Software and Systems Modeling (2021) 20:1919–1943
https://doi.org/10.1007/s10270-021-00872-3

SPECIAL SECTION PAPER

Applying MDD in the content management system domain

Scenarios, tooling, and a mixed-method empirical assessment

Dennis Priefer[1] · Wolf Rost[1] · Daniel Strüber[2] · Gabriele Taentzer[3] · Peter Kneisel[1]

Received: 1 March 2020 / Revised: 19 January 2021 / Accepted: 1 February 2021 / Published online: 25 February 2021

Abstract
Content management systems (CMSs) such as Joomla and WordPress dominate today's web. Enabled by standardized extensions, administrators can build powerful web applications for diverse customer demands. However, developing CMS extensions requires sophisticated technical knowledge, and the complex code structure of an extension gives rise to errors during typical development and migration scenarios. Model-driven development (MDD) seems to be a promising paradigm to address these challenges; however, it has not found adoption in the CMS domain yet. Systematic evidence of the benefit of applying MDD in this domain could facilitate its adoption; however, an empirical investigation of this benefit is currently lacking. In this paper, we present a mixed-method empirical investigation of applying MDD in the CMS domain, based on an interview suite, a controlled experiment, a field experiment, and case studies. During the experiments, we used JooMDD, an MDD infrastructure instantiation for CMS extensions. This infrastructure, which is also presented in this work, consists of a DSL with model editors, code generators, and reverse engineering facilities. We consider three scenarios of developing new (both independent and dependent) CMS extensions and of migrating existing ones to a new major platform version. The experienced developers in our interviews acknowledge the relevance of these scenarios and report on experiences that render them suitable candidates for a successful application of MDD. We found a particularly high relevance of the migration scenario. Our experiments largely confirm the potentials and limits of MDD as identified for other domains. In particular, we found a productivity increase up to factor 11.7 and a quality increase up to factor 2.4 during the development of CMS extensions. Furthermore, our observations highlight the importance of good tooling that seamlessly integrates with already used tool environments and processes.

Keywords Model-driven development · Content management systems · Empirical assessment

Communicated by Tao Yue, Man Zhang, and Silvia Abrahao.

✉ Dennis Priefer
Dennis.Priefer@mni.thm.de

Wolf Rost
Wolf.Rost@mni.thm.de

Daniel Strüber
d.strueber@cs.ru.nl

Gabriele Taentzer
taentzer@informatik.uni-marburg.de

Peter Kneisel
Peter.Kneisel@mni.thm.de

[1] Institute for Information Sciences, Technische Hochschule Mittelhessen, Gießen, Germany

[2] Radboud University, Nijmegen, the Netherlands

[3] Philipps-Universität Marburg, Marburg, Germany

1 Introduction

Model-driven development (MDD, [54]) has been conceived as a development paradigm that aims to increase developer productivity and software quality by raising the abstraction level via the use of models. For over 15 years, many efforts have been made to empirically investigate this proposed benefit in various software domains, including telecommunications [2], finance [16], embedded systems [18], and conventional web applications [36]. A domain that has received little attention so far, despite its large-scale practical importance, are content management systems (CMSs).

CMSs [3,31] are an important cornerstone for today's web. In fact, around 59.5% of all websites use one of the various CMS platforms [64] such as WordPress, Joomla, Shopify, and Drupal. From the top CMS platforms, WordPress holds the largest market share (63.5% of all CMS-based websites), fol-

lowed by Shopify (4.5%), Joomla (3.9%), and Drupal (2.6%). A CMS platform aims to provide certain core functionality such as management of users, content, sites, media, templates, and languages. If additional functionality is required, the CMS instance at hand has to be extended. To this end, all major CMS platforms support *software extensions*. Example extensions include web shops, image galleries, or the management of domain-specific data, such as customer information. An extension can be deployed to a running CMS instance without changing the platform itself. This ensures many benefits: leaving the platform unchanged ensures consistency of the system, even after version updates of the platform. Encapsulating feature as extensions supports reuse and free distribution of functionality.

While the plugin mechanisms of WordPress, Shopify, and Drupal are simple, they lack support for complex extensions, such as data management and event triggers. Therefore, plugins for these platforms are often developed as monolithic artefacts. A more sophisticated extension mechanism is offered by Joomla, which provides a variety of *extension types* to facilitate the development of feature-rich extensions. The extension types represent functional requirements. *Components* provide full data management capabilities. *Modules* provide presentation tools for data managed by a component, allowing the development of new extensions using data of existing ones, e.g. a module presenting data of a third-party component. Joomla's extension mechanism is based on an API and naming conventions: For a consistent deployment to the core platform, an extension must conform to a standard file and code structure. To support a clean separation between logic and representation, Joomla components follow a model–view–controller (MVC) pattern on file and code level. If these artefacts are named correctly, most tasks are done by the underlying API. Yet, a mistake during development can break the whole extension.

Developers of extensions face similar challenges during development and maintenance like most web application developers, namely: (i) To ensure compliance with the structure and coding standards, comprehensive technical knowledge is required. A typical procedure during extension development is to *create a clone* of an existing extension and to modify it to satisfy the new requirements. This procedure, however, shows a high susceptibility to errors (e.g. unintended mismatches between class identifiers and file names). (ii) Whenever the underlying platform evolves, existing extensions have to be updated or migrated to adapt to the new platform. The required effort for updating or migrating the extensions increases tremendously if the amount of extensions to migrate grows. One contribution of this paper is an interview study with practitioners from the CMS domain, who confirmed the role of these challenges in practice.

Since extensions in popular CMSs such as Joomla rely on standardized file and code structures, they largely consist of generic and schematically recurring fragments. Therefore, they represent a technical space that may largely benefit from the application of MDD.

In this work, we investigate the practical applicability of MDD during the development of CMS extensions. From our experiences of developing extensions for the Joomla CMS for over 10 years, we introduce three development scenarios (Sect. 2) we deem as particularly important: *Developing an independent extension*, *developing a dependent extension*, and *migrating an extension to a new platform version*.

We present the following contributions:

- An **interview study** (Sect. 3) based on semi-structured expert interviews with eight individual practitioners from the Joomla community. We studied the relevance of our scenarios in practice, and the significance of MDD as a solution approach to address the associated challenges.
- An **MDD infrastructure** (Sect. 4) called JooMDD for developing CMS extensions for the Joomla platform. It comprises a DSL with model editors, a code generator, and a model extraction tool. The DSL and respective model editors can be used for CMS extension development in general, whereas the code generator, model extraction, and extension extraction tools are instantiated for the Joomla CMS. We discuss how we extended an earlier version of JooMDD to precisely address the requirements arising from our scenarios.
- A **controlled experiment** (Sect. 5) conducted with 14 developers on comparing conventional extension development with MDD, focusing on the first two scenarios. The experiment follows a within-groups design and was conducted based on JooMDD. All developers had experience with Joomla extension development, but little knowledge of MDD.
- A **field experiment** (Sect. 6) conducted with four experienced developers from the Joomla community. To gain insights about the usefulness, acceptance, and open challenges of an MDD approach, we asked the developers to use JooMDD during representative tasks covering all three development scenarios.
- Three **case studies** (Sect. 7) showing the application of JooMDD in real-world settings. They represent heterogeneous extensions that are actively used in administrative, software development, and teaching activities.

This paper extends our earlier conference paper [39], where we first present the interview study and the two experiments (corresponding to Sects. 3, 5, and 6 of this paper). Our present paper includes three main extensions: (i) in Sect. 4, a detailed description of JooMDD, which extends the high-level overview of an earlier version given in [41], and a discussion of how we extended it to address all requirements arising from the scenarios; (ii) in Sect. 5, additional results

from the controlled experiment addressing the effect of MDD on the quality of the developed extensions as well as a more rigorous statistical analysis of our data; and (iii) in Sect. 7, the three case studies, showing the practical application of Joomla extensions developed.

We share lessons learned (Sect. 8) and discuss threats to validity (Sect. 9). In Sects. 10 and 11, we discuss related work and conclude this work.

As we discuss in Sect. 10, to our knowledge, JooMDD is the only tool with full support for all scenarios.

2 Development scenarios

From our experience in developing CMS extensions, we identify three frequently occurring development scenarios: the development of independent and dependent extensions and the migration of extensions to new CMS platform versions.

Scenario 1: Development of Independent Extensions This scenario addresses the development of independent extensions to be used in a running CMS instance. Independent extensions are particularly desirable during evolution: If a developer changes the extension, no side effects due to dependencies occur. Yet, it is fundamental to comply with the development guidelines to ensure a correct interplay between the extension and underlying platform. Even subtle errors can lead to crashes that are not discovered until runtime.

Examples for independent extensions are WordPress plugins, Joomla components, or Drupal modules which have no dependencies to other extensions. During evolution of other installed extensions, the extension should not show any side effects. The only dependency of such extensions is the use of the CMS core API. So, a CMS instance can be augmented by, for example, web shop functionality which can be installed and used independently.

The scenario occurs in two variants: First, the initial development of an extension and, second, its iterative improvement (evolution). Both are addressed in this scenario, with the initial development as the first iteration (see Fig. 1).

Scenario 2: Development of Dependent Extensions This scenario involves the development of extensions that depend on other extensions, by using some of their artefacts—a common practice to prevent multiple implementations of the same functionality.

We discuss examples for such extensions in Joomla and WordPress. In Joomla, relevant components are those that reuse data access objects (DAO) or view templates from other components, and modules which use the database of existing components. The latter is commonly implemented to allow the flexible representation of component data without the need of extending the existing component. For example, one may add a shopping cart module for an existing web shop

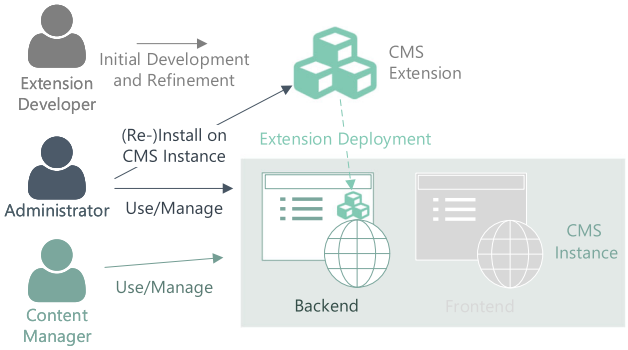


Fig. 1 Development of independent extensions

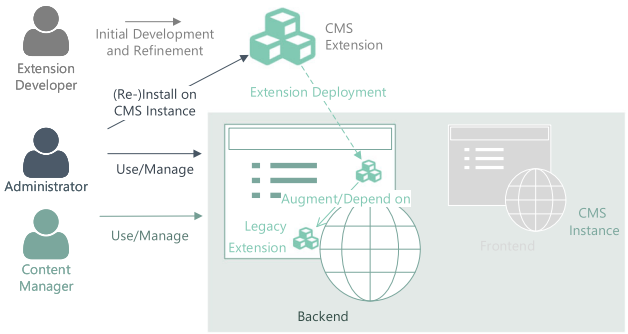


Fig. 2 Development of dependent extensions

component. In WordPress, plugins may refer to other plugins. This allows developers to augment existing extensions (e.g. third-party extensions) without changing their code base (see Fig. 2).

Like Scenario 1, this scenario comprises two subscenarios: initial development and evolution of a dependent extension.

Scenario 3: Migration of a Legacy Extension to a new Platform Version This scenario addresses the migration of a legacy extension to a new version of the underlying CMS platform. Particularly, major version changes are characterized by excessive changes of the platform which usually break existing extensions. So, every new platform version forces extension developers to migrate their legacy extensions to the new API to ensure their operability within updated CMS instances.

In the last decade, the popular CMSs like WordPress, Joomla, and Drupal were subject to several platform improvements in form of new major version releases. Most of them included excessive API changes which had to be considered during extension migration, for example, from Joomla version 1.5 to 2.5.

As experience has shown, missing documentation and required effort often led to dying extensions since developers were not able to migrate their software in a reasonable amount of time. In this case, administrators have to replace the extension which, in turn, is associated with additional effort (see Fig. 3). If no alternative extension exists, admin-

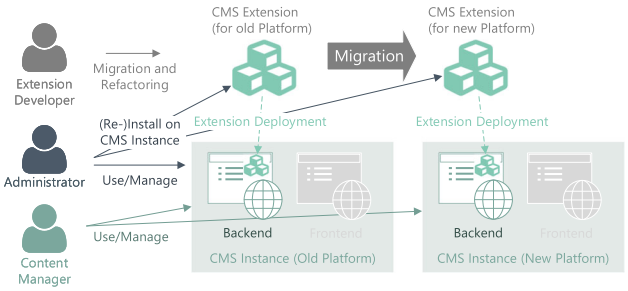


Fig. 3 Migration of an extension to a new platform version

Fig. 4 Standard view with CRUD functionality within CMS extensions

istrators are often forced to keep their CMS instance running with older platform versions until the required extensions are operable on it.

3 Semi-structured expert interviews

To validate the relevance of the previously defined three development scenarios (Sect. 2), we conducted a set of semi-structured expert interviews with 8 industrial practitioners from the Joomla CMS community in 2018. With the interviews we address the following research questions:

- RQ1: How relevant are our scenarios to industrial practitioners from the CMS domain?
- RQ2: Which challenges faced by industrial practitioners from the CMS domain are of a kind where an MDD approach could help?

RQ1 seeks to explore the industrial relevance of the scenarios that we faced in our own experiences. RQ2 is an exploratory question that seeks possible tasks where MDD can be beneficial. The answer to RQ2 yields a hypothesis which we later aim to validate with our experiments.

3.1 Set-up

To obtain insights into the experiences of the Joomla developers, we design an interview guide (provided as part of our online Appendix [42]). The interview guide contained questions from eight question categories:

- *Years of development experience in Joomla (Q1)*
- *Experience in different Joomla versions (Q2)*
- *Experience with different extension types (Q3)*
- *Amount of use of standard CRUD views (Q4)*
- *Amount of extensions developed (Q5)*
- *Applied procedure to implement extensions (Q6)*
- *Amount of extensions migrated (Q7)*
- *Used migration approach (Q8)*

The first questions (Q1, Q2, Q3), were asked to determine the experience of the interviewees. Q3, Q5, and Q7 aimed to determine the relevance of scenarios 1–3 (RQ1). Q4, Q6 and Q8 aimed to study the potential for a successful application of an MDD approach (RQ2).

Q4 is based on our experience that most Joomla components comprise a large amount of standard views with CRUD functionality. CRUD (Create, Read, Update, Delete) are the most basic operations that can be applied to an entity. As shown in Fig. 4, such views comprise a list to present the entities, a toolbar of buttons to provide CRUD functionality, and a detail page to create or edit an entity.

Questions Q5 and Q6 allow obtaining information about the extension development procedure of the interviewees and potential pitfalls during the process. To this end, we asked questions in the following form: *"What is your procedure to implement a new component?"*, *"What is your procedure to augment a component?"*, and *"What kind of extension did you develop?"*.

Q7 and Q8 aim to obtain information about currently used approaches to migrate an extension when the underlying platform changes. The following extract of the interview guide shows the approximate questions: *"Did you ever try to use parts of a third party component in your own extension?"*, *"Did you migrate an extension from one Joomla version to another?"*, and *"Did you ever experience that the Joomla conventions lead to errors in your extensions?"*. The answers help to identify the relevance of the scenarios and possible problems faced during extension development, which concerns RQ1 and RQ2.

In a prestudy to provide further context, specifically for Q4, we investigated the amount of views in publicly available projects, based on the official Joomla extension directory [34], to which we applied a structured process to obtain unique Joomla extensions with components and available source code. We adjusted the filter criteria to only search for extensions that are *components* for *Joomla 3* and *free to download*, which resulted in 751 extensions. Using a crawler we downloaded 92 associated extensions (other download links lead to individual download pages which we could not handle automatically). We added the top 50 extensions, ordered by *overall user rating*, and removed duplicates, mislabelled extensions, and extensions with installation prob-

Table 1 Experience of the interviewees

Interviewees	Years (Q1)	Versions (Q2)	Types (Q3)	Amount of CRUD (Q4)
I1	6	2.5–3.8	M	n.a.
I2	5	2.5–3.8	C,M,P	*"few standard views."*
I3	13	1.0–3.8	P	*"I try to use the Joomla API wherever I can."*
I4	13	1.0–3.8	C,M,P,L	90%
I5	13	1.0–3.8	C,M,P	100%
I6	13	1.0–3.8	M,P	80–90%
I7	4	3.0–3.8	C,M	60–65%
I8	10	1.5–3.8	C,M,P,L	100%

Abbreviations: Component (C), Module (M), Plugin (P), Library (L)

Table 2 Development procedure and migration approach

Interviewees	Development		Migration	
	# extensions (Q5)	Procedure (Q6)	# extensions (Q7)	Approach (Q8)
I1	1	Clone-and-Own	2	n.a.
I2	3	Boilerplate	–	–
I3	2	Clone-and-Own	2	–
I4	10–15	Boilerplate	10–15	*"Rewrite and fix errors until it works."*
I5	100	Clone-and-Own	15–20	*"Rewrite most of the extensions to get rid of old stuff."*$_T$
I6	20	Clone-and-Own	n.a.	*"Iterative until it works."*$_T$
I7	2	Clone-and-Own	–	*"Read changelog then iterative until it worked again."*$_T$
I8	40	Clone-and-Own	15	*"Read a migration guide (if it exists) and then fix errors until it works."*$_T$

lems. In total, we considered 50 extensions with 592 views. We found that 212 views were standard list views, while 191 were standard detail views. In short, 68.07% of the inspected views are standard views with CRUD functionality. To gain further insight, in Q4, we asked the interviewees for the amount of such standard views in their extensions.

All interviews were recorded and transcribed; we provide all transcripts in our online Appendix [42]. In average an interview took 20 minutes which led to a total time of around 3 hours of conversations, transcribed to 11050 words. When we quote from the transcripts, these are given in italicized and quotation marks. Citations are taken verbatim from the transcripts. In some cases, we translated citations from German to English; these are marked with a capitalized T.

3.2 Scenario evaluation

Tables 1 and 2 summarize the interviewees' answers. The names of the interviewees were anonymized and abbreviated to I1-8. Each cell contains the interviewee's answer mapped to the corresponding question. We discuss our three development scenarios along the answers, citing individual statements from the interviews where appropriate.

Scenario 1 (Development of Independent Extensions) The interviewees stated that it would take 2 to 8 hours to have an independent installable component with no business logic following their usual (mostly clone-and-own) procedure: *"it takes us, I think, two hours to set up everything. So, two hours for a component."* Another interviewee said: *"it's between four to eight hours and that of course is still quite a bit more work."* and *"in an hour I can make a frontend view and backend view, but that's a very basic."* All interviewees typically follow either a clone-and-own procedure or use a boilerplate generator to develop a new extension. Relevant statements include: *"Copy&Paste and adapt what is necessary"*$_T$, *"We also have now our own boilerplate we use just to create a component"*, and *"If I need to start a new component I now would start with the boilerplate"*.

In accordance to our findings concerning the amount of standard views with CRUD functionality, they stated: *"I think that ninety percent at least are standard list views."*, which we address within this scenario. Other interviewees stated

around the same amount. Generally, the percentage of standard views was given as 60 up to 100 percent.

Scenario 2 (Development of Dependent Extensions) The interviewees augment third-party and core components with own modules: *"I have done that. I'm still doing it. The example here's in my extension I'm exporting prices from a webshop and the whole logic of price calculation - I don't want to recover that - so I'm using the model from the third-party component that has the logic in it."* Another typical scenario is the augmentation of an existing component by a new view. Even though one of the interviewee had no experience in this scenario, he still sees the relevance of it: *"it would be a nice case, but not one I use. But it's a use case!"*

Scenario 3: Migration of a Legacy Extension to a new Platform Version Depending on the number and complexity of their extensions, migrating legacy extensions takes our interviewees at least a couple of days, mostly months: *"migrating to the next version took maybe a couple days."*, *"You should take a year. I think so if you want to do it well. . . .The daily business goes on. Maybe without any interruptions, it still will take a couple of months."* and *"It took me 6 months to migrate 10 extensions."*. An interesting finding is that no interviewee uses a tool for migrating extensions (Q8). Their approach is mostly *"fix errors until it works."*. I5 even rewrites his extensions completely. Considering the amount and types of his extensions he put high effort into the migration even though all his views are standard views with CRUD functionality. This statement was confirmed by almost every other interviewee for their extensions. This shows that the interviewees have to put tremendous effort into the migration.

3.3 Interpretation

As shown in Table 1 and 2, we interviewed industrial practitioners with many years of experience (Q1 and Q2) in developing and migrating all kinds of Joomla extensions (Q3, Q5 and Q7). Addressing RQ1, the relevance of our development scenarios presented in Sect. 2 is confirmed by the industrial practitioners' statements. During the last decade the practitioners had to face the same recurring challenges like implementing requirements with a high amount of redundant code. Additionally, they had to go through major platform changes with the corresponding migrations of their extensions. In both cases, extension maintenance and evolution required large effort. We conclude the relevance of our scenarios is high. Considering RQ2, in line with our own investigation, the amount of standard views is so high that most of the migration steps can be performed automatically. This is also true for the case of developing new extensions. Although some developers use boilerplate code to create new extensions, most of them use the clone-and-own approach, which takes them at least hours to have an installable extension. These issues can be directly addressed by MDD, since decreasing development effort for such scenarios is an acknowledged strength of MDD.

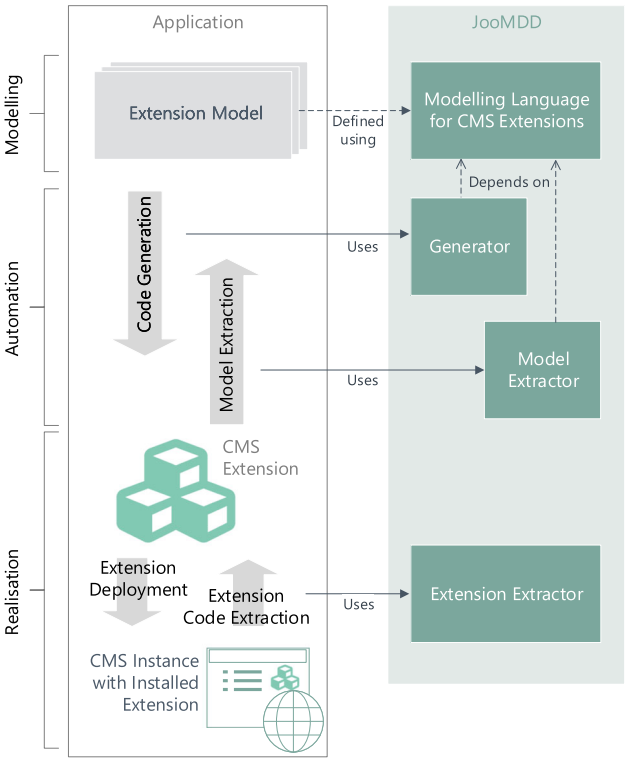


Fig. 5 JooMDD facilities as MDD infrastructure

In conclusion, we hypothesize that the practitioners can significantly benefit from applying MDD during the development and migration of CMS extensions.

4 JooMDD: A model-driven infrastructure for developing Joomla extensions

Our interview study shows that there is potential for using MDD to develop and migrate CMS extensions for saving time and improving quality. In this paper, we investigate this hypothesis with our MDD infrastructure JooMDD [40,41]. In what follows, first, we give an overview of JooMDD. Second, we present the core tool set in more detail. Third, we introduce two new extensions that we developed to provide full support for all three scenarios. Based on the extensions, to our knowledge, JooMDD is the only available MDD infrastructure addressing all three described scenarios (see Sect. 10).

4.1 Overview

JooMDD supports Joomla extension developers with a set of MDD tools: a text-based DSL with editors for modelling extensions, a code generator for generating implementations,

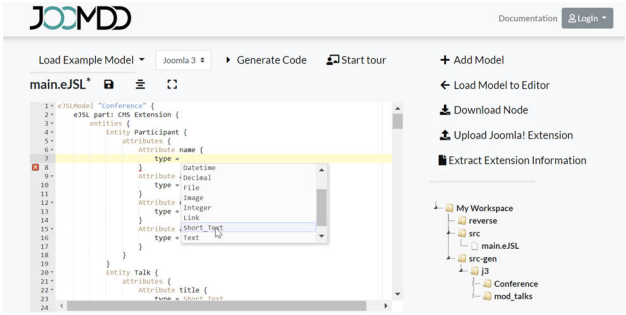


Fig. 7 Web IDE comprising the eJSL model editor

a package extractor for deployed extensions, and a model extractor for extracting models from legacy code. Figure 5 shows the provided artefacts of JooMDD in the context of MDD. The facilities can be used for applying forward and reverse engineering steps in a model-driven manner. This includes the extraction of deployed extensions, model extraction, code generation, and extension evolution.

The JooMDD project and usage instructions are publicly available: https://github.com/thm-mni-ii/JooMDD.

The infrastructure can be deployed to the most commonly used development environments in the CMS domain, that is, *IntelliJ IDEA*, *PhpStorm*, and *Eclipse*. JooMDD's editor plugin is customized for integration with each of these environments. The plugin provides an Xtext-based textual editor with syntax highlighting, error messages, dependency checks, and auto completion support for keywords and references between model elements. In addition, a platform-independent web IDE, depicted in Fig. 7, comprising the whole tool set is publicly available at https://tinyurl.com/joomdd-web. The web IDE allows developers to use JooMDD's full functionality without installing it locally.

To generalize JooMDD to other CMSs than Joomla requires the implementation of platform-specific generator templates for code generation. Furthermore, the implementation of platform-specific model discoverers can extend JooMDD for model extraction using extensions from other CMSs.

4.2 Core tool set

DSL for CMS extensions The *text-based DSL* consists of three parts: a part to model the data management of an extension (*entities*), a part for the definition of a page flow and interaction of extension views (*pages*), and a part for the description of an extension structure (*extensions*). The language can be used to describe CMS extensions in general incorporating the overall extension features of popular CMSs like WordPress, Joomla, and Drupal. The language syntax is inspired by the available Xtext examples and documentation that illustrate the intended use of Xtext. Figure 6 shows an excerpt of the mentioned parts for a simple conference component. Figure 6a illustrates the entity part, in which data entities are defined, for instance a participant entity with its attributes. Pages describe the representation of data entities. Figure 6b presents a list view definition for the representation of all existing participants, whereas a detail page is specified for the representation the details of one participant. Links specify an interaction between both views. Besides some meta information, the extensions part contains references to the previously defined pages, and includes them as component views as shown in Fig. 6c. A more detailed language documentation can be found at the abovementioned GitHub page.

Code generator for Joomla extensions The *code generator* is tailored to the two main Joomla versions actively

```
Entity Participant {
    attributes {
        Attribute name {
            type = Short_Text
        }
        Attribute affiliation {
            type = Text
        }
        Attribute nationality {
            type = Text
        }
        Attribute address {
            type = Text
        }
    }
}
```

(a) Entity

```
IndexPage Participants {
    *entities =  Participant
    representationColumns = Participant.name,
                            Participant.address,
                            Participant.affiliation
    links {
        InternalContextLink Details {...}
    }
}
DetailsPage Participant {
    *entities =  Participant
    editFields {
        Participant.name {...},
        Participant.address {...},
        Participant.affiliation {...},
        Participant.nationality {...}
    }
    links {
        InternalLink Index {...}
    }
}
```

(b) Pages

```
Component MyConference {
    Manifest {
        authors {...}
        copyright = "Copyright (C) 2020."
        license = "GNU General Public License"
        version = "1.0.0"
    }
    languages {
        Language de-DE {...}
        Language system de-DE {}
        Language en-GB {}
        Language system en-GB {}
    }
    sections {
        FrontendSection {
            *Pages {
                *Page : Participants
                *Page : Participant
            }
        }
        BackendSection {...}
    }
}
```

(c) Extension

Fig. 6 DSL instance example: excerpt of a simple conference model

used, Joomla 3 (the current stable release) and Joomla 4 (the upcoming release). It supports extension developers during the development of independent extensions such as components (Scenario 1), and dependent extensions such as modules that use data of existing components (Scenario 2). If an independent extension is to be newly developed, the generator creates the full extension code. When using an existing extension as reference within a new extension, the relationship between the old and new extension can be specified in the model. The generator then generates the new extension, and not the existing one anew. This scenario is presented within Sect. 6.1 (Scenario 2). To support iterative refinements of existing extensions in scenarios 1 and 2, the code generator generates complete installable extension packages, which can be deployed to a running Joomla instance by Joomla's update mechanism.

Model extractor for Joomla 3 extensions The *model extractor* provided by JooMDD supports developers during the reengineering or migration of a legacy extension (scenario 3). It discovers model features from legacy code of existing Joomla 3 extensions, comprising PHP, HTML, JavaScript, and SQL files, creates an extension model based on the provided DSL with the main elements types *entity*, *page*, and *extension*. The extraction tool prerequisites installable extension packages as input. The model extractor is also useful in Scenario 2: Typically, an existing extension must be modelled manually to support model-level references. This step can be automated by using the model extraction tool. The extracted model contains all information needed to model (and generate) new extensions based on the existing one.

4.3 Extensions for scenarios 2 and 3

To support the evaluation of MDD during our addressed scenarios, we developed two extensions of JooMDD: First, we extended the existing code generator to support both considered Joomla versions 3 and 4 (previously only 3), which is especially relevant for platform migration in Scenario 3. Second, to support model extraction in Scenarios 2 and 3, we developed a package extraction tool.

Code generation for Joomla 4 Interoperability between the Joomla framework and deployed components requires that the components implement MVC architecture on code and file base. Even though the same architecture is supported by both considered major platform versions 3 and 4 (Alpha 12 [35]), the file and code structure of components differ a lot. Therefore, we provide respective generator templates for both versions which adhere to the respective file and code architecture (cf. Fig. 8).

Extension Extractor To make an existing extension available for augmenting or migrating it, it needs to be available as self-contained installable extension package. This makes all required files available in one place. Whereas CMSs like

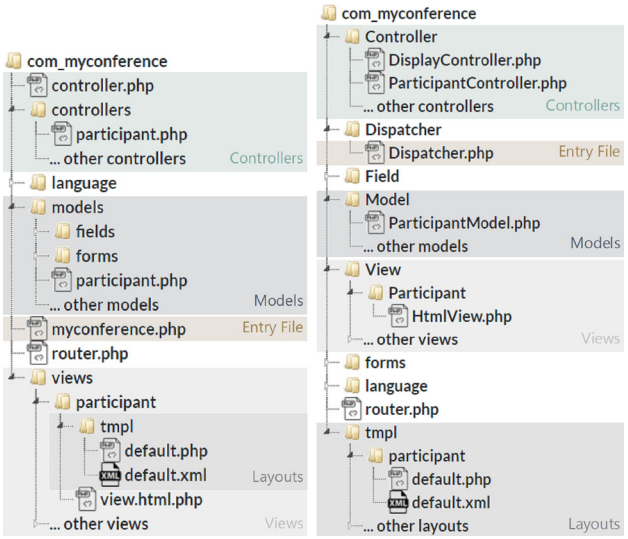


Fig. 8 File structure in Joomla 3 (left) versus Joomla 4 (right)

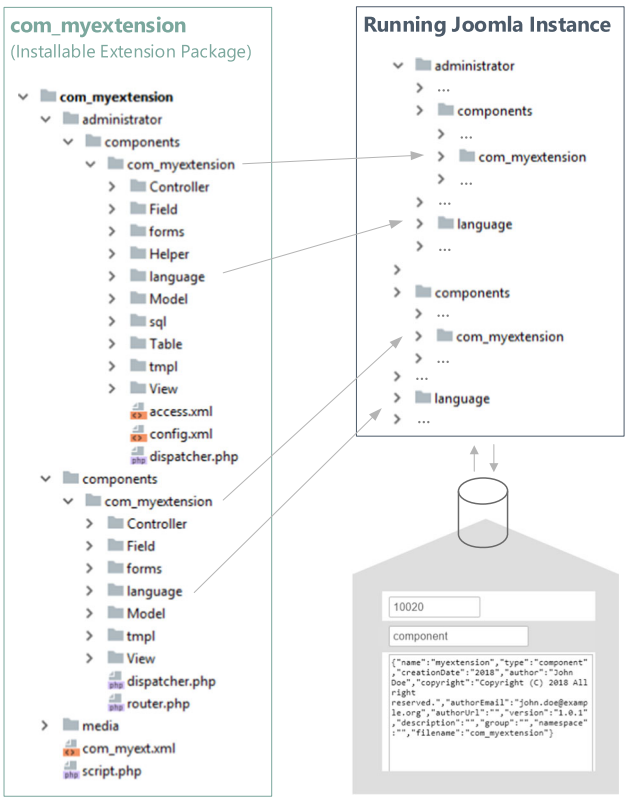


Fig. 9 Separation of a deployed Joomla component

WordPress or Drupal maintain installed extensions within a specific extension folder, Joomla uncompresses extensions and keeps their files separately. The files of Joomla components are subdivided into different folders during the extension installation routine (cf. Fig. 9).

An extraction of extension information from installed Joomla components can therefore be a challenging task.

Often, third-party extensions are typically available as installable extension package, e.g. within extension repositories. However, if this is not the case and an installed extension has to be used during our addressed development scenarios 2 or 3, developers have to extract it from the running CMS instance. The same applies to core extensions, which are typically not available as installable extension packages. Therefore, we provide the ***ExtPorter*** tool, it supports the extraction of files and database schemes of Joomla components which are deployed to a running Joomla instance.

The tool itself is realized as a Joomla component and must be installed to the system where extraction activities have to be performed. This decision has two advantages. First, information of installed extensions can be accessed by using functions of the Joomla framework. So, the implementation effort can be reduced and direct access to the file system or the database is not necessary. Second, developers can use the tool within their well-known environment.

5 Controlled experiment

To study the effect of MDD to productivity and software quality, we conducted a controlled experiment, in which the participants were asked to complete development tasks with conventional programming and MDD. We researched the effect of JooMDD during Scenario 1 and 2 using the example of Joomla CMS to answer the following research questions:

- RQ3: Can MDD increase the productivity of CMS extension development for Scenario 1 and 2?
- RQ4: Can MDD affect the software quality of developed CMS extensions?

5.1 Set-up

By our experimental investigation, we aim to validate the effect of MDD in a systematic, disciplined, controlled and computable manner [37]. To ensure methodological correctness, we follow empirical guidelines as presented in [47], [21], and [66]. Moreover, we incorporate the ideas of the experimental design which is presented by Panach et al. in [36], since it also addresses the investigation of quality and productivity effects of MDD.

These guidelines propose the formulation of hypotheses which can be tested to answer the addressed research questions. Therefore, we formulate null hypotheses (H0) about the effect of our selected treatments, addressing RQ3 (productivity) and RQ4 (quality):

$H0_1$: The developer productivity during Joomla extension development applying MDD is similar to productivity following a traditional development method.

$H0_2$: The software quality of Joomla extensions developed by applying MDD is similar to software quality following a traditional development method.

Falsifying the null hypotheses in our experiments would allow us to conclude that productivity and software quality are indeed improved when applying MDD in the considered scenarios.

Subjects We recruited 14 developers with significant expertise in Joomla extension development. Five participants are experienced Joomla extension developers with a high level (2–10 years) of experience; two of them were also subjects of our interviews presented in Sect. 3. Nine participants were students from an intensive course on Joomla programming; five of them also work as Joomla developers for Joomla extensions used in production in a university website context. We justify the selection of student participants with their comparable performance to professionals when using new software development tools [44].

To ensure sufficient knowledge in extension development for the Joomla CMS, we conducted an external knowledge assessment at the beginning of the experiment, based on a multiple-choice test. We found that all participants have relevant knowledge in extension development. However, two participants showed a knowledge deficit in detailed extension development (MVC interaction). Additionally, we assessed the modelling experience of the participants with the result that all participants had average to high experience (mainly with UML).

Additionally, we asked questions to get an impression of the open-mindedness towards MDD. To this end, using a five-point Likert scale, we asked for an estimation of the need for a tool or method for the transformation of requirement documents into models and implementations. 29% estimated an average need, whereas 50% estimated a high and 21% a very high need for such tools and methods. Moreover, 71% of the participants made use of a generator approach before. This shows that none of the participants was disinclined from the outset.

Design We applied a *within-groups* design (paired design blocked by experimental objects [21,36]). The participants were assigned randomly to two equally sized groups of 7 people. Both groups started with conventional programming followed by a model-driven development session. Each development session had a maximum duration of 3 hours. To encounter a possible learning effect, each participant solved different tasks with both development methods (cf. Table 3). To avoid bias due to one of the tasks being more complicated, we randomized the assignment of tasks to development methodologies between participants.

The tasks, based on two different requirements of similar complexity, were handed out during the development sessions. The requirements were concerned with implementing

Table 3 Study design

Session	Factor	Requirement A	Requirement B
1	Traditional	Group 1	Group 2
2	MDD	Group 2	Group 1

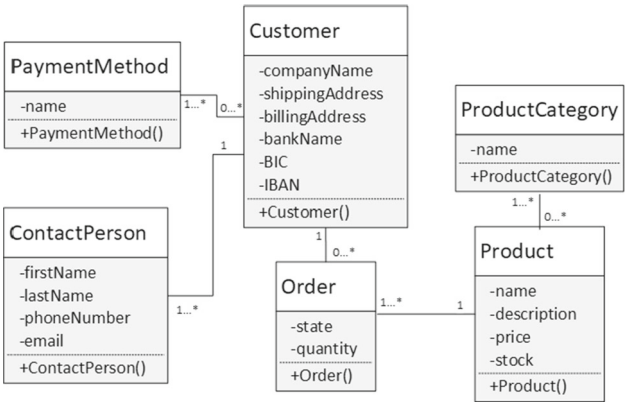


Fig. 10 Entity model of possible solution for CRM requirement

extensions for a university management system (requirement A) and a customer-relationship management system (requirement B). To exemplify the complexity of the requirements, Fig. 10 illustrates a possible solution model with the entities of requirement B. These kinds of requirements are in accordance to the common requirements addressed by extension developers in the community. Based on the popular extensions of the Joomla extension directory and statements of industrial practitioners (see Sect. 3), data management extensions are common in the domain. Group 1 had to implement the first requirement by hand and the second one with MDD, whereas group 2 started with the second requirement followed with the first one with MDD. Both requirements consisted of an independent Joomla component (Scenario 1) with 14 views in total. In particular, 6 list and 6 edit views for the management of each entity in the administration section (backend), as well as 1 list and 1 details view for the end-user (frontend). Each view consist of an representation part as well as a CRUD implementation in the backend section. In addition, the subjects were asked to implement a dependent module (Scenario 2) illustrating data of the implemented component. In total, we had 28 test items for the subsequent evaluation of the results. After each of both sessions, the subjects submitted their solutions.

Instrumentation In the conventional programming session, the participants were free to use a development environment of their choice. For the model-driven development session, they were asked to use the JooMDD web editor for extension development. This allowed us to minimize technical noise regarding the installation of IDE plugins.

Procedure Before the actual development session, a presentation about the experiment and the requirements for a complete solution was given, based on a different example than the one used in the experiments. To ensure anonymous handling of the results and eliminate possible biases, the subjects were identified by a subject ID they had to write on each artefact they filled out during the experiment. The subjects used their own notebooks with their familiar development set-up of choice.

The experiment began with the completion of a demographic questionnaire. Afterwards, the subjects had to complete a knowledge assessment questionnaire (development, Joomla, MDD). Afterwards, the two programming sessions followed. At the start of the first session, a requirements specification was given to all subjects. As previously described, group 1 had to implement Joomla extensions (1 independent component and 1 dependent module) for university management, group 2 for customer relationship management. In a presentation before the second session, an overview of the domain-specific language, the code generator, and the web editor was given. In the second session, the subjects had to implement the second requirement specification using these tools. Since there was no explicit JooMDD training beforehand, the subjects had to follow the infrastructure documentation to fulfil the given requirements. The experimenters gave support considering the infrastructure features. No help for achieving the development results was provided. In each session, the participants had to check the fulfilled requirements in the specification list. After each session the subjects had to answer questions considering the development method as well as the quantity and quality of the development results.

At the end, a closing questionnaire was conducted to get insight to the acceptance of the tools. The complete experiment took 9 hours per participant.

Measurement To address the RQ3 and RQ4, stated in the beginning of this section, each RQ needs to be operationalized in terms of a response variable. These variables are the observed subjects of change based on the effects of the treatments.

RQ3 requires a response variable for measuring the effect of MDD to developer *productivity*. Productivity is often measured as amount of fulfilled requirements to effort ratio [36]. In our experiments, the invested time effort is constant, since all subjects used the maximum session duration of 3 hours during both sessions. Therefore, we can measure productivity in terms of the *amount of fulfilled requirements*. We manually determined the amount of fulfilled requirements, using the requirements description handed out to the participants as a checklist (provided as part of our online Appendix [42]). In contrast to the productivity measurement in [39], we performed a more detailed analysis: instead of just counting the

number of implemented requirement groups, we checked for each individual test case if it was implemented.

RQ4 requires a response variable for the effect of MDD to the *quality* of developed CMS extensions. We used a variable tailored to the quality requirements of the domain: At the time of the experiment, the general coding standard for PHP-based web application development was PSR-2 [38]. PSR-2 aims to *"reduce cognitive friction when scanning code from different authors"* [38], thus contributing to maintainability, one of the quality characteristics in the ISO25010 standard [19]. As part of the task description, we asked our participants to adhere to PSR-2 while implementing the requirements. We measured quality in terms of the *amount of code style violations to LoC ratio* for each requirement. We used *PHP_CodeSniffer* [53] for violation detection and *PHPLOC* [4] for measuring lines of code in each view. Both are standard tools in the PHP community.

Statistical analysis To analyse our results statistically, we tested the two null hypotheses $H0_1$ and $H0_2$ using a standard hypothesis testing approach. First, to select an appropriate test, we first checked whether our measurement data are normally distributed, using the Shapiro–Wilk test [46]. According to this test, the data were not normally distributed, rendering tests that rely on normally distributed data (e.g. t test [21,33]) inapplicable to our data. Instead, we used a nonparametric Mann–Whitney U test [17], which is liberal in its assumptions in the input data. This test yields a *p-value*, which allows to reject the null hypothesis in case that p is smaller than an upfront-defined significance threshold. We used a standard significance threshold of $\alpha = 0.05$. Furthermore, we considered the effect sizes for the comparisons, using Vargha and Delaney's A_{12} score [58], which measures effects on a scale between 0 and 1. Vargha and Delaney suggest to interpret the A_{12} score using the following reference values: 0.56 = small; 0.64 = medium; 0.71 = large [58].

5.2 Results

Below, we present the test results of the submitted solutions based on the observed dependent variables (productivity and quality). To gain insight about the effects of MDD on the development of different requirement groups.

Productivity Table 4 summarizes the results of the controlled experiment based on the amount of fulfilled requirements per development session (3 h each). To gain insight about the productivity gain, we count the number of passed test cases and build the average percentage of requirement fulfilment for each requirement group (A and B) and calculated the respective mean and median productivity coefficient as well as the standard deviation (SD) for both development sessions (baseline, MDD). As Table 4 shows, the overall mean coefficient between the baseline session and the session with MDD varies between 5.9 and 11.7 and the overall median coefficient varies between 14.6 and 18.2.

Table 4 Productivity results: overview

Requirement		Baseline (%)	MDD (%)	Coefficient
A	Mean	7.8	91.3	11.7
	Median	6.3	90.9	14.4
	SD	6.9	4.4	
B	Mean	9.6	56	5.8
	Median	3.6	59.7	16.6
	SD	10.3	34.4	
Overall	Mean	8.7	73.7	8.5
	Median	4.9	89.2	18.2
	SD	8.5	29.9	

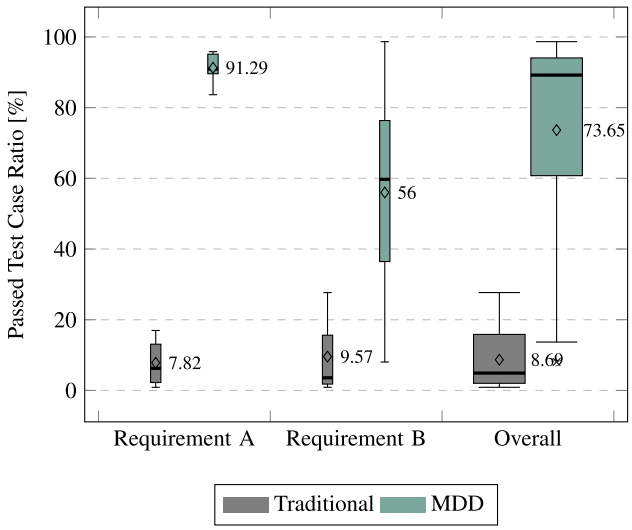


Fig. 11 Passed test case ratio (overview)

Figure 11 illustrates the box plot for the overall productivity result to visualize the productivity differences and variances. The productivity variance of requirement A is quite small for both treatments, whereas MDD of requirement B shows a wide range of productivity amounts. However, this result is based on two outliers (8% and 13%) which were not removed due to the small sample size. Nevertheless, the median of the overall MDD productivity ratio is 30% lower for requirement B. This indicates an effect based on the complexity of the requirement.

Table 5 presents more detailed results for our productivity measurements. For each subject we measured the percentage of passed tests and summarized the results based on relevant requirement groups in each development session. For each requirement group, we calculated the mean and the median. The *Component Structure* row indicates the overall percentage of functional completeness by all participants considering a component that is installable, supports multi-

Table 5 Productivity results: detailed insights

Requirement group	Scenario		Requirement A			Requirement B			Overall		
			Baseline (%)	MDD (%)	Coeff.	Baseline (%)	MDD (%)	Coeff.	Baseline (%)	MDD (%)	Coeff.
Component Structure	1	Mean	53.6	100	1.9	67.9	71.4	1.1	60.7	85.7	1.4
		Median	50	100	2	75	100	1.3	75	100	1.3
		SD	22.5	0		31.3	48.8		27.2	36.3	
Component Views	1	Mean	5.7	88	15.4	5.61	55.6	9.9	5.7	71.8	12.6
		Median	5.4	90.2	16.7	0	61.3	N/A	2.7	87.7	32.5
		SD	6.4	6.3		7	35.1		6.5	29.5	
Component CRUD	1	Mean	7.1	97.6	13.8	10.1	57.4	5.7	8.6	77.5	9
		Median	4.2	100	23.8	0	66.7	N/A	2.1	91.7	43.7
		SD	7.9	6.3		15.6	33		12	30.9	
Module	2	Mean	0	52.4	N/A	0	28.6	N/A	0	40.5	N/A
		Median	0	66.7	N/A	0	0	N/A	0	0	N/A
		SD	0	50.4		0	48.8		0	49.2	

language ability (by language files) and provides update scripts. The *Component Views* row specifies the overall percentage of implemented view features (e.g. table columns, filters, orderings, correct fields and HTML field types). In the *Component CRUD* row the overall percentage of implemented CRUD functionality for each view, including the required buttons and a correct implementation of the associated actions, are collected. The *Module* row provides the overall percentage of fulfilled requirements based on a module that is installable, uses the data of the implemented component, and illustrates the data in a module position. Requirement groups *Component Structure*, *Component Views*, and *Component CRUD* in union represent our Scenario 1 (development of an independent component), whereas requirement group *Module* represents our Scenario 2 (development of a dependent module).

The previously presented results show a pronounced positive effect of MDD during Joomla extension development. To verify these results statistically, the corresponding null hypothesis $H0_1$ has to be rejected. As previously described, we applied a nonparametric Mann–Whitney U test, comparing the mean of two data sets (traditional, MDD).

According to this test, there is a statistically significant difference in the productivity results of both treatments with $U = 11$, $Z = -4.318$, and $p = 1.58 \cdot 10^{-5}$. Considering effect size based on the A_{12} measure yields a score of 0.944. Consequently, we can quantify the effect of using MDD to productivity as *large*.

Quality The results of our quality measurements are presented in Table 6, showing the measured ratios between code style violations and LoC.

To compare the views in session 1 and session 2, we started by analysing the views from session 1 and then we analysed the exact same views from session 2 only even there might be more views. Thus, for requirement A the compared views are 4 and 12 for requirement B. As Table 6 shows, the overall mean coefficient between the baseline session and the session with MDD varies between 0.42 and 0.47 times less violation and the overall median coefficient varies between 0.42 and 0.43 times less violations. This equates to a quality increase of factor 2.4.

Table 6 Quality results: overview

Requirement		Baseline (%)	MDD (%)	Coefficient
A	Mean	1.42	0.66	0.47
	Median	1.6	0.67	0.42
	SD	0.48	0.04	
B	Mean	1.77	0.75	0.42
	Median	1.58	0.67	0.42
	SD	1.58	0.21	
Overall	Mean	1.59	0.71	0.45
	Median	1.59	0.67	0.42
	SD	1.11	0.15	

To use as many artefacts as possible for our quality assessment, we considered all implemented views from both sessions, even if they did not fulfil all requirements. Doing so may potentially provide an advantage to traditional development, since fewer solutions were handed in for the more complex extension types (cf. the results for RQ3). To mitigate bias, we present the quality results itemized by different extension types (which avoids bias especially in the simpler cases). Bias in the complex cases is mitigated by the fact that, in the worst case, the bias is negative against our approach—the reported quality benefit presents a lower bound.

In Fig. 12, the corresponding box plot for the overall result of the violation to LoC ratio is presented. As previ-

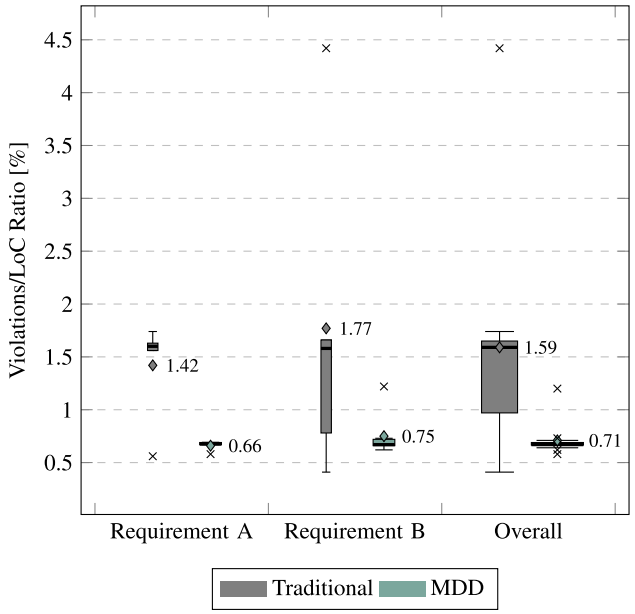


Fig. 12 Code style violations to LoC ratio (overview)

ously described, one outlier (4.42%) led to a higher mean value for the ratio of requirement B in the first session. However, the median of both requirements is quite similar for the respective development method. This indicates that the considered requirements had no explicit effect on the observed code quality.

Table 7 presents more detailed results of our quality measurements. For each requirement group, we calculated the mean and the median code style violations to LoC ratio. The *Component Views: List* row presents the overall percentage for a list view including CRUD functionality. In average a list view consists of around 435 LoC in 5 files. Similar to this, the *Component Views: Edit* row provides the overall percentage for a detail view including the required input fields, buttons, and associated actions. In average a detail view consists of 570 LoC in 5 files. The *Module* row shows the overall percentage for a module. It should be noted that no participant was able to implement a module in the baseline session, so no coefficient can be calculated. Based on our code generator, a module usually consists of 3 files with an average of 255 LoC.

Similar to the productivity result, the previously presented results show a positive effect on the quality of the developed Joomla extension by applying MDD. To evaluate the corresponding null hypothesis $H0_2$ we also applied the non-parametric Mann–Whitney U test. For this test, we only considered the subjects which submitted code and rejected all subjects with 0% productivity amount. This led to 10 subjects for session 1. During the second session, all subjects submitted code for the views which were compared. Having different group sizes is permissible for our test method, which is geared towards unpaired groups.

According to the applied test, there is a statistically significant difference in the quality results of both treatments with $U = 29$, $Z = -2.593$, and $p = 0.01$ which is below α. Based on the A_{12} score of 0.793, we can quantify the effect of using MDD to the quality measurement as large.

5.3 Interpretation

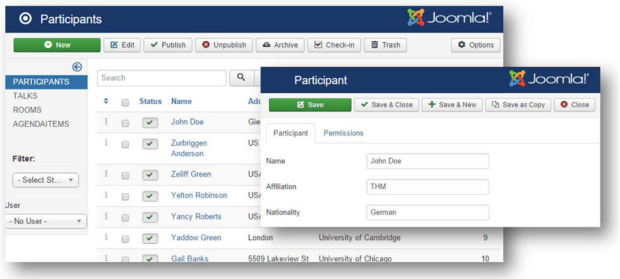
With the results of the experiment, we can answer our research questions. Regarding RQ3 (productivity), even the lowest measured mean coefficient is higher than 5, which shows that the subjects were significantly more productive with MDD during the implementation of each requirement. Regarding RQ4 (quality), the subjects improved the code quality by reducing the amount of code style violations in average at least by the factor 0.42. The latter result is especially remarkable due to the initial advantage for traditional development, for which a larger portion of participants did not hand in any implementations for the more complicated extension types.

Table 7 Quality results: detailed insights

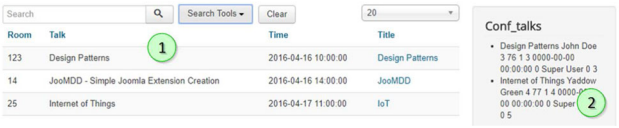
Requirement group	Scen.		Requirement A			Requirement B			Overall		
			Baseline (%)	MDD (%)	Coeff.	Baseline (%)	MDD (%)	Coeff.	Baseline (%)	MDD (%)	Coeff.
Component Views: List	1	Mean	1.55	0.60	0.39	1.46	0.63	0.43	1.51	0.61	0.4
		Median	1.83	0.54	0.30	1.43	0.54	0.38	1.63	0.54	0.33
		SD	0.68	0.12		0.74	0.19		0.67	0.15	
Component Views: Edit	1	Mean	1.32	0.74	0.56	2.84	0.87	0.31	2.08	0.81	0.39
		Median	1.27	0.81	0.64	1.23	0.80	0.65	1.25	0.81	0.65
		SD	0.46	0.12		4.05	0.26		2.83	0.21	
Module	2	Mean	0	0.38	N/A	0	0.40	N/A	0	0.38	N/A
		Median	0	0.38	N/A	0	0.40	N/A	0	0.39	N/A
		SD	0	0.02		0	0		0	0.02	

Both results correspond to the interview statements and our previous research, according to which a large amount of extensions consists of generic code for standard views with CRUD functionality. By applying MDD, these extension parts can be developed faster with better quality. This supports our hypothesis that MDD can substantially enhance the software quality of CMS extensions and increase the developer productivity. The same applies to the development of dependent extensions, whereas the significance of the module requirement should be interpreted with caution.

The experiment design did not allow to conduct a separate module development session. Therefore, only one subject decided to implement the module in the first development session but with no result. The others focused on component development within the sessions time slot. Due to the fact that all subjects were faster during the second session, more of them were able to develop the required module.



(a) List and Details View within a Joomla 3 Instance (Backend)



(b) Conference Management Component (1) and New Module (2) in the Frontend

Fig. 13 Standard views with CRUD functionality

6 Field experiment

In the controlled experiment, due to the high effort for understanding and implementing requirements with two different development methodologies, we focused on the first two scenarios and did not address Scenario 3 (migration). To complement the results with more qualitative insights regarding the usefulness, acceptance, and open challenges of MDD in all three scenarios, we conducted a field experiment with four extension developers of the Joomla community who were also subjects of our conducted interview (see Sect. 3) but did not attend the controlled experiment (see Sect. 5). All developers had a high level of experience (5–13 years), leading to a good knowledge of the processes and problems during extension development and migration.

6.1 Set-up

After an introduction to our MDD tool JooMDD, we observed the developers during the three development scenarios within a total time of 6 hours. To obtain feedback, we subsequently conducted interviews with the participants addressing the MDD approach during the scenarios. To minimize technical noise, the scenarios were applied by using the JooMDD web IDE since it integrates all infrastructure components homogeneously. Additionally, we provided a Joomla installation of the latest available version (3.8) at that time to ensure equal conditions for all participants.

In this section we describe the set-up for the field experiment. For each scenario we define the requirements and the procedure.

Development of an Independent Component (Scenario 1) In Joomla, components are the most commonly developed type of independent extensions. Therefore, we set the task to develop a conference management component as an extension to the Joomla core. To stipulate the requirements, we specified a class model (cf. [40]) for the management of a conference. In the first scenario, the goal was to develop a component for the management of conference data by standard views with CRUD functionality. Specifically, each entity should be displayable in a standard list and details view, such as those shown in Fig. 13a.

The figure shows these views from the perspective of a Joomla administrator who can make the same views visible to site visitors using a menu entry. The resulting component must at least consist of 4 list views and 4 edit views for the management in the backend and 8 views for the frontend representation of the entities. For every view the respective MVC and CRUD code has to be generated as well. Our reference extension model for this scenario has a total of 230 LoC. This includes 4 data entities and 8 different pages which are used for both the frontend and backend. The generated component, including 16 views, consists of 17k LoC.

We started the first part of the development session with the developers by introducing JooMDD and the JooMDD web IDE. Subsequently, we introduced the requirements for the conference component and proposed a possible development procedure using JooMDD. This procedure comprises the use of an example model in the web IDE and to change it to the required conference structure. In the next step the developers had to generate a component based on their specified model. As part of the introduction we explained the structure of the generated code and how the generated component should look like if the code generation worked properly. Provided that a valid model is used as input, the generator creates a full installable conference component. So, no individual code had to be added. As a next step, the developers had

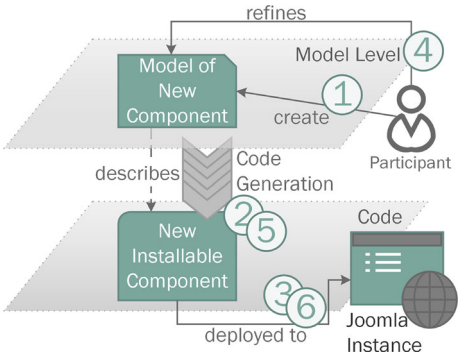


Fig. 14 Development of an independent component (scenario 1)

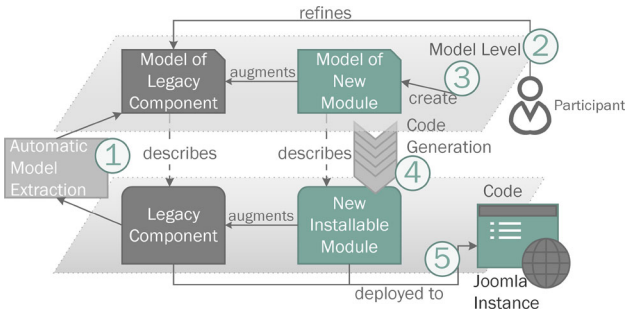


Fig. 15 Development of a dependent module (scenario 2)

to install the component to a Joomla-based web site, which we provided. The developers then had to check if the extension was installed properly and if it worked as homogeneous part of the web site. To this end, they had to create some conference data and try the common CRUD functionality. In addition, they had to create frontend menu entries, to check if the frontend representation works as well.

As next step, the developers had to refine their existing model. They had to add a new data entity and pages to display and manage this entity. After the refinement of the model, the developers had to generate the component anew and reinstall it to the web site. After that, the developers had to check again if the extension works properly. If everything was done correctly, the existing conference data should be still available in the system. The whole process of the first use case is also illustrated in Fig. 14.

Development of a Dependent Module (Scenario 2) In the second part of the experiment, the task was to add a new module to the existing conference component using its DAO, to provide a new representation of the conference talks within a Joomla site which has the conference component installed. Once installed, the module should work together with the already installed conference component by using the component's MVC model for the data access, thus allowing to show a presentation of the obtained data—in our case conference talks which are managed by the component. Figure 13b illustrates a Joomla instance which already has the conference component installed (1) and the new module which uses its data (2) for a different representation. Therefore, an existing extension package of a conference management component is required in this scenario. The participants could use the already downloaded extension package from Scenario 1.

As next step, they had to upload the extension package to the JooMDD web IDE and use the JooMDD model extractor to extract a domain model from the conference component package. We decided for this component to make sure that the input extension matches the Joomla standard file and code schemes to ensure that the extracted models are as complete as possible. If the resulting model contains some validation errors (e.g. illegal identifiers), the participants had to refactor these model elements.

As a further step of this scenario, the participants had to augment the model by new elements to define a new Joomla module with references to the extracted component-specific model elements. Then, using the new model as input, the participants had to use the code generator of the web IDE to create an installable extension package of the new module.

To complete this scenario, the developers were asked to install the module to our provided Joomla installation and, if it has been installed properly, create a module instance, which has to be placed on the frontend section of the website. If everything worked properly, the module had to illustrate the data of the already installed component similar to Fig. 13b. See Fig. 15 for an overview of the procedure.

Migration of a Legacy Component from Joomla 3 to 4 (Scenario 3) For the third scenario, we required the code migration of a component from Joomla platform version 3 to 4. The release of the stable Joomla 4 version is planned for late 2020. Even though the new major release of Joomla requires a completely new extension structure (cf. Fig. 8), the migrated component should include the same features as for the old Joomla version. So, the whole extension structure of an existing Joomla 3 component has to be migrated to the required Joomla 4 structure. Once installed to a Joomla 4 instance, the component views should also be displayed homogeneously and work properly.

The first steps of the procedure, depicted in Fig. 16, were similar to the ones described for Scenario 2. The participants had to use an installable extension package of a Joomla 3 component, upload it to the web IDE, extract a model, and refactor that model. Again, we decided to use the conference component to ensure a full model extraction.

After the model refinements, the participants had to generate the component by choosing J4 as generator option (part of the web IDE) and download the resulting extension package. During the experiment back in 2018 the code generator did not generate fully operable components but created the correct new file structure with the main code changes for Joomla

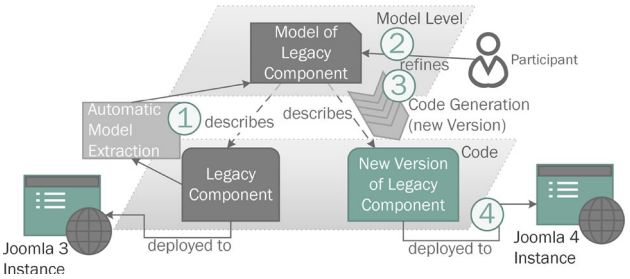


Fig. 16 Migration of a legacy component from Joomla 3 to Joomla 4 (scenario 3)

4. Thus, the participants had to inspect the new components to get an overview of the new component structure.

6.2 Observations

In the field experiment we made the following observations: *Development of an Independent Component (Scenario 1)* Before the procedure started, we observed some reservations against the use of MDD approaches. This also applied to JooMDD. After the first session, however, they were surprised that the tools worked so well. By using the example models as a reference, the developers were able to quickly learn the use of the tools provided by JooMDD. Several editor features were well received, like the auto completion, the error validation, and the syntactical sugar like curly brackets in the DSL, which clarified the structure and model hierarchy. However, one participants had problems with keywords of the DSL. Particularly, the *page* keyword in the model made some problems. The participant expected the keyword *view*, since pages in the model represent views in actual components. Another technical aversion we observed relates the usability of the web IDE. While three of the participants liked the platform-independent editor, the functions of the buttons have not been clear enough. One of the participants disliked the platform-independent solution and preferred to use the available PhpStorm plugin. After 20 minutes all participants had installed their first generated component to the provided Joomla installation. We did not observe different results between participants with more or less technological knowledge.

Development of a Dependent Module (Scenario 2) During the second scenario we observed that 2 of the participants had problems with the resulting model after the model extraction step. Since the model was not completely free of validation errors, the developers found it hard to orient themselves, due to the mass of unfamiliar generated model code. However, with some help, they were able to create and test the new modules in minutes.

Migration of a Legacy Component from Joomla 3 to 4 (Scenario 3) The first observation we made in the third scenario was that, except for one participant, the group had no experience in extension development for the new Joomla 4 version. However, by using JooMDD and following the predefined steps, the group was able to create their first Joomla 4 components, based on the previously generated conference component for Joomla 3. The participants were fascinated by the scenario, since the whole process did not require more than 5 minutes and 4 clicks for the example component. Since no migration steps are defined in any documentation, the participants were grateful to use the generated extension as first reference for their future extension development.

Concluding remark While some of these observations highlight specific issues of JooMDD, they can be used to inform both future versions of JooMDD, and other MDD infrastructures in this domain.

6.3 Developer feedback

During and after the field experiment the developers gave us feedback regarding MDD approaches in general.

The overall observations were positive. One participant stated: *"I think this is really useful to speed up the process and actually when you have to create a standard component which has to do something really easy you can make one really quick. So it's can be a time saver. Yes I'm sure it can."* Another participant observed: *"If I now look at JooMDD what we did today, if I add a new view in the model file, I would still copy&paste a view from there because the structures mostly going to be the same. But there's less to change because there's only one single file where I need to change maybe two names or three names then the rest will be generated. So, it's less error prone then what we're doing now."*

The Joomla experts also pointed out the importance of good naming, especially in the case of name clashes. For instance, in the web IDE, the button for our model extraction tool is labelled "Model Extraction". For MDD infrastructure developer, it is clear that this will extract a model from a given extension. But the Joomla developers first thought about a model in the MVC pattern, which components follow. The same goes for the page part in the DSL. A page in Joomla is called a view, so the developers mentioned, why not name it like that. This is due to the fact that the DSL should also be applicable for other CMSs. It might be a good approach to create a dialect for the DSL to support various CMS-specific language parts. This will enable the domain experts to use their common terms. However, the CMS-specific terms will then be mapped to general terms in the background.

The developers also expressed what they expect from an MDD infrastructure. Concerning the capabilities regarding data management, one participant notes: *"A generator approach has to take care of the database as well, the tables, entities, keys, constraints and also create the database access object."*$_T$ One point often mentioned by the developers was

that they want something like a wizard, which guides them through the extension creation process: *"I was talking about the wizards which can even speed up to the process even more."* One of the developer stated that he would like to have a command line tool to be able to create the model file with predefined model features, so that he does not have to create it by hand: *"With commands, like 'build view x,y,z' and then it asks for the details."*$_T$

Developers want support for the whole development life cycle, beyond the initial steps: *"I expect that the generator is not a generator once and change never option. I expect that it's meant to be part of a continuous developing situation."*

They see the need of version management for the involved textual models [25]: *"If it appears to be a bug in my component after six months and I want to be able to go back to the last one that was generated or the last one before that."* and *"at least have a history."*

Additionally, they expect the generator to be always up to date: *"What I would expect is that if I have my logic inside the code generator it would spit out a component in the new style that I put in a different engine and the engine gives me different code to be doing with Joomla 3, Joomla 4, or whatever platform it's supposed to be running on."* and *"I expect the code generator to always be up to date to the newest Joomla version."*$_T$

Finally, regarding future directions of MDD in the CMS domain, a participant stated: *"The focus should be on what you do should be good and it should be able to hook into your own custom code. . . .You're not restricted to just the generated part and you're not forced to hack into the generated code but to just have enough possibilities to do at your own stuff."* and *"Not try to cover too much. Because then I think you'll be working eighty percent of the time on twenty percent of the functionality."*

7 Case studies

In this section we present three case studies, featuring Joomla extensions that have been developed using JooMDD as the primary development tool. The case studies were conducted at the Institute for Information Sciences[1] in Gießen, Germany, where extensions for the CMS Joomla have been developed for over a decade. These extensions are deployed to several Joomla installations which represent the websites for the Technische Hochschule Mittelhessen (THM) and its departments.

The first two case studies show the use of JooMDD for the initial development of components in administrative and development activities, demonstrating that JooMDD is ready for being used in real-world projects. Whereas in the past most of such extensions have been developed in the conventional way, we used JooMDD for the development of new extensions or the augmentation of existing ones. The third case study is on our use of JooMDD for teaching in web development lectures.

7.1 Precourse management for students

At THM, students can attend a precourse before their regular studies. This allows them to prepare for their study programme. The management of precourse students has been done by a now-outdated external website, which was not part of the university's official website pool. Therefore, the requirement was to incorporate the precourse management into the main Joomla installation of the university.

To this end, JooMDD was used to develop a Joomla extension for managing precourses and their attendees. The main developer in this project was a student, who developed the extension as part of his thesis work. The student had experience in general web development but no experience in Joomla and MDD. However, with an introduction to both systems, by the first and second author of this paper, the student was able to perform the required steps. The extension developed in this project is made up by a component with CRUD views for the management of courses and registered attendees. It works together with the user management of the Joomla core. The component was eventually deployed and used productively at THM's website.

The defined model consisted of 9 data entities and 18 pages, whereas the generated component comprised 30 pages in form of MVC combinations. Since these combinations require the largest amount of code, the high number of pages in the model led to 25k LoC for the whole component with only 347 LoC in the corresponding model. The generated extension was refined by individual functionality (approximately 2.5k LoC) before it was installed and used productively. Thus, more than 90% of the component could be created in a model-driven manner. Only the missing 10% were custom business logic which had to be incorporated by hand. However, this was easily done by adding new functions to the controller and model as well as changing HTML in a view.

After its deployment, the features of the component became incorporated to another Joomla component which is used for the whole resource management like rooms, courses, and schedules of the university within a separate Joomla instance. We included the model created during this case study as a showcase example into JooMDD.

The added value of this case study is to show that an inexperienced user in both the domain and MDD was able to generate an Joomla extension with just a little help. In addi-

[1] Institute at the computer science department at the Technische Hochschule Mittelhessen, http://thm.de/mni/forschung/institute-gruppen/ii

tion, the used MDD tool was able to generate 90% of the extension and the student was able to hook into the generated code to implement the missing custom logic.

7.2 Joomla extension extractor

In Joomla, the code for a deployed extension is spread over many folders in the Joomla instance at hand. If the original installation package is not available, it is a challenging task to get all the relevant extension information from the installed Joomla component. It becomes necessary to collect all files, folders, and the database schema related to one component. When evolving our code generator, we needed to perform this task repeatedly.

To avoid manual effort during this task, a developer who was experienced in Joomla and MDD developed a component called *ExtPorter* for automating it. The goal of ExtPorter is to retrieve an installable extension package from an existing extension deployed to an Joomla instance. Even though some parts contain individual code, the component's main structure and the management views have been fully generated by the JooMDD code generator. Our defined model, consisting of one data entity and two pages, comprised a total amount of 83 LoC. The code of the component, largely consisting of four views (respective frontend and backend views for each page), comprised 5 KLoC.

Beyond the productive use during the enhancement of our generator, ExtPorter is also useful for other developers, as they can create installable packages for legacy components, e.g. to use them as input for our model extraction tool. Therefore, we made ExtPorter publicly available as part of the JooMDD infrastructure (see the description in Sect. 4). ExtPorter was also used by the participants during the field experiment and benefits from their feedback (see Sect. 6).

The added value of this case study is to show that the MDD infrastructure JooMDD has reached a maturity level where it can handle complex requirements that significantly go beyond the CRUD functionalities of basic data management applications.

7.3 Use of JooMDD in teaching

Besides the use of JooMDD in development projects, we use the tools within web development classes to teach the development of PHP-based applications. The students have to (further) develop extensions for Joomla or create new features and patches for the Joomla core. In the first years, the students required most of the time for learning the structure of Joomla extensions and how to implement them. By using JooMDD we were able to reduce the learning effort. By using the DSL and code generator, the students get a better understanding for the required file and code structure, since they can easily change some abstract parts in the model, generate anew and inspect the changes.

The added value of this case study is to show that MDD can be used to teach a complex new technology to learners. At the beginning, students can use the MDD infrastructure to quickly generate artefacts and do changes to them on a high abstraction level. The students can investigate the generated artefacts and observe the changes to get familiar with the new technology. In the further learning process, the students inspect the structure and generated code to get a profound understanding. Using an MDD infrastructure is a straightforward procedure to generate the artefacts anew and observe the changes on different abstraction levels.

8 Lessons learned

In this section we address the lessons learned of our conducted studies among CMS extension developers. Most of them are consistent to the ones presented by Whittle et al. in [65]. This applies especially to the following ones:

Finding the right problem is crucial. All three development scenarios we presented have proven to be significant. The migration scenario, however, is considered as especially pressing and got most attention.

Match tools to people, not the other way around. Developers refused working with Eclipse. For potential reasons see [22]. Instead, they are used to IDEs by JetBrains or web IDEs and await corresponding tool support. In this context, as developers pointed out, MDD has the potential to reduce error susceptibility in contrast to clone-and-own approaches. Additionally, we found three specific sublessons:

1. *Integrate MDD tooling seamlessly into already used tool environments* Developers also asked to consider possibilities for custom code integration into generated code.
2. *Use domain terminology as much as possible* A DSL dialect may better reflect the developer's understanding of a specific domain (such as CMS extension development with Joomla).
3. *Handle models as usual development artefacts* Developers specifically asked for version management support to consider model histories.

More focus on processes, not only on tools Developers ask for wizards supporting them in following predefined processes as they occurred in selected application scenarios.

In addition, we have found further lessons learned which are in accordance to existing MDD adoptions like [5,9,55]:

Apply MDD to develop components instead of whole systems While certain kinds of system components are well suited for MDD others may be not. The developers shall be guided to the promising applications.

MDD for learning new platform versions By automatically migrating a vast part of a CMS extension, developers can learn how a new platform version (here Joomla 4) shall be used. It also becomes easier to add individual code where needed.

MDD for teaching activities Teaching a complex system to inexperienced developers can be overwhelming for them due to technical hurdles (cf. [55]). Using an MDD infrastructure obfuscates technical details and producing high-quality software components encourages students to become acquainted with a new technology. Furthermore, they can do changes on a high abstraction level in the model and re-generate the software artefact on demand. This supports inexperienced developers during the familiarization process with the technology.

MDD for rapid prototyping. Since MDD enables rapid development of software artefacts that cover high amount of domain standards, it can also be used for rapid prototyping within an iterative development process (cf. [5]). The prototype can be successively refined to the final product or even be completely discarded in an early iteration if it does not fulfil the stakeholders' requirements.

9 Threats to validity

Despite the promising results of all presented studies in the previous sections, our work is subject to a number of threats to validity. These are presented by following the classification which is discussed by Wohlin et al. in [66].

Construct Validity We study practical applicability by focusing on three development scenarios that we consider as common in the domain. While the expert interviews confirm the crucial role of these scenarios, the participants also pointed us to an additional scenario that we did not consider yet. In particular, the abstraction of shared functionality into libraries. This threatens construct validity, since we aim to study the development of extensions in general, independent of their type. We aim to study this case in future work.

Even though our interviews confirm that iteratively refining already existing extensions (extension evolution) is a challenge, our interview guide did not contain explicit questions considering the related subscenarios. Since these are part of Scenario 1 and 2, this is a threat to construct validity. Further interview iterations should incorporate such questions to get the interviewees opinion of this subscenario.

Our measurement of productivity relies on a checklist of requirements which was handed out to our participants as a specification to fulfil their tasks. To avoid the risk of imprecise values in the checklists, we have independently checked the solutions and completed the checklists ourselves. Our measure of quality only focuses on adherence to coding guidelines, which is positively correlated with maintainability [15]. However, software quality is a comprehensive construct with further concerns such as performance, usability, and security. Measuring these concerns requires additional experiments.

To address *evaluation apprehension* by the subjects, we only choose volunteer participants from academia and industry during the experiment and only industrial practitioners during the workshop. Additionally, all documents and development results of the experiment were submitted in an anonymized form to reduce biases based on relationship between the experimenters and the subjects.

Internal Validity Based on our decision of using different requirements during the two sessions, we address the threat of *learning effects* during the experiment. Besides affecting the outcome based on the requirements, a learning effect could also affect the choice of the development approach during the traditional development session. This effect is avoided by our study design: traditional in the first session, MDD in the second. Another design would have affected the outcome, as some of the subjects would have used other boilerplate generators, if we had exchanged the treatments between the groups during the sessions. This also concerns *subject motivation* which was guaranteed due to the chosen study design. Otherwise, subjects might find it frustrating to realize the given requirement following a traditional development method after applying MDD. Additionally, our study design avoids the *compensatory rivalry* threat, since all subjects applied the same treatment.

Conclusion Validity The *reliability of the results* relies on the quality of the artefacts provided to the participants, in particular, the tasks and examples. To mitigate the associated threat, we worked with examples and tasks that are already well proven from use in teaching and measured the results with objective metrics. A severe threat to conclusion validity is based on the *statistical significance* due to our small sample size. To address this threat in the experiment, we applied the blocked within groups design. So, we could collect date from all subjects during both sessions, which we were able to directly compare. However, the sample size during the field experiment was too small to make any reliable conclusion.

To avoid that a *heterogeneous knowledge of the subjects* affect the outcome, we applied external knowledge assessments at the beginning of the experiment (multiple-choice tests). So, we assured that all subjects exhibit an adequate background in extension development and modelling required to apply both development methods.

During the measurements, we used the complete data sets without removing the outliers. So we avoided the threat of *data fishing*, which is often applied to mine data to get a specific result, but may lead to incorrect conclusion. Our results, however, may be distorted due to the outliers we included during measurement. However, there is no effect on the conclusion validity, since they lead to an advantage

for traditional development. Removing the outliers would only strengthen our conclusion.

Our studies suffer from the threat of *random irrelevancies*, since we cannot ensure that all subjects spend the complete time with extension development. Again, the significant experiment result would not be different, if we could avoid this threat.

External Validity The five main threats to external validity are: First, we only considered extensions of the Joomla CMS. It yet has to be examined if MDD for the selected requirements and case studies is also suitable for other CMSs, in particular WordPress, the most frequently used system. Since Joomla has the most complex extension mechanism, it is likely that the positive results for Joomla may also generalize to other CMSs like WordPress. However, a new code generator and model extractor is required for the specific needs of each given CMS.

Second, while we involved experts from the domain as participants, the sample size is still relatively small. Our methodology applied to study productivity in scenarios 1 and 2 is qualitative and quantitative.

Third, our experiments rely on specific tooling, namely, our JooMDD infrastructure. We chose JooMDD since it was the only tool available fully supporting our three considered scenarios. However, it would be worthwhile to compare the ability of different tools to support developers during a subset of the scenarios.

Fourth, our DSL has a particular textual syntax, whose design was informed by available examples from the Xtext framework developers, rather than a user study. While variations in the textual syntax could affect the productivity, there are some inherent trade-offs. For example, while our syntax for entity definition could be more concise, we consider the use of keywords such as "Attribute" as beneficial to non-expert users due to their explanatory value.

Fifth, our experimental setting prohibited the use of industrial large-scale example applications, due to the required effort for understanding a large-scale system. We argue that applications of the considered size are still representative for many applications in the field.

10 Related work

Our consideration of related work is twofold: On the one hand, we survey existing MDD approaches being applied in the CMS domain. On the other hand, we relate our field study to other empirical works on MDD in practice.

10.1 MDD in the CMS domain

Several related works propose platform-independent metamodels for the development of specific CMS instances [29,56,63]. Code generation for concrete CMS instances was firstly investigated by Saraiva et al. in [51]. However, none of these works addresses extensibility scenarios of CMSs through standardized extension types taking their interdependencies into account.

The ReLiS framework by Bigendako et al. [6] could be considered a specialized CMS with extension capabilities. It supports researchers during the collaborative conduction of systematic review (SR) projects. The authors present a DSL, a web-based editor, and tool support to automatically build, install, and (re-)configure individual SR projects as extensions to the ReLiS platform. Particularly, the automatic deployment of generated extensions is a promising feature which could be a supportive feature of our proposed infrastructure. However, the presented approach deals on actual extension instances in a running application, whereas our approach addresses extension packages on a higher abstraction level.

Only the XIS-CMS framework presented in [14] has been applied to model and develop CMS modules addressing the CMS DotNetNuke. As this CMS has a limited extension mechanism like WordPress, JooMDD is the first one providing suitable abstractions and automation facilities for a more sophisticated extension mechanism.

Trias et al. introduce a reengineering method and a reverse engineering tool for the migration of complete CMS-based applications in [57]. So, migrations of an instance from one CMS to another CMS can be realized. Even though this approach could potentially be used improve the model extraction component, it is currently tailored to WordPress, a CMS with limited extensibility features. The usefulness for other CMSs with a more sophisticated extension mechanism, such as Joomla, has yet to be investigated. In [62], Vermolen et al. present an approach for the evolution of data models. This approach provides a well-defined strategy to deal with changes to existing data entities. Incorporating it into our work in future will help us to improve the extensibility during the (partial) augmentation of existing legacy extensions (Scenario 2).

Existing model-driven reverse engineering approaches in the web context are tailored to a specific framework, modelling language, or aspect such as the approaches described in [24] and [45]. The authors of [24] consider automatic model extraction based on ASP.net applications and a transformation to WebML [11,12], whereas the proposed extraction process in [45] addresses the examination of graphical user interfaces to extract model information. These works are not suitable in our specific domain, even though the latter work is relevant for our future work, since we have to retrieve UI information from legacy extensions such as extension view or widget representations (Scenarios 2 and 3).

In Table 8, we collect existing tool support for MDD of Joomla extensions available online. The considered tools can

Table 8 Tool support for scenario 1–3

Tool	Joomla Version	S.1	S.2	S.3
Component Generator [52]	3	✓	×	×
Component Builder [59]	3	✓	×	×
Component Creator [20]	3	✓	✓	×
Component Architect [48]	3	✓	✓	×
JCCreator [1]	3	✓	✓	×
JooMDD [40,41]	3, 4	✓	✓	✓

be used to define extension information in an abstract manner and use it for code generation. However, all of these tools are limited to the development of independent components (Scenario 1) or dependent modules (Scenario 2). Therefore, JooMDD stands out due to its unique migration support. We excluded [49,67] and [7] due to a lack of functionality. They generate a very basic scaffold only.

In [43], the authors examine the CRUD aspect in web applications rigorously and propose an approach for the automatic extension of CRUD operations in IFML [8] models. However, since we decided to keep the DSL for CMSs as abstract as possible, explicit CRUD definitions are not provided on model level. Therefore, in contrast to this work, we presuppose that according code generators implement this feature to implicitly generate CRUD operations based on a representation kind in the instance model. The automatic generation of code for CRUD-intensive web applications has been addressed by various works, mostly in form of scaffolding generators. These generators can be used during early stages in development, but do rarely adopt an MDD approach to support developers throughout the whole development life cycle. Typically, these approaches operate at a low abstraction level and require manual adaptions of generated artefacts.

General MDD approaches for the web domain such as the ones in [8,12,26] can be used to create complete websites in a model-driven way but are not suitable for the use cases we considered in this paper since they do not address CMSs and the model-driven development of their extensions.

10.2 Empirical studies on MDD in practice

There have been several efforts to investigate various aspects of MDD in practice. Practical adoption of MDD has been the focus of various studies [10,27,28,50,60,61,65] that generally focus on the embedded system or mobile development domain. Sousa et al. present the adoption of MDD in an industrial modernization scenario in [50], whereas in [65], Whittle et al. develop a taxonomy of tool-related considerations based on empirical data stemming from industry. This taxonomy distinguishes technical factors (concerning technical aspects of MDD tools) from organizational and social ones (focusing on tool use and application within working processes). This taxonomy was used to analyse interviews from industry, mainly at companies such as Ericsson and Volvo. Although developed for empirical studies in other domains, most of their lessons learned are confirmed by our studies among CMS extension developers as stated in Sect. 8.

Mohagheghi et al. [32] reflect the use of MDD in four cases from companies in different domains (enterprise applications, telecommunication, aerospace crisis management systems and geological systems) based on interview and questionnaire studies, focusing on the practical motivation for using MDD and subjective usability aspects. In [23], the adoption of MDD in the openETCS project (railway domain) is analysed based on practical experience, surveys and interviews. The authors of both studies state that MDD can generally be applied successfully, while methodologies and tools are a main inhibiting factor.

The study in [13] investigates the usability of web applications being developed in a model-driven way. The closest studies are probably presented in [30,36], and [37]. In [30], the authors investigate the effect of model-driven development on the maintainability of web applications in comparison to code-centric development. They conducted an experiment where 27 graduated students had to perform a set of maintainability tasks in two groups. Specifically, they investigated the effectiveness, efficiency, usefulness, and ease of use of the development approaches w.r.t. changeability. As result of that study, the authors found *a perceived loss of control with MDD approaches*, *that model-driven development is slightly more learnable and less complex than code-centric development*, and *that developers are not as satisfied with the MDD approach as expected*. Panach et al. research the impact of MDD on quality, effort, productivity, and developer satisfaction in comparison to conventional development of web applications in [36]. The authors conducted a controlled experiment with 26 students as part of an MDD course. The same applies to the presented study in [37], where the authors compare the performance of conventional development with MDD of web applications by conducting a controlled experiment with 29 senior students. Both studies observed a positive effect on the researched variables during MDD adoption (e.g. 90% reduced development time [37]).

In contrast to these studies, we performed our studies with experienced developers in addition to students. Experienced developers are usually confronted with the development and evolution of much larger projects. Specifically, they need to develop and integrate new software components and to migrate code, tasks that are not covered by these studies.

To the best of our knowledge, there is no other empirical study on the use of MDD in the CMS domain.

11 Conclusion and future work

Using an instance of an open source CMS as dynamic web application, developers can add additional features by implementing installable extensions. However, developing these extensions can be a time-consuming and complex task, even for experienced extension developers. Therefore, we propose the use of MDD during the development as efficient alternative to conventional programming.

In this work, we share the results of a mixed method empirical investigation of applying MDD during CMS extension development. All conducted studies refer to three major development scenarios we identified beforehand: development of both dependent and independent extensions and migration of an extension to a new platform version.

First, we conducted semi-structured expert interviews with extension developers coming directly from the CMS domain. This allowed us to study the representativeness of these scenarios. Second, by conducting a controlled experiment, we compared the conventional extension development method with MDD. During the experiment we focused on the first two scenarios. The results showed a clear gain in productivity and quality when an MDD infrastructure is used for extension development. Third, we conducted a field experiment with the goal of obtaining direct feedback about acceptance, usefulness, and open challenges of the adopted MDD approach. To this end, we asked four experienced extension developers from the Joomla domain to use the MDD infrastructure during all three development scenarios. Fourth, we presented three case studies from the practical use of JooMDD in the context of administrative and development tasks as well as teaching.

Conclusively, we share the lessons learned from our work. Generally, the slogan *focus more on processes and people, not only on tools* from Whittle et al.'s work [65], applies to our domain as well. Since there is still little data about application scenarios where MDD can be applied successfully, we tried to identify three relevant scenarios. W.r.t. these scenarios we can conclude that *MDD-based migration support was particularly welcomed*.

Considering possible directions for future work, we outline two main directions. First, we like to conduct more qualitative studies of CMS development for other CMS platforms, like WordPress, Shopify, or Drupal. Due to the positive findings of our Scenario 3 and the significant difference between Joomla version 3 and 4, we see the potential to use JooMDD as the MDD infrastructure in these studies. Second, our selection of considered scenarios is potentially not exhaustive. Additional scenarios which include more sophisticated tasks have to be identified. In particular, one of our participants pointed out the abstraction of shared functionality into libraries as a viable scenario.

Moreover, from our experience, the following situations may present further application scenarios for model-driven development: First, the augmentation of existing extensions with custom features. New features may be added directly to the existing extensions in form of new views, new or refined database structures, or overwrites. This is a subscenario of Scenario 2 (development of dependent extensions) since the new feature is an augmentation to an existing, deployed extension and usually depends on the original code or data. Second, the reengineering of a legacy extension in the context of quality assurance. Extension developers have to discover the structure of the legacy code to reengineer the extension to the new desired structure. Then, after the reengineering process, the resulting extension can be reinstalled to running CMS instances or provided as new version, e.g. in an extension directory.

To support additional scenarios in a model-driven manner, an extension of the current existing MDD infrastructure JooMDD is required. Otherwise, MDD cannot be researched as viable alternative to conventional extension development in these scenarios.

In this context, a functional refinement of the MDD infrastructure should be considered to handle individual business logic during common development scenarios. Examples for such business logic may be data processing and the integration of custom web services. While the current version of the DSL allows the specification of custom business logic, e.g. as part of page actions (used in Joomla components and modules) or OO-based definitions which can be wrapped by (Joomla) plugins, the current realization of the transformation tools cannot handle more individual business logic.

With the exception of one comment during the field experiment, we generally did not receive negative feedback about the syntax of the text-based DSL. However, the trade-off between conciseness and explanatory value of using more keywords could be addressed in an additional user study in the future.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Alnasser, M.: JCCreator (2019). https://jc-creator.com
2. Baker, P., Loh, S., Weil, F.: Model-driven engineering in a large industrial context—Motorola case study. In: International Conference on Model-Driven Engineering Languages and Systems, pp. 476–491. Springer (2005)
3. Barker, D.: Web Content Management: Systems, Features, and Best Practices. O'Reilly, Beijing and Boston (2016)
4. Bergmann, S.: PHPLOC (2020). https://github.com/sebastianbergmann/phploc
5. Bernardi, M.L., Lucca, G.A.D., Distante, D.: Model-driven fast prototyping of RIAs: from conceptual models to running applications. In: International Conference on Advances in Computing, Communications and Informatics, pp. 250–258 (2014)
6. Bigendako, B., Syriani, E.: Modeling a tool for conducting systematic reviews iteratively. In: International Conference on Model-Driven Engineering and Software Development, pp. 552–559. Scitepress–Science and Technology Publications, Lda, Setubal, PRT (2018)
7. Boilerplate Contributers: Boilerplate files for Joomla! extensions (2019). https://github.com/joomla-extensions/boilerplate
8. Brambilla, M., Fraternali, P.: Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML. Morgan Kaufmann, Waltham, MA (2015)
9. Bunse, C., Gross, H., Peper, C.: Embedded system construction—evaluation of model-driven and component-based development approaches. In: Models in Software Engineering, pp. 66–77. Springer, Berlin, Heidelberg (2009)
10. Burden, H., Heldal, R., Whittle, J.: Comparing and contrasting model-driven engineering at three large companies. In: International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. ACM, New York, NY, USA (2014)
11. Ceri, S.: Designing Data-Intensive Web Applications. Morgan Kaufmann Publishers, Amsterdam and Boston (2010)
12. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing web sites. Comput. Netw. **33**(1–6), 137–157 (2000)
13. Fernandez, A., Abrahão, S., Insfran, E.: Empirical validation of a usability inspection method for model-driven web development. J. Syst. Softw. **86**(1), 161–186 (2013)
14. Filipe, P., Ribeiro, A., Silva, A.R.: XIS-CMS: Towards a model-driven approach for developing platform-independent CMS-specific modules. In: International Conference on Model-Driven Engineering and Software Development, pp. 535–543 (2016)
15. Hegedus, P.: Revealing the effect of coding practices on software maintainability. In: International Conference on Software Maintenance, pp. 578–581. IEEE (2013)
16. Heijstek, W., Chaudron, M.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: Conference on Software Engineering and Advanced Applications, pp. 113–120. IEEE (2009)
17. Hettmansperger, T., McKean, J.: Robust Nonparametric Statistical Methods. CRC Press, Boca Raton (2010)
18. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: International Conference on Software Engineering, pp. 471–480. ACM (2011)
19. ISO/IEC 25010: ISO/IEC 25010:2011, systems and software engineering—systems and software quality requirements and evaluation (square)—system and software quality models (2011)
20. Jensen Technologies SL: Component Creator (2019). https://www.component-creator.com
21. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation, 1st edn. Springer, Berlin (2010)
22. Kahani, N., Bagherzadeh, M., Dingel, J., Cordy, J.R.: The problems with cclipse modeling tools: a topic analysis of eclipse forums. In: International Conference on Model-Driven Engineering Languages and Systems, pp. 227–237. ACM, New York, NY, USA (2016)
23. Karg, S., Raschke, A., Tichy, M., Liebel, G.: Model-driven software engineering in the OpenETCS project: project experiences and lessons learned. In: International Conference on Model-Driven Engineering Languages and Systems, p. 238–248. ACM, New York, NY, USA (2016)
24. Katsimpa, T., Panagis, Y., Sakkopoulos, E., Tzimas, G., Tsakalidis, A.: Application modeling using reverse engineering techniques. In: Symposium on Applied computing, p. 1250. ACM, New York, NY (2006)
25. Kehrer, T., Pietsch, C., Kelter, U., Strüber, D., Vaupel, S.: An adaptable tool environment for high-level differencing of textual models. In: International Workshop on OCL and Textual Modeling, pp. 62–72 (2015)
26. Kraus, A., Knapp, A., Koch, N.: Model-driven generation of web applications in UWE. In: International Workshop on Model-Driven Web Engineering (2007)
27. Krogmann, K., Becker, S.: A case study on model-driven and conventional software development: The Palladio editor. In: Software Engineering 2007—Beiträge zu den Workshops—Fachtagung des GI-Fachbereichs Softwaretechnik, pp. 169–175. Gesellschaft für Informatik e. V., Bonn (2007)
28. Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Softw. Syst. Model. **17**(1), 91–113 (2018)
29. Martínez, S., Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Cabot, J.: Towards an access-control metamodel for web content management systems. In: Sheng, Q.Z., Kjeldskov, J. (eds.) Current Trends in Web Engineering, vol. 8295, pp. 148–155. Springer International Publishing, Cham (2013)
30. Martinez, Y., Cachero, C., Meliá, S.: Empirical study on the maintainability of web applications: model-driven engineering versus code-centric. Empir. Softw. Eng. **19**(6), 1887–1920 (2014)
31. McKeever, S.: Understanding web content management systems: evolution, lifecycle and market. Ind. Manag. Data Syst. **103**(9), 686–692 (2003)
32. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M.A.: An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. Empir. Softw. Eng. **18**(1), 89–116 (2013)
33. Montgomery, D.C.: Design and Analysis of Experiments, 5th edn. Wiley, New York / Chichester (2001)
34. Open Source Matters Inc.: Joomla! Extensions directory (2018). https://extensions.joomla.org/
35. Open Source Matters Inc.: Joomla 4 is on the horizon ... Alpha 12 (2019). https://developer.joomla.org/news/793-joomla-4-is-on-the-horizon-alpha-12.html
36. Panach, J.I., España, S., Dieste, O., Pastor, O., Juristo, N.: In search of evidence for model-driven development claims: an experiment on quality, effort, productivity and satisfaction. Inf. Softw. Technol. **62**, 164–186 (2015)
37. Papotti, P.E., Prado, A.F., Souza, W.L., Cirilo, C.E., Pires, L.F.: A quantitative analysis of model-driven code generation through software e. In: Advanced Information Systems Engineering, pp. 321–337. Springer, Berlin, Heidelberg (2013)
38. PHP Framework Interop Group: PSR-2: Coding style guide (2020). https://www.php-fig.org/psr/psr-2/
39. Priefer, D., Kneisel, P., Rost, W., Strüber, D., Taentzer, G.: Applying MDD in the content management system domain: scenarios and empirical assessment. In: International Conference on Model-Driven Engineering Languages and Systems, pp. 56–66 (2019)

40. Priefer, D., Kneisel, P., Strüber, D.: Iterative model-driven development of software extensions for web content management systems. In: European Conference on Modelling Foundations and Applications, pp. 142–157. Springer International Publishing, Cham (2017)
41. Priefer, D., Kneisel, P., Taentzer, G.: JooMDD: A model-driven development environment for web content management system extensions. In: International Conference on Software Engineering, pp. 633–636. ACM, New York, NY, USA (2016)
42. Priefer, D., Rost, W., Strüber, D., Taentzer, G., Kneisel, P.: Online Appendix: model-driven development in the content management system domain: empirical assessment during common development scenarios (2020). https://figshare.com/articles/journal_contribution/Model-Driven_Development_in_the_Content_Management_System_Domain_Empirical_Assessment_during_Common_Development_Scenarios/12661538
43. Rodriguez-Echeverria, R., Preciado, J., Sierra, J., Conejero Manzano, J.M., Sánchez-Figueroa, F.: AutoCRUD: automatic generation of CRUD specifications in interaction flow modelling language. Sci. Comput. Program. **168**, 165–168 (2018)
44. Salman, I., Misirli, A.T., Juristo, N.: Are students representatives of professionals in software engineering experiments?. In: IEEE International Conference on Software Engineering, vol. 1, pp. 666–676 (2015)
45. Sánchez Ramón, Ó., Sánchez Cuadrado, J., García Molina, J.: Model-driven reverse engineering of legacy graphical user interfaces. Autom. Softw. Eng. **21**(2), 147–186 (2014)
46. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality. Biometrika **52**(3–4), 591–611 (1965)
47. Shull, F., Sjøberg, D.I.K., Singer, J.: Guide to Advanced Empirical Software Engineering. Springer, London (2008)
48. Simply Open Source: Component Architect (2019). https://www.componentarchitect.com
49. Skyline Software: Module-Creator (2019). https://extstore.com/tools/module-creator
50. Sousa, V., Syriani, E., Paquin, M.: Feedback on how MDE tools are used prior to academic collaboration. In: Symposium on Applied Computing, pp. 1190–1197. ACM, New York, NY, USA (2017)
51. Sousa Saraiva, J.: Development of CMS-based web applications with a multi-language model-driven approach. (PhD Thesis), Universidade Técinica de Lisboa, Lisbon, Portugal (2013)
52. Spacedog ApS: Component Generator (2019). https://www.componentgenerator.com/
53. Squiz Labs: PHP_CodeSniffer (2020). https://github.com/squizlabs/PHP_CodeSniffer
54. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Hoboken, NJ, USA (2006)
55. Torres, J., Resendiz, J., Aedo, I., Dodero, J.M.: A model-driven development approach for learning design using the LPCEL editor. J. King Saud Univ. Comput. Inf. Sci. **26**(1), 17–27 (2014)
56. Trias, F.: Building CMS-based web applications using a model-driven approach. In: International Conference on Research Challenges in Information Science, pp. 1–6. IEEE, Piscataway, NJ (2012)
57. Trias, F., de Castro, V., López-Sanz, M., Marcos, E.: RE-CMS: A reverse engineering toolkit for the migration to CMS-based web applications. In: Symposium on Applied Computing, pp. 810–812. ACM, New York, NY, USA (2015)
58. Vargha, A., Delaney, H.D.: A critique and improvement of the CL common language effect size statistics of McGraw and Wong. J. Educ. Behav. Stat. **25**(2), 101–132 (2000)
59. Vast Development Method: Joomla Component Builder (2019). https://www.joomlacomponentbuilder.com/
60. Vaupel, S., Strüber, D., Rieger, F., Taentzer, G.: Agile bottom–up development of domain-specific IDEs for model-driven development. In: International Workshop on Flexible Model Driven Engineering, pp. 12–21 (2015)
61. Vaupel, S., Taentzer, G., Gerlach, R., Guckert, M.: Model-driven development of mobile applications for android and iOS supporting role-based app variability. Softw. Syst. Model. **17**(1), 35–63 (2018)
62. Vermolen, S.D., Wachsmuth, G., Visser, E.: Generating database migrations for evolving web applications. SIGPLAN Not. **47**(3), 83–92 (2011)
63. Vlaanderen, K., Valverde, F., Pastor, O.: Model-driven web engineering in the CMS domain: a preliminary research applying SME. In: Enterprise Information Systems, vol. 19, pp. 226–237. Springer, Berlin, Heidelberg (2009)
64. W3Techs: Usage statistics and market share of content management systems for websites, August 2020 (2020). https://w3techs.com/technologies/overview/content_management
65. Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., Heldal, R.: Industrial adoption of model-driven engineering: Are the tools really the problem? In: International Conference on Model-Driven Engineering Languages and Systems, vol. 8107, pp. 1–17. Springer, Berlin/Heidelberg (2013)
66. Wohlin, C., Runeson, P., Hst, M., Ohlsson, M.C., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer, Berlin (2012)
67. xdsoft: Joomla Module Generator (2019). https://xdsoft.net/joomla-module-generator/

Dennis Priefer is a PhD student at the Philipps University in Marburg, Germany. He works as a lecturer and research assistant at the Institute for Information Sciences at the Technische Hochschule Mittelhessen in Gießen, Germany. His interests in research and teaching are in model-driven engineering, web-based systems, content management systems, and agile software development principles. He is currently finishing his doctorate which focuses specifically on researching the effect of model-driven engineering in the context of web content management systems.

Wolf Rost is a PhD student at the Philipps University in Marburg, Germany. He works as a research assistant at the Institute for Information Sciences at the Technische Hochschule Mittelhessen in Gießen, Germany. His research interests are model-driven engineering and information extraction, which he focuses on in his PhD thesis titled "Mining of DSLs and Generator Templates from Reference Applications".

Daniel Strüber is an assistant professor in the software science group at Radboud University in Nijmegen, the Netherlands. His research interests are in model-driven engineering, AI engineering, variant-rich systems, search-based software engineering and software security. He was awarded his doctoral degree from Philipps University Marburg, Germany, in 2015 and worked as a post-doctoral researcher at University of Koblenz and Landau, Germany, and Gothenburg University, Sweden. He is a coauthor of over 70 papers with six Best Paper Awards. He has been an Organizing Committee member of MISE'19, MODELS'18 and STAF'17, and a Program Committee member of several leading conferences, including FASE, MODELS and SPLC. He is the lead developer of Henshin, a model transformation language used by groups in Germany, Canada, UK, Norway and Luxembourg.

Gabriele Taentzer is professor for Software Engineering at the Philipps-Universität Marburg, Germany. Her current interests in research and teaching are concerned with model-driven software development, especially domain-specific languages and model transformations, and software quality assurance. She considers open problems in software development from both the theoretical and practical viewpoints. She draws a bow from the formal foundation of concepts and methods based on graph transformation to the implementation of supporting tools as Eclipse plug-ins and their applications.

Peter Kneisel is a professor at the Technische Hochschule Mittelhessen, associate dean of the department Mathematics, Natural and Computer Science and head of the Institute of Information Sciences. His main research interests, which he represents mainly in teaching, are in the areas of web-based systems and interdisciplinary studies, mainly in digital humanities, where he founded and is leading the study program "Social Media Systems", combining management, communication sciences and IT. He is coauthor of several papers in the area of model-driven development and its application to web-based systems.