Applying MDD in the Content Management System Domain: Scenarios and Empirical Assessment

Dennis Priefer, Peter Kneisel, Wolf Rost
Institute for Information Sciences
Technische Hochschule Mittelhessen
Gießen, Germany
[dennis.priefer,peter.kneisel,wolf.rost]@mni.thm.de

Daniel Strüber
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden
danstru@chalmers.se

Gabriele Taentzer
Philipps-Universität Marburg
Marburg, Germany
taentzer@informatik.uni-marburg.de

***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 highly schematic 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, and a field experiment. 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 17 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.

***Index Terms*—Model-Driven Development, Content Management Systems, Empirical Assessment**

I. Introduction

Model-driven development (MDD, [1]) has been conceived as a development paradigm that aims to increase developer productivity 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 [3], embedded systems [4], and conventional web applications [5]. A domain that has received little attention so far, despite its large-scale practical importance, are Content Management Systems.

Content Management Systems (CMSs) [6], [7] are an important cornerstone for today's web. In fact, around 55% of all websites use one of the various CMS platforms [8] such as WordPress, Joomla, and Drupal. 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.

While the plugin mechanisms of WordPress 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 artifacts. 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 sophisticated standard file and code structure. Joomla components follow a Model-View-Controller (MVC) pattern on file and code level. If all artefacts (models, views, and controllers) are named correctly, most tasks are done by the underlying API. Yet, a mistake during development can break the whole component.

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 version. 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 model-driven development (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. II) we deem as particularly important: *Developing an independent extension*, *developing a dependent extension*, and *migrating an extension to a new platform version*. Based on these scenarios, we make the following contributions:

- An **interview study** (Sect. III) 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.
- A **controlled experiment** (Sect. V) 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 an existing MDD infrastructure. All developers had experience with Joomla extension development, but little knowledge of MDD.
- A **field experiment** (Sect. VI) 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 a suitable MDD infrastructure during representative tasks based in all three development scenarios.

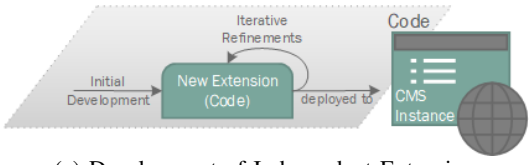
We share lessons learned in Sect. VII and discuss threats to validity in Sect. VIII. We conducted the experiments with *JooMDD* (Sect. IV, [9], [10]), an MDD infrastructure comprising a DSL with model editors, a code generator, and a model extraction tool. As we discuss in Sect. IX, to our knowledge, JooMDD is the only tool with full support for all scenarios. However, our obtained results may generalize to other tools as far as they support our envisioned scenarios.

II. Development Scenarios

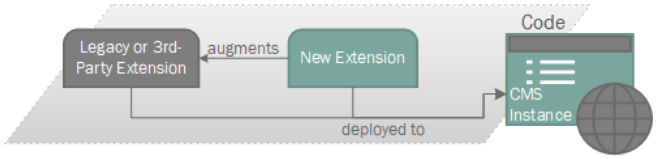
From our experience of developing CMS extensions, we identify three frequently occurring development scenarios: The development of both independent and dependent extensions, and the migration of an extension between two versions of the CMS platform. We now describe these scenarios in detail.

A. Scenario 1: Development of Independent Extensions

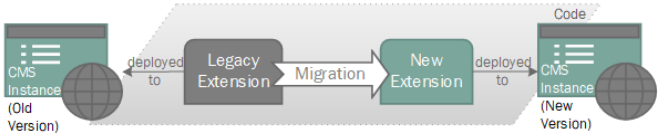
This scenario addresses the development of independent extensions to be used in a running CMS instance. Independent extensions are particularly desirable during evolution: If a developer changes the extension, no side effects due to dependencies occur. However, it is fundamental to comply with the development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



(a) Development of Independent Extensions



(b) Development of Dependent Extensions



(c) Migration of an Extension to a new Platform Version

Fig. 1: Development scenarios.

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

B. 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. In Joomla, components may reuse data access objects (DAO) or view templates from other components, and modules may use the database of existing components. In WordPress, plugins may refer to other plugins. This allows developers to augment existing extensions (e.g. 3rd-party extensions) without changing their code base (see Fig. 1b).

C. 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. Especially major version changes are characterized by tremendous 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. 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. 1c).

TABLE I: Result of the Semi-Structured Expert Interviews.

Cat.	P1	P2	P3	P4	P5	P6	P7	P8
C1 (years)	6	5	13	13	13	13	4	10
C2 (versions)	2.5-3.8	2.5-3.8	1.0-3.8	1.0-3.8	1.0-3.8	1.0-3.8	3.0-3.8	1.5-3.8
C3 (# ext.)	1	3	2	10 - 15	100	20	2	40
C4 (type)	M	C,M,P	P	C,M,P,L	C,M,P	M,P	C,M	C,M,P,L
C5 (# ext.)	2	-	2	10 - 15	15 - 20	n.a.	-	15
C6 (approach)	n.a.	-	-	*"Rewrite and fix errors until it works."*	*"Rewrite most of the extensions to get rid of old stuff."*$_T$	*"Iterative until it works."*$_T$	*"Read the changelog then iterative until it worked again."*$_T$	*"Read a migration guide (if it exists) and then fix errors until it works."*$_T$
C7 (#CRUD)	n.a.	*"...few standard views."*	*"I try to use the Joomla API wherever I can."*	90%	100%	80 - 90%	60 - 65%	100%
C8 (Proc.)	CAO	Boilerplate	CAO	Boilerplate	CAO	CAO	CAO	CAO

Abbreviations: Component (C), Module (M), Plugin (P), Library (L), Clone-and-Own (CAO)

III. Semi-Structured Expert Interview

In order to validate the relevance of the previously defined development scenarios, we conducted a set of semi-structured expert interviews with 8 industrial practitioners from the Joomla community in 2018. Based on the interviews we aimed to address the following research questions:

- RQ1: How relevant are our scenarios to industrial practitioners from the CMS domain?
- RQ2: Which problems faced by industrial practitioners from the CMS domain can be addressed by an MDD approach?

A. Set-up

To obtain insights into the experiences of the Joomla developers, we designed 8 questions based on the following categories: **Years of development experience in Joomla (C1)**, **Knowledge in different Joomla versions (C2)**, **Amount of extensions developed (C3)**, **Extension types (C4)**, **Amount of extensions migrated (C5)**, **Used migration approach (C6)**, **Amount of use of standard CRUD views (C7)**, **Applied procedure to implement extensions (C8)**.

C1-5 were asked as warm-up questions, and to determine the experience of the interviewees. In addition, the answers to these categories will determine the relevance of the scenarios 1-3 (RQ1). C6 aims to obtain information about their currently used approach to migrate an extension when the underlying platform changes. This category will help to identify the relevance of the scenarios and possible problems faced during extension development, which concerns RQ1 and RQ2. C7 is based on our experience that most Joomla components comprise a large amount of standard views with CRUD functionality. As shown in Fig. 2, 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.

In a pre-study to motivate C7, we investigated the amount of views in actual projects, based on the official Joomla extension directory [11], to which we applied a structured process to obtain unique extensions with components and available

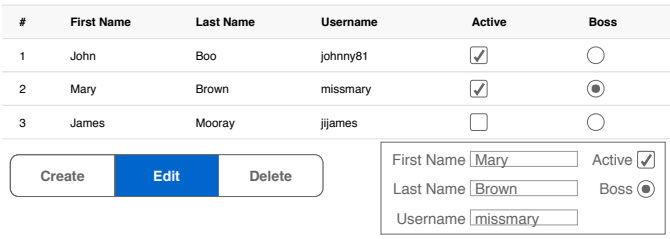


Fig. 2: Standard View with CRUD Functionality within CMS Extensions

source code. In total, we considered 50 extensions with 592 views and found that 212 views were standard list views and 191 were standard detail views. In short, 68.07% of the views we inspected are standard views with CRUD functionality. To gain further insight, in C7, we asked the interviewees for the amount of such standard views in their extensions. C8 allows obtaining information about the typical extension development procedure of the interviewees and potentials pitfalls during the process. Based on the answers of C7 and C8 it is possible to identify problems that can be addressed by an MDD approach (RQ2).

All interviews were recorded and transcribed. 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. Please also note that citations are verbatim taken from the transcripts. Citations that were translated to English are marked with a capitalized T as index.

TABLE I summarizes the answers of the interviewees. C1-8 are the categories defined above. The names of the participants were anonymised and abbreviated to P1-8. Each cell contains the participant's answer mapped to the corresponding category.

B. Scenario Evaluation

During the interviews we received the following answers which can be assigned to our three development scenarios:

1) Development of Independent Extensions: In the interviews the developers stated to require 2 to 8 hours to have an

independent installable component with no business logic following their usual (mostly clone-and-own) procedure: *"...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..."*.

Regarding to their usual procedure they stated: *"Copy&Paste and adapt what is necessary."*$_T$

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, they think that, 60 up to 100 percent are standard views.

2) Development of Dependent Extensions: Developers stated that they augmented 3rd-party and core components by 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!"*.

3) Migration of a Legacy Extension to a new Platform Version: During the interviews the respondents reported that it requires them a couple of days for these steps: *"...migrating to the next version took maybe a couple days."*. If the number of extensions and complexity grows it took them months, as some interviewees stated: *"You should take a year. [...] Maybe without any interruptions, it still will take a couple of months."* and *"It took me 6 months to migrate 10 extensions."*.

C. Interpretation

As shown in TABLE I, we interviewed industrial practitioners with many years of experience (C1 and C2) in developing and migrating all kinds of Joomla extensions (C3, C4 and C5). It takes them at least a couple of days, mostly months to migrate their extensions to a new Joomla version. An interesting finding is that no participant uses a tool for migrating extensions (C6). Their approach is mostly *"...fix errors until it works."*. P5 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 practitioner for their extensions. This shows that the practitioners have to put tremendous effort into the migration. The results of C7 further supports our own investigation and shows that the amount of standard views with CRUD functionality are 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 a new extension, most of them use the clone-and-own approach, which takes them at least hours to have a simple installable extension. The relevance of our scenarios, which we presented in Fig. 1, is strongly confirmed by the industrial practitioners' statements in section III-B1, III-B2, and III-B3 and coincides with our own observations. In respect to RQ1, we can conclude that the relevance is high.

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 requires large effort. With regard to RQ2. this can be directly addressed by MDD, since decreasing development effort for these scenarios is an acknowledged strength of MDD. So, the practitioners can significantly benefit from applying MDD of CMS extensions.

IV. Background: Used MDD tooling

Our interview study shows that there is potential for using MDD to develop and migrate CMS extensions for saving time and improving quality. In our experiments, we studied the practical benefit further, using an available tool infrastructure called JooMDD [9], [10]. JooMDD supports Joomla extension developers with a set of MDD tools: a text-based DSL with editors for modeling extensions, a code generator for generating implementations, and a model extractor for extracting models from legacy code. We selected JooMDD since it was the only available MDD infrastructure addressing all three scenarios of interest (see Sect. IX); however, the results may generalize to other tools as far as they address these scenarios. JooMDD and a detailed description of its components are publicly available at https://github.com/icampus/JooMDD.

The 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 code generator 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 illustrated in Sect. VI-A2.

Moreover, JooMDD supports developers during the reengineering or migration of a legacy extension (scenario 3). A model extractor for 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 model extractor is also useful in scenario 2: Usually, 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.

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 to the IDE integration, a platform-independent web IDE, comprising the whole tool set, can be found at https://tinyurl.com/joomdd-web. The web IDE allows developers to use JooMDD's full functionality without installing it locally.

V. Controlled Experiment

To evaluate whether MDD can significantly increase productivity during CMS extension development, we conducted a controlled experiment where we compared conventional programming with MDD exemplarily for the Joomla CMS. We aimed to verify the hypothesis that JooMDD increases productivity in terms of development speed during Joomla extension development. To this end, we address RQ3: Can MDD increase the productivity of CMS extension development for scenario 1 and 2?

A. Set-up

We selected 14 developers with significant expertise in Joomla extension development. 5 participants are industrial Joomla extension developers with a high level of experience (2 - 10 years of experience). 9 participants were students from an intensive course on Joomla programming, 5 of which also work as Joomla developers for productively used Joomla extensions in an university website context. We justify the selection of student participants with their comparable performance to professionals when using new software development tools [12]. Moreover, 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 knowledge in extension development. However, 2 participants showed a knowledge deficit in detailed extension development (MVC interaction).

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

The design followed a within groups design, in which 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 conventional programming and with MDD. 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. Group A had to implement the first requirement by hand and the second one with MDD, whereas group B 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 administration section (backend), as well as 1 list and 1 details view for the end-user (frontend). In addition, the subjects were asked to implement a dependent module (scenario 2) illustrating data of the implemented component. After each session, the subjects submitted their solutions.

Before the actual development session, a presentation about the experiment and the requirements for a complete solution was given. 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. After completing a demographic questionnaire, the subjects had to complete the knowledge assessment (development, Joomla, MDD). Afterwards, the two programming sessions followed. At the start of the first session, a requirements specification was given to all subjects. Group A had to implement Joomla extensions (1 independent component and 1 dependent module) for university management, group B 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 remaining requirement in a model-driven way. 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 MDD. After 9 hours the experiment ended.

B. Results

In TABLE II we summarize the results of the controlled experiment based on the amount of fulfilled requirements during the development sessions. For each subject we show the amount of fulfilled requirements based on relevant requirement groups in each development session. The *Component Structure* row indicates the overall percentage of fulfilled requirements by all participants considering a component that is installable, supports multi-language ability (by language files) and provides update scripts. The *Component Views* row specifies the overall percentage of implemented views so that the specified requirements (e.g. table columns, filters, orderings, correct fields and HTML field types) are fulfilled. The *Component CRUD* row gives the overall percentage of implemented CRUD functionality for each view, including the required buttons and a correct implementation of the associated actions. The *Module* row illustrates the overall percentage of fulfilled

TABLE II: Result of the Controlled Experiment.

		Requirement A			Requirement B			Overall		
Requirement group	**Scenario**	***Baseline***	***With MDD***	***Coeff.***	***Baseline***	***With MDD***	***Coeff.***	***Baseline***	***With MDD***	***Coeff.***
Component Structure	1	67%	90%	1.4	76%	86%	1.1	71%	88%	1.2
Component Views	1	5%	64%	13	4%	89%	21.5	5%	77%	16.9
Component CRUD	1	7%	76%	10.6	12%	97%	8.2	10%	87%	9.1
Module	2	14%	62%	4.3	0%	52%	N/A	7%	57%	8

requirements based on a module that is installable, uses the data of the implemented component, and illustrates the data in a module position. If requirements were not completely fulfilled, we rated them as partly fulfilled (2/3 or 1/3) whereas we differentiated between more and less than 50% completion. Requirement groups *Component Structure*, *Component Views*, and *Component CRUD* in union represent our scenario 1 (development of an independent component). Requirement group *Module* represents our scenario 2 (development of a dependent module).

To gain insight about the productivity growth, we build the average percentage of fulfilled requirements for each requirement group and calculated the respective productivity coefficient for both development sessions (baseline, with MDD). As TABLE II shows, the overall coefficient between the baseline session and the session with MDD varies between 1.2 and 16.9.

C. Interpretation

With the results of the experiment we answer RQ3: Can MDD increase the productivity of CMS extension development for scenario 1 and 2? The results show, that even the lowest coefficient is higher than 1 which shows that the subjects were more productive with MDD for each requirement group. During the development of views and the respective CRUD functionality, the subjects increased their average productivity by the factor 16.1 and 9.1. This corresponds to the interview statements and our previous research, that a tremendous amount of extensions consists of generic code for standard views with CRUD functionality. By applying MDD, these extension parts can be developed faster. This supports our hypothesis that MDD can substantially increase the development of CMS extensions.

The same applies to the development of dependent extensions, whereas the significance of the module development 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. 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.

VI. Field Experiment

The controlled experiment from the previous section supported a quantitative assessment of the potential productivity benefit of MDD in the CMS domain. Due to the high effort for understanding and implementing the 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. All developers had a high level of experience (5-13 years), leading to a good knowledge of the processes and problems during extension development. After an introduction to the selected MDD tool JooMDD, we observed the developers during the three development scenarios within a total time of 6 hours.

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. To provide direct feedback of the participants, we subsequently conducted interviews with the participants addressing the MDD approach during the scenarios.

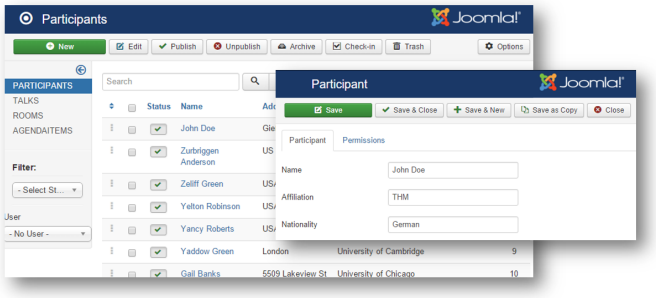
A. Set-up

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

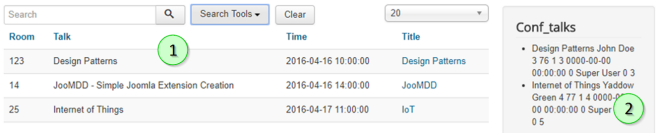
1) Development of an Independent Component (Scenario 1): In Joomla, components are the most commonly developed type of independent extensions. Therefore, we set the task to develop a conference management component as an extension to the Joomla core. To stipulate the requirements, we specified a class model 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. 3a.

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



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



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

Fig. 3: Standard Views with CRUD Functionality

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 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 of the conference entities works properly.

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. 4a.

2) 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. Fig. 3b 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. To this end, the participants could use the already downloaded extension package from scenario 1.

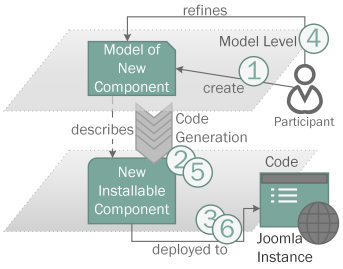
As next step, they had to upload the extension package to the JooMDD web IDE and use the JooMDD model extractor to extract a domain model from the conference component package. We decided for this component to make sure that the input extension matches the Joomla standard file and code schemes to ensure 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.

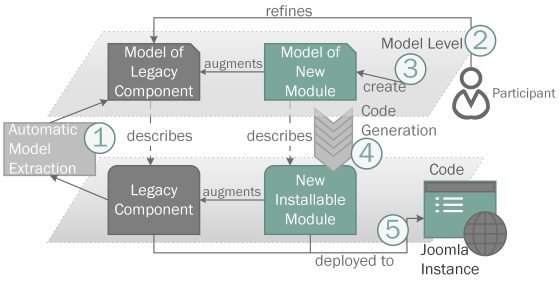
To complete this scenario, the developers were asked to install the module to our provided Joomla installation and, if it has been installed properly, create a module instance, which has to be placed on the frontend section of the website. If everything worked properly, the module had to illustrate the data of the already installed component similar to Fig. 3b. See Fig. 4b for an overview of the procedure.

3) Migration of a Legacy Component from Joomla 3 to Joomla 4 (Scenario 3): For the third scenario, we required the code migration of a component from Joomla platform version 3 to 4, which is scheduled for release in 2019. Even though the new major release of Joomla requires a completely new extension structure, 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.

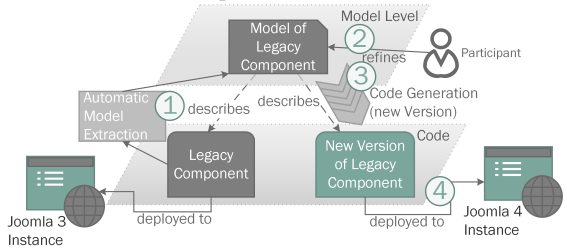
The first steps of the procedure, depicted in Fig. 4c, was similar to the ones described for scenario 2. The participants had to use an installable extension package of a Joomla 3 component, upload it to the web IDE, extract a model, and refactor that model. Again, we decided to use the conference component 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 4. Therefore, the participants had to inspect the new components to get an overview of the new component structure.



(a) Dev. of an Independent Component (Scenario 1)



(b) Dev. of a Dependent Module (Scenario 2)



(c) Migration of a Legacy Component from Joomla 3 to Joomla 4 (Scenario 3)

Fig. 4: Model-Driven Procedure of each Scenario

B. Observations

In the field experiment we made the following observations in the considered development scenarios.

1) 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 the introduced MDD infrastructure. 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, some participants had problems with keywords of the DSL. Especially, the *page* keyword in the model made some problems. The participants 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. Even though, 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 would have 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.

2) 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.

3) Migration of a Legacy Component from Joomla 3 to Joomla 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.

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.

C. 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 that: *"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 that: *"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 developers also expressed what they expect from an MDD infrastructure. One point often mentioned by the developers was that they want something like a wizard, which guides them through the extension creation: *"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, so he do not have to create it by hand: *"With commands, like 'build view x,y,z' and then it asks for the details."*$_T$

The 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 [13]: *"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 [...]. "*

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."*

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."*

VII. 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 [14]. 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 [15].) 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 sub-lessons:

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 pre-defined processes as they occurred in selected application scenarios.

In addition, we have found further lessons learned:

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.

VIII. 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.

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 is a threat to the conclusion 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.

Internal Validity The reliability of the results relies on the quality of the artefacts provided to the participants, in particular, the interview questions, tasks and examples. To mitigate the associated threat, we worked with examples and tasks that are already well-proven from use in teaching. In the controlled experiment, we anonymised the whole subject data to eliminate any biases.

External Validity The two main threats to external validity are: First, we only considered extensions of the Joomla CMS. It yet has to be studied if MDD is also suitable for other CMS, in particular WordPress, the most frequently used CMS. 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. An extensive description of the quantitative study considering the quality of the development results will be part of future work.

IX. Related Work

Our consideration of related work is two-fold: 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.

A. MDD in the CMS Domain

Several related works propose platform-independent metamodels for the development of specific CMS instances [16]–[18]. Code generation for concrete CMS instances was firstly investigated by Saraiva et al. in [19]. However, none of these works addresses extensibility scenarios of CMSs through standardized extension types taking their interdependencies into account. Only the XIS-CMS framework presented in [20] 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.

TABLE III: Tool support for scenario 1-3

Tool	Joomla Vers.	S.1	S.2	S.3
Component Generator [21]	3	✓	×	×
Joomla Component Builder [22]	3	✓	×	×
Component Creator [23]	3	✓	✓	×
Component Architect [24]	3	✓	✓	×
JCCreator [25]	3	✓	✓	×
JooMDD [9], [10]	3, 4	✓	✓	✓

In TABLE III, we collect existing tool support for MDD of Joomla extensions available online. The considered tools can 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 [26], [27] and [28] due to a lack of functionality. They generate a very basic scaffold only.

General MDD approaches for the web domain such as the ones in [29]–[31] 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.

B. Empirical studies on MDD in practice

There have been several efforts to investigate various aspects of MDD in practice. The practical adoption of MDD has been the focus of various studies [14], [32]–[36] 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 [36], whereas in [14], 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. VII.

Mohagheghi et al. [37] 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. They state that MDD can generally be applied successfully, while methodologies and tools are a main inhibiting factor.

The study in [38] investigates the usability of web applications being developed in a model-driven way. The closest study is probably presented in [39] where the maintainability of web applications is investigated. The authors compare model-driven development of web applications with 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*.

In contrast to that study, 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 the study in [39].

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

X. Conclusion

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. By conducting a controlled experiment, we compared conventional extension development with MDD. During the experiment we focused on the first two scenarios during the implementation of given requirements. The results showed a significant productivity gain by using an MDD infrastructure for extension development. In a field experiment we asked four experienced Joomla extension developers to use an MDD infrastructure for Joomla extensions, during the examples of all three development scenarios. So, we received direct feedback about acceptance, usefulness, and open challenges of the adopted MDD approach.

Conclusively, we share the lessons learned of the studies. Generally, we found that *focussing more on processes and people, not only on tools* which summarizes lessons learned by Whittle et al. [14], also helps to satisfy experienced developers in the CMS community. Since there is still little data about which application scenarios are good for MDD, we tried to identify three relevant scenarios. W.r.t. these scenarios we can conclude that *MDD-based migration support was welcomed the most*. In future work, we like to conduct more qualitative studies identifying further relevant development scenarios in the domain, even with developers for other CMSs like WordPress or Drupal.

REFERENCES

[1] T. Stahl, M. Voelter, and K. Czarnecki, *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc., 2006.

[2] P. Baker, S. Loh, and F. Weil, "Model-driven engineering in a large industrial context—motorola case study," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 476–491.

[3] W. Heijstek and M. R. Chaudron, "Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process," in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2009, pp. 113–120.

[4] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of mde in industry," in *Proceedings of the 33rd international conference on software engineering*. ACM, 2011, pp. 471–480.

[5] J. I. Panach, S. España, O. Dieste, O. Pastor, and N. Juristo, "In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction," *Information and software technology*, vol. 62, pp. 164–186, 2015.

[6] S. McKeever, "Understanding Web content management systems: Evolution, lifecycle and market," *Industrial Management & Data Systems*, vol. 103, no. 9, pp. 686–692, 2003.

[7] D. Barker, *Web content management: Systems, features, and best practices*. Beijing and Boston: O'Reilly, 2016.

[8] W3Techs, "Usage Statistics and Market Share of Content Management Systems for Websites, January 2018," 2018. [Online]. Available: https://w3techs.com/technologies/overview/content_management/all

[9] D. Priefer, P. Kneisel, and G. Taentzer, "JooMDD: A Model-Driven Development Environment for Web Content Management System Extensions," in *ICSE Companion '16: Companion Proceedings of the 38th International Conference on Software Engineering*. New York, NY, USA: ACM, 2016, pp. 633–636.

[10] D. Priefer, P. Kneisel, and D. Strüber, "Iterative Model-Driven Development of Software Extensions for Web Content Management Systems," in *Modelling Foundations and Applications: 13th European Conference, ECMFA 2017, Held as Part of STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings*. Cham: Springer International Publishing, 2017, pp. 142–157.

[11] Open Source Matters Inc., "Joomla! Extensions Directory," 2018. [Online]. Available: https://extensions.joomla.org/

[12] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 666–676.

[13] T. Kehrer, C. Pietsch, U. Kelter, D. Strüber, and S. Vaupel, "An adaptable tool environment for high-level differencing of textual models," in *International Workshop on OCL and Textual Modeling co-located with 18th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2015)*, 2015, pp. 62–72.

[14] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?" in *Model-Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science / Programming and Software Engineering. Berlin/Heidelberg: Springer Berlin Heidelberg, 2013, vol. 8107, pp. 1–17.

[15] N. Kahani, M. Bagherzadeh, J. Dingel, and J. R. Cordy, "The Problems with Eclipse Modeling Tools: A Topic Analysis of Eclipse Forums," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. New York, NY, USA: ACM, 2016, pp. 227–237.

[16] S. Martinez, "Towards an Access-Control Metamodel for Web Content Management Systems," in *Current Trends in Web Engineering*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2013, vol. 8295, pp. 148–155.

[17] F. Trias, "Building CMS-based Web applications using a model-driven approach," in *Sixth International Conference on Research Challenges in Information Science*. Piscataway, NJ: IEEE, 2012, pp. 1–6.

[18] K. Vlaanderen, F. Valverde, and O. Pastor, "Model-Driven Web Engineering in the CMS Domain: A Preliminary Research Applying SME," in *Enterprise Information Systems*, ser. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 19, pp. 226–237.

[19] J. d. S. Saraiva, "Development of CMS-based Web Applications with a Multi-Language Model-Driven Approach," Dissertation, Universidade Técinica de Lisboa, Lisbon, Portugal, 01.01.2012.

[20] P. Filipe, A. Ribeiro, and A. R. da Silva, "XIS-CMS: Towards a model-driven approach for developing platform-independent CMS-specific modules," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, pp. 535–543.

[21] S. ApS. (2019) Component generator. [Online]. Available: https://www.componentgenerator.com/

[22] V. D. Method. (2019) Joomla component builder. [Online]. Available: https://www.joomlacomponentbuilder.com/

[23] J. T. SL. (2019) Component creator. [Online]. Available: https://www.component-creator.com

[24] S. O. Source. (2019) Component architect. [Online]. Available: https://www.componentarchitect.com

[25] M. ALNASSER. (2019) Jccreator. [Online]. Available: https://jc-creator.com

[26] S. Software. (2019) Module-creator. [Online]. Available: https://extstore.com/tools/module-creator

[27] xdsoft. (2019) Joomla module generator. [Online]. Available: https://xdsoft.net/joomla-module-generator/

[28] (2019) Boilerplate files for joomla! extensions. [Online]. Available: https://github.com/joomla-extensions/boilerplate

[29] M. Brambilla, *Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML*. Waltham, MA: Morgan Kaufmann, 2015.

[30] A. Kraus, A. Knapp, and N. Koch, "Model-Driven Generation of Web Applications in UWE," Ph.D. dissertation, Ludwig-Maximilians-Universität München, München, 01.01.2008.

[31] S. Ceri, P. Fraternali, and A. Bongio, "Web Modeling Language (WebML): a modeling language for designing Web sites," *Computer Networks*, vol. 33, no. 1-6, pp. 137–157, 2000.

[32] H. Burden, R. Heldal, and J. Whittle, "Comparing and Contrasting Model-driven Engineering at Three Large Companies," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14. New York, NY, USA: ACM, 2014, pp. 14:1–14:10.

[33] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based Engineering in the Embedded Systems Domain: An Industrial Survey on the State-of-practice," *Software & Systems Modeling*, vol. 17, no. 1, pp. 91–113, 2018.

[34] S. Vaupel, G. Taentzer, R. Gerlach, and M. Guckert, "Model-driven development of mobile applications for android and ios supporting role-based app variability," *Software and System Modeling*, vol. 17, no. 1, pp. 35–63, 2018.

[35] S. Vaupel, D. Strüber, F. Rieger, and G. Taentzer, "Agile bottom-up development of domain-specific ides for model-driven development," in *Workshop on Flexible Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2015)*, 2015, pp. 12–21.

[36] V. Sousa, E. Syriani, and M. Paquin, "Feedback on how mde tools are used prior to academic collaboration," in *Proceedings of the Symposium on Applied Computing*, ser. SAC '17. New York, NY, USA: ACM, 2017, pp. 1190–1197.

[37] P. Mohagheghi, W. Gilani, A. Stefanescu, and M. A. Fernandez, "An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases," *Empirical Software Engineering*, vol. 18, no. 1, pp. 89–116, 2013.

[38] A. Fernandez, S. Abrahão, and E. Insfran, "Empirical Validation of a Usability Inspection Method for Model-driven Web Development," *J. Syst. Softw.*, vol. 86, no. 1, pp. 161–186, 2013.

[39] Y. Martinez, C. Cachero, and S. Meliá, "Empirical Study on the Maintainability of Web Applications: Model-driven Engineering vs Code-centric," *Empirical Softw. Engg.*, vol. 19, no. 6, pp. 1887–1920, 2014.