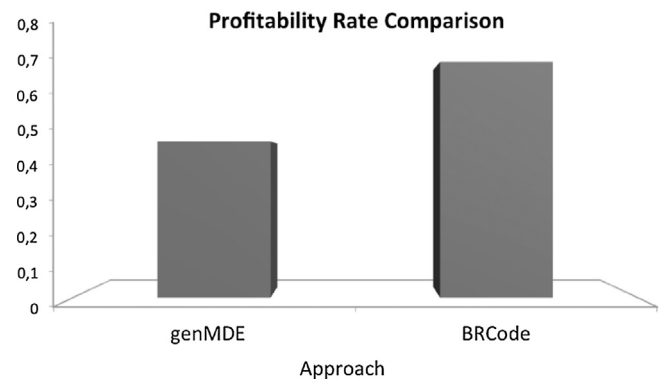


Fig. 11. Profitability rate.

BRCODE is significantly greater than that obtained using the generative MDE approach. The data indicate that software development teams that use the generative MDE approach might reduce the ROI by 96%, compared to the BRCODE approach.

Fig. 12 presents the data distribution. The ROIs were evaluated for the BRCODE and the generative MDE approach. This graph shows that the BRCODE has a strong advantage over the genMDE approach. The median for BRCODE indicates the most ROI values are concentrated in the top of the plot. In contrast, the median for genMDE is very low. This indicates that half of the values of the ROI for the generative MDE approach are between 0 and 0.5.

Table 2
Descriptive statistics of the profitability.

Variable	Treatment	SD	Min	25th	Med	75th	Max	Mean	% Diff
Profitability	genMDE	9.36	-19	-5.50	0	3.5	19.5	-1.71	234.88%
	BRCODE	2.08	-3	0	3	3	3	1.26	

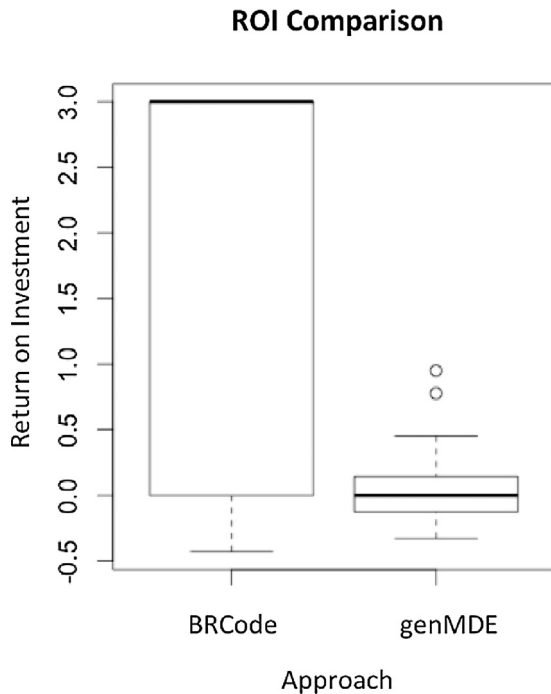
SD: standard deviation, Min: minimum, Med: median, Max: maximum, Diff: difference.

Table 3

Descriptive statistics of Return on Investment (ROI).

Variable	Treatment	SD	Min	25th	Med	75th	Max	Mean	% Diff
Return on Investment	genMDE	0.36	3.3	−0.12	0	0.14	0.95	0.07	95.37%
	BRCODE	1.6	0.43	0	3	3	3	1.54	

SD: standard deviation, Min: minimum, Med: median, Max: maximum, Diff: difference.

**Fig. 12.** Return on Investment.

6. Related work

Academia recognizes the importance of code generation from software models, according to mapping studies that concern this subject, such as [12–14]. Table 4 summarizes the main aspects of the related mapping studies that we found. The data suggest that extensive research has been conducted on MDE approaches, in particular, regarding code generation. We briefly discuss these articles as follows.

Table 4

Resume of the related mapping studies.

Articles	Year	Number of studies	Search range	Purpose
Syriani et al. [12]	2017	440	2000–2015	Analyse the TBCG state-of-art.
Seriai et al. [13]	2014	752	2000–2012	Raise the state-of-art about the validation of visualization tools.
Mehmood and Jawawi [14]	2013	65	1997–2012	Scrutinize the state-of-art about the code generation from aspect models.

Table 5

Comparative table of related works.

Articles	Proposes a MDE technique	Validation	Case	Evaluates/discuss quality or performance variables
Rosales-Morales et al. [15]	No	Yes	No	Yes
Hutchinson et al. [5]	No	Yes	No	Yes
Buchmann and Westfechtel [16]	Yes	No	No	No
Mikami et al. [17]	Yes	No	No	No
Azadegan et al. [18]	No	Yes	Yes	Yes
Sun et al. [19]	No	Yes	Yes	Yes

Syriani et al. [12] selected 440 studies, which cover 15 years of research on TBCG. TBCG refers to techniques that produce code from high-level specifications, such as software models and templates. They emphasized that researchers are still engaged in producing template-based approaches. In addition, their results imply that MDE approaches are being applied actively in software projects. This indicates that the practice of MDE in industry no longer presents crucial problems that need to be resolved.

Seriai et al. [13] analyzed research that was published over a 12-year period on the validation of software visualization tools. Their selection process found 752 articles. Their analysis of these articles identified an absence of validation studies for visualization tools. In other words, the evaluations that focus on the effectiveness of current visualization tools are limited to qualitative studies.

Mehmood and Jawawi [14] mapped studies that focus on the generation of aspect-oriented code from models. They selected 65 studies from 255 studies, which were published over a 15-year period. The results show a concentration of new proposals. This implies that this area is at the problem-solving stage.

Table 5 presents a comparison of related works. Experimental works have been playing a key role in evaluating the effectiveness of MDE approaches [5,15]. In [15], the authors performed a comparative analysis of the Integrated Development Environments (IDEs) for MDE. They used a set of quality metrics to evaluate these tools, such as effectiveness, productivity, safety, and satisfaction. In [5], the authors evidence the productivity gains of the MDE approach over the traditional and code-centric methods.

Some approaches were proposed with the objective of generating code from software models. In [16], the authors present a transformation tool for generating Java source code from class diagrams and other Java source code. In this case, the class diagrams are the models that drive the development. This tool was built with the purpose of updating the class diagrams from source codes interactively. This process is well known as round-trip engineering. In this work, the database schema is the artifact that is responsible for maintaining the integrity of the stored data. Because of its critical function, proper testing of

Table A.6
Study data.

Modules	Effort estimation		Development effort		Quality assurance		General effort		Profitability	
	BRCode	genMDE	BRCode	genMDE	BRCode	genMDE	BRCode	genMDE	BRCode	genMDE
M1	4	40	5	38.5	0	6	5	44.5	–1	–4.5
M2	4	60	3	55	1	13.5	4	68.5	0	–8.5
M3	4	20	3.5	18.5	1	7	4.5	25.5	–0.5	–5.5
M4	4	40	6	32.5	0	7.5	6	40	–2	0
M5	4	30	1	31	2	1.5	3	32.5	1	–2.5
M6	4	8	1	7.5	0	0	1	7.5	3	0.5
M7	4	8	1	4.5	0	0	1	4.5	3	3.5
M8	4	20	1	17.5	0	0	1	17.5	3	2.5
M9	4	80	1	76	0	0	1	76	3	4
M10	4	30	1	16	0	8	1	24	3	6
M11	4	36	4	36	0	17.5	4	53.5	0	–17.5
M12	4	5	1	7	0	0	1	7	3	–2
M13	4	8	1	5.5	0	0	1	5.5	3	2.5
M14	4	100	7	82	0	13	7	95	–3	5
M15	4	150	1	163	0	0	1	163	3	–13
M16	4	40	4	39	0	20	4	59	0	–19
M17	4	40	1	20.5	0	0	1	20.5	3	19.5
Total	68	715	42.5	650	4	94	46.5	744	21.5	–29

the database schema is an important task. Mikami et al. [17] present an approach to the domain-driven development (DDD) and use of an object-relational mapping (ORM) tool that has the ability to evolve database schemes. The main purpose is to encapsulate a business model in the global architectural context and the logic of structuring the business at the design level.

Some case studies were found that consider productivity in an enterprise environment. However, they did not evaluate the effects that a new MDE technique has in relation to an older approach. In [18], the authors carried out a case study of software models before the beginning of the software process. In [19], the authors evaluated some ERPs in an industrial environment.

7. Conclusion and future works

This article proposed the BRCode, which is an interpretive MDE approach that is based on a stable architecture for improving the developers' productivity and the financial gains of companies. The BRCode was evaluated through a case study at a software production organization in Brazil. The case study involved two development teams, who used either the generative MDE approach or the BRCode to implement 17 modules of an enterprise application. The results were encouraging and showed the potential of the BRCode as a tool for fostering productivity and financial gains in software production organizations.

BRCode is an evolving model. We have identified the following future research directions: (1) allowing developers to create native applications for Android and iOS in mobile devices; (2) adding a feature to enable the development of user interfaces, using emerging front-end technologies, such as ReactJS and AngularJS; (3) allowing alternative implementations of features as micro-services; (4) replicating and expanding the evaluation by applying BRCode to the development of more complex enterprise applications in a company, focusing especially on software production organizations; and (5) testing the BRCode with volunteers who will answer a questionnaire based on the Technology Acceptance Model (TAM) [20] to evaluate its usability.

Appendix A. Study data

Table A.6 shows the data collected in our study.

References

- [1] M. La Rosa, M. Dumas, R. Uba, R. Dijkman, Business process model merging: an approach to business process consolidation, *ACM Trans. Softw. Eng. Methodol.* (TOSEM) 22 (2013) 11.
- [2] M. Overeem, S. Jansen, An exploration of the 'it' in 'it depends': generative versus interpretive model-driven development, *MODELSWARD* (2017) 100–111.
- [3] V. García Díaz, N. Valdez, E. Rolando, J.P. Espada, P.G. Bustelo, B. Cristina, J.M. Cueva Lovelle, C.E. Montenegro Marín, A brief introduction to model-driven engineering, *Tecnura* 18 (2014) 127–142.
- [4] B. Hailpern, P. Tarr, Model-driven development: the good, the bad, and the ugly, *IBM Syst. J.* 45 (2006) 451–461.
- [5] J. Hutchinson, J. Whittle, M. Rouncefield, S. Kristoffersen, Empirical assessment of MDE in industry, 33rd International Conference on Software Engineering (ICSE), IEEE, 2011, pp. 471–480.
- [6] M.E. Fayad, A. Altman, Thinking objectively: an introduction to software stability, *Commun. ACM* 44 (2001) 95–98.
- [7] K. Czarnecki, U.W. Eisenecker, K. Czarnecki, *Generative Programming: Methods, Tools, and Applications*, vol. 16, Addison Wesley Reading, 2000.
- [8] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Longman Publishing Co., Inc., 2002.
- [9] J. Buffenbarger, Amake: cached builds of top-level targets, *Comput. Lang. Syst. Struct.* 50 (2017) 20–30.
- [10] Microsoft, *AspNet.net*, (2017) . <https://www.asp.net/>.
- [11] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, *Experimentation in Software Engineering*, Springer Science & Business Media, 2012.
- [12] E. Syriani, L. Luhunu, H. Sahraoui, Systematic Mapping Study of Template-Based Code Generation, (2017) arXiv preprint arXiv:1703.06353.
- [13] A. Seriai, O. Benomar, B. Cerat, H. Sahraoui, Validation of software visualization tools: a systematic mapping study, 2014 Second IEEE Working Conference on Software Visualization (2014) 60–69.
- [14] A. Mehmood, D.N. Jawawi, Aspect-oriented model-driven code generation: a systematic mapping study, *Inf. Softw. Technol.* 55 (2013) 395–411 Special Section: Component-Based Software Engineering (CBSE), 2011.
- [15] V.Y. Rosales-Morales, G. Alor-Hernández, J.L. García-Alcaráz, R. Zatarain-Cabada, M.L. Barrón-Estrada, An Analysis of Tools for Automatic Software Development and Automatic Code Generation, *Revista Facultad de Ingeniería Universidad de Antioquia*, 2015, pp. 75–87.
- [16] T. Buchmann, B. Westfechtel, Using triple graph grammars to realise incremental round-trip engineering, *IET Softw.* 10 (2016) 173–181.
- [17] M.M. Mikami, K.M. Sandrino, M.S.M.G. Vaz, An approach to modeling and evolution of database model through the entity framework code first, *Iberoam. J. Appl. Comput.* 5 (2016).
- [18] A. Azadegan, K.N. Papamichail, P. Sampaio, Applying collaborative process design to user requirements elicitation: a case study, *Comput. Ind.* 64 (2013) 798–812.
- [19] H. Sun, W. Ni, R. Lam, A step-by-step performance assessment and improvement method for ERP implementation: action case studies in Chinese companies, *Comput. Ind.* 68 (2015) 40–52.
- [20] F.D. Davis, Perceived usefulness, perceived ease of use, and user acceptance of information technology, *MIS Q.* (1989) 319–340.