information

MDPI

Article

A Process Model for Component-Based Model-Driven Software Development

Afrah Umran Alrubaee [1,†], Deniz Cetinkaya [2,*], Gernot Liebchen [2] and Huseyin Dogan [2]

1 Department of Computer Science, University of Kerbala, Karbala 56001, Iraq; afrahumran15@gmail.com
2 Department of Computing and Informatics, Bournemouth University, Poole BH12 5BB, UK; gliebchen@bournemouth.ac.uk (G.L.); hdogan@bournemouth.ac.uk (H.D.)
* Correspondence: dcetinkaya@bournemouth.ac.uk
† Previous address: Software Engineering Department, Atılım University, Ankara 06836, Turkey.

Received: 20 April 2020; Accepted: 1 June 2020; Published: 5 June 2020

Abstract: Developing high quality, reliable and on time software systems is challenging due to the increasing size and complexity of these systems. Traditional software development approaches are not suitable for dealing with such challenges, so several approaches have been introduced to increase the productivity and reusability during the software development process. Two of these approaches are Component-Based Software Engineering (CBSE) and Model-Driven Software Development (MDD) which focus on reusing pre-developed code and using models throughout the development process respectively. There are many research studies that show the benefits of using software components and model-driven approaches. However, in many cases the development process is either ad-hoc or not well-defined. This paper proposes a new software development process model that merges CBSE and MDD principles to facilitate software development. The model is successfully tested by applying it to the development of an e-learning system as an exemplar case study.

Keywords: software development process model; software engineering; component-based development; model-driven development; model-driven software development; reusability; reusable components; metamodeling

1. Introduction

Developing a high quality software system is not an easy task because there are many challenges during the development process in different stages. Unclear requirements, any kind of changes during the development process, bad communication among the development team members, problems in the analysis stage, design issues, etc., are some examples of these challenges. With the increase in size and complexity of software systems, the problems become more critical. In order to overcome these challenges and to develop high quality, cost effective and reliable systems, various development approaches have been proposed. Two of these approaches are Component-Based Software Engineering (CBSE) and Model-Driven Software Development (MDD). These two different approaches both attempt to produce high quality, low cost and on time software systems with a focus on reusability and productivity. However, there are significant problems related to the use of models and components during the development of software systems. This paper proposes a novel approach to merge MDD and CBSE principles in a well-defined process.

CBSE is a software development approach that proposes constructing software systems from existing software components instead of developing systems from scratch [1]. The goal of CBSE is reducing cost and producing high quality and more reliable systems. Szyperski [1] defined the software component, as "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". Although almost 20 years passed, there is not much change in this definition. The component-based software engineering community generally agrees that a component is an independent software unit which can be composed of other components through well-defined interfaces to develop new and larger software systems [2].

One of the most important features of a software component is that it can be considered as a sub-system, which can be deployed and executed directly. To achieve this, a component should follow the standards of a component model and conform to a specification. To enable composition between independently developed components, commonly accepted standards are needed to explain composition and interaction mechanisms such as CORBA Component Model, Component Object Model (COM) or Enterprise JavaBeans (EJB) [3]. In this way, not only reusability and productivity are provided but also interoperability, upgradability and maintainability of the new system become easier.

CBSE needs software components to be available in well-organized component repositories, which facilitate searching and finding the required components [4]. Therefore, CBSE includes two development processes. The first one is component development process that deals with developing reusable software components and storing them in repositories in such a way that makes them searchable and reachable. The second process is software system development which is about constructing the system from existing pre-developed components.

When existing components are used in CBSE, the effort needed for design and implementation is reduced. However, additional processes are added to the development process such as component assessment and adaptation. Component assessment includes searching for candidate components that may satisfy desired functionalities, selecting the most suitable components from these candidates and verifying their functional properties [5,6]. Different techniques for searching activities can be used such as keyword based search, signature matching, classification based search, etc. [7]. Component adaptation means adapting the selected components according to the requirements and includes setting the interfaces to be able to integrate with other components to construct the target system. There are various adaptation techniques such as using wrappers, parameters, adapters, etc. [5,8–10].

Another software development approach that focuses on reusability and productivity is Model-Driven Software Development, which is abbreviated as MDD or MDSD in the literature and also known as Model-Driven Engineering (MDE). MDD is a software development approach which aims to address the challenges of software development process through representing the essential aspects of the required system as models and generating the final source code from these models (semi)automatically [2,11]. A model is an abstraction of a system, which shows the main properties of the system and excludes unnecessary details, which leads to reducing the complexity of the system [12]. Models enhance the communication among development team members and provide better understanding.

There are many applied research studies that show the benefits of using software components and model-driven approaches [13,14]. However, in many cases the development process is either ad-hoc or not well-defined in these studies. While MDD and CBSE provide useful mechanisms to increase reusability they focus on different aspects. MDD focuses more on the code generation from high level abstractions whereas CBSE focuses on component integration via interfaces. Their success is dependent on each other because focusing only one aspect cannot realise the potential benefits of each approach. On the other hand, currently guidelines do not exist to apply these approaches together in an effective way [15]. There is a lack of practical framework and process model to utilise component models at different stages.

This paper proposes a software development process model that merges MDD and CBSE principles to facilitate software development. The main research question in this study is "Which lifecycle stages can be identified and integrated in a software development process model that merges model-driven and component-based approaches?". The proposed process model embeds MDD principles in a CBSE lifecycle to generate models for components. These models are transformed to intermediate models until they reach to a final source code. The main contribution of this paper is the definition of a novel software development process model which merges MDD and CBSE to introduce a model driven approach for component development within the development lifecycle. Components and their associated models are saved in a repository to be reused. Our process model also introduces a reusability assessment step to make the process more efficient. Each stage in the process is explained and a sample workflow is illustrated to apply the proposed model. An exemplar case study is used to validate the model which follows the suggested workflow and definitions.

The outline of the paper is as follows: This section provides a brief introduction to our research. Next section provides a brief background information about MDD and CBSE. Section 3 presents the related work. Section 4 proposes a component-based model-driven software development process model that merges the CBSE and MDD approaches. The software development lifecycle and process workflow are presented in Section 5 with description of each stage in the lifecycle. Section 6 presents a working example to validate the proposed model. Finally, Section 7 concludes the paper and discusses the future work.

2. Background Information

2.1. MDD Concepts

The process of representing a system as a set of models to abstract the essential aspects is called modeling and the main means of modeling process is the modeling language. Any modeling language should consist of three elements that are the abstract syntax, concrete syntax and semantics. The abstract syntax consists of the concepts, the relationships, and well-formedness rules, where well-formedness rules state how the concepts may be combined. The concrete syntax determines how the modeling elements are illustrated when the language is used. The expression generated with the concrete syntax is called the model. The semantics of a modeling language is the additional information to explain what the abstract syntax actually means. The abstract syntax contains a set of modeling concepts and rules that specify well-formed expressions of these concepts [11,12].

In MDD literature, the most common method to define modeling languages is metamodeling [11,16]. In the context of MDD, metamodeling is the process of complete and precise specification of a modeling language ml_A in the form of a metamodel mm_A, which in turn can be used to specify models by using the ml_A language. The metamodel mm_A is described by a modeling language as well, which is mostly different than ml_A language and it is called as metamodeling language mml_X. There are several well-known metamodeling languages like Meta-Object Facility (MOF), Eclipse Ecore, MetaGME, etc. [11]. In MDD, different models are generated and each of these models represents a system in different levels of abstractions from different perspectives. This is because important aspects may be different at various points in time of software development lifecycle.

Due to the fact that there are different models in MDD and the goal is automated code generation, the core activity of MDD becomes the model transformations. Model transformation is the process of transforming a model to another form. The resulting form of model transformation can be a new model at a different level of abstraction, so the transformation process can be called a model-to-model transformation, or the result can be source code, in which the model transformation is called code generation. The model transformations in MDD are formal, generally supported by metamodels, and take place according to the rules of a model transformation language, such as QVT or ATL, where the main objective is to systematize model transformations [17].

2.2. CBSE Process Models

There are various CBSE process models in the literature. We briefly explain them here and provide further links to the original sources for interested readers. A further analysis is presented in Section 4.

Somerville [2] proposes a sequential process that contains six steps. The first one is outlining the requirements stage. The second one is trying to find components that coincide with the requirements.

The next step is modifying the requirements to comply with the components; this step is for negotiating the requirements, after that the system will be designed, then searching for components again, and at last integrating the component to construct the final system. It is assumed that components do not need any major adaptation to satisfy the requirements which is actually not very realistic.

Crnkovic et al. [5] proposed the V Development Model and adapted this model for traditional software development. They separated the lifecycle into two processes such as system development and component development where each of them follows the V model. In addition, the paper introduces new steps to system process model which are selecting, adapting and testing the components. Capretz [18] proposed the Y model that considers instability and change so it allows iteration in all stages. It uses domain engineering in component development process and then the components are archived. After that, components are selected from the archive according to a framework. Y model applies V model in system lifecycle.

Another development model, which is named with a letter as others, is W Development Model that is proposed by Lau et al. [19]. In this model, lifecycle consists of two V models, one for component development and another for system development. Component selection, adaptation, and deployment stages represent the link between the two V models. The purpose of using V model with component lifecycle is to enable component verification and validation before storing it in a repository. Tomar and Gill [20] proposed the X development model which includes four sub-cycles, one for component-based software development and three for component development. They consider three cases for component-based development, which are development for reuse, development with modification and development without modification.

3. Related Work

Many studies introduced component-based approach as a solution to provide reusability and manage the complexity during the software development process. Vale et al. [21] investigate the state-of-the-art of the CBSE through a systematic mapping study. They analyse main objectives, research topics, application domains, research intensity and applied research methods of CBSE. The main objectives found were to increase productivity, save costs and improve quality. Badampudi et al. [13] present a systematic literature review about component selection and component origins as well. Main concerns of CBSE relate to the technologies and tools since tool support is essential for the success of CBSE. For this reason, many studies emphasize the tool support. However, tool support is not enough if there are not formal specifications, process models and frameworks utilised as a baseline.

Similarly, model-driven approaches have been commonly used to reduce the design and implementation time and to increase the productivity during the software development process [14,22,23]. Da Silva [22] presents a survey on model-driven engineering based on a unified conceptual model and provides background information about MDD. Although MDD has been shown to be an applicable and cost-effective approach for software development, it could not reach its potential in the software industry due to challenges [24]. To increase the use of available MDD tools and methods, a recent trend in MDD literature is combining MDD with other software development approaches. For example, Alfraihi and Lano [14] present a systematic literature review on combining agile development approach and MDD. Authors state that agile development and MDD can coexist and benefit from each other. However, they also conclude that methodological aspects have not been discussed in most studies.

Since the practical aspects of MDD are more noticeable, researchers mostly focus on the implementation of the MDD approach and perform applied research. However, theoretical and methodological aspects should also be studied and formal frameworks should be provided. Mussbacher et al. [25] present a study where 15 MDD experts joined to identify the biggest problems and grand challenges of MDD. They recognize various major problems, and the most common three of them are highlighted here. The first one is the shortcomings of MDD to address increasing demands on software. The ever-increasing need to customize and tailor software to specific usage contexts is evident but current modeling approaches, techniques and tools cannot cope with the challenges when applied to complex systems or systems of systems. The second shortcoming is the lack of fundamentals sources and guidelines in MDD. Mussbacher et al. [25] state that unlike most other fields of engineering, model-driven engineering does not have a Body of Knowledge (BoK) as such. However, there is a great effort in the academia to formalize the MDD process [11] and improve the usage of MDD in complex system design and development [26,27]. Finally, the last shortcoming is the inconsistencies between software artefacts. The authors highlight the issue of synchronization of the interim models between different levels of abstraction. Our proposed process model will open new ways to overcome these challenges by introducing the adaptable components into the MDD process and defining the components with their associated models.

As a related work, the number of studies that combine CBSE and MDD on the methodology level are very limited in the literature. This is most probably because these approaches are thought as an alternative to each other. However, there are still some attempts that apply CBSE and MDD together. Atkinson et al. [28] present the initial ideas to strictly separate the abstract description of components from their implementation, and they focus on separation of concerns by applying component-based and model-driven approaches together in the development of a system.

Weiss et al. [29] present an approach to facilitate generic reusable specifications and models to create spacecraft systems and sub-systems within a model-driven development process. These reusable specifications can be tailored for a specific design, executed and validated in a simulation environment, and transformed into the real system. Phung-Khac et al. [30] present an approach to develop component-based adaptive distributed applications by using model transformations to build evolutionary variants of the components at deployment level. The authors separate the communication and the functional aspects of a distributed application, and try to automate the development process.

Kainz et al. [31] propose a three-phase model-driven development approach to build component-based embedded systems. The authors split the development process into run-time architecture development, component development and application development phases where each phase builds on the input of the previous phase. In each phase the corresponding expert is able to model the relevant properties of the system and to concentrate on the aspects of the system related to their expertise. They present model-to-metamodel (M2MM) transformations, where the metamodel in a level could be generated from the model at the previous level. Liu et al. [32] present the roadmap of the theory and tool support for component-based model-driven software development. They discuss the concepts, techniques and design decisions in their research. The proposed approach supports multiple dimensional modeling such as hierarchy of components and different views of the system. Clemente et al. [33] provide a fully Model Driven Architecture (MDA) compliant approach that allows component-based systems to be driven from the early stages of the component lifecycle onwards. The authors propose a highly practical approach and present a prototype implementation with further empirical analyses.

Kathayat et al. [34] present a service-oriented development approach leading to reusable components derived from service models. The study discusses different kinds of reusable components as collaborations to provide services, partially supporting components, and components for system compositions. Herrero Agustin and Del Barco [35] propose a framework for model-driven web application development by composing heterogeneous web components. They propose a UML profile for web development and present a transformation model to generate web applications from the design models. Mizuno et al. [36] propose an approach that applies component-based development technologies that have mainly been developed at the software implementation level to the modeling level. They investigate a method of loosely coupling model elements by applying the component concept to model-based development.

Although there is evidence of applied research on this topic in the literature, most of the aforementioned work does not follow or propose a well-defined process model or a framework that fits to the application process. In many cases the development process is either ad-hoc or not well-defined.

Most of the research deals with either service-level components or applied in a specific domain such as embedded systems. All of these studies show the applicability of MDD and CBSE approaches together with emphasizing the benefits. On the other hand, the interplay of CBSE and MDD is still a challenging task due to several reasons. Recent studies identify the challenges related to the interplay of CBSE and MDD and show that the subject is still interesting to the research communities [36,37]. One of the most fundamental research gaps is the lack of underlying specifications and guidelines for the development process. In this paper, we propose a software development process model, namely CompoMDD, that merges MDD and CBSE approaches to bridge this gap. The model aims to focus on low-level software components while supporting service-level components as well.

4. CompoMDD: Component-Based Model-Driven Software Development

This section presents the Component-Based and Model-Driven software development process models utilised. The main purpose of combining the two approaches is to take advantages of MDD and CBSE such that the resulting system will have higher quality, and it will be more reliable and maintainable with less effort.

The CompoMDD process model embeds the stages of model-driven development in component-based software development lifecycle. This model consists of two main activities: component development and software system development [2,18]. Component development activity focuses on developing new components from scratch. It mainly follows the stages of software development lifecycle, which are somewhat similar to requirements analysis, design, implementation, test and maintenance. On the other hand, the component-based software system development activity focuses on selecting, evaluating, adapting and testing the appropriate components. There is not much effort required for design and implementation.

When the CBSE process models in Section 2.2 and the CBSE practice are analysed, it can be identified that every component should have the ability to be deployed independently, to be composed with other components, and to probably provide services. Besides, a component should be able to interact with other components via well-defined interfaces. They should be developed according to a standardized component model, and they should have detailed documentation to facilitate potential reuse. Table 1 shows the results of the analysis based on a comparison of process models which include a component adaptation step.

Table 1. Analysis of existing component-based process models.

Models	**V [5]**	**Y [18]**	**W [19]**	**X [20]**
Components can be deployed independently	Yes	Yes	Yes	Yes
Components can be composed with other components	Yes	Yes	Yes	Yes
Components can interact with other components	Yes	Yes	Yes	Yes
Separation between component development and system development activities	Yes	Yes	Yes	Yes
Adaptation step to modify the components	Yes	Yes	Yes	Yes
Component verified and validated before it is stored in repository	Yes	No	Yes	Yes
Use of domain engineering	No	Yes	Yes	Yes
Embedding another software development approach	No	No	No	No
Considering the potential reuse of components	No	No	No	No

The comparison shows that none of the analysed models embed another software development approach into to the process to increase productivity. Moreover, none of them considers the potential reuse of components, but instead they all assume that all components are developed to be reusable. We introduce a reusability assessment step into the process compared to the existing studies.

From a methodological point of view, we merge the best practice in CBSE and MDD to propose a new process model for component-based software engineering which uses model-driven development methods to improve the overall process. Model driven development stages are mainly adopted from the formal MDD framework of Cetinkaya et al. [11] which defines an MDD process as a tuple

$$mdd = \{n, MML, ML, MO, SL, pl, MTP, STP, MT, sc, TO\} \quad (1)$$

where:

n is the number of models;
MML is an ordered set of metamodeling languages;
ML is an ordered set of modeling languages (preferably defined with metamodels);
MO is an ordered set of models, m0 is the initial model;
SL is a set of supplementary languages (including at least a model transformation language);
pl is a programming language for final code generation;
MTP is a set of formal model transformation patterns;
STP is a set of other supplementary model transformation patterns;
MT is a set of formal model transformations;
sc is the final source code; and
TO is a set of tools to ease the activities.

MDD is utilised during component conceptual modeling and reusable component development. CBSE process is improved with reusability assessment and adaptation steps. Sub-processes are enhanced by allowing iterations. Domain engineering and domain specific modeling in the context of MDD are utilised during component model specification.

The proposed model is shown in Figure 1 and is called as CompoMDD. The model represents an integrated approach to build software systems using component-based development and model-driven development. It consists of two development processes, namely software system development and component development, which complement each other.

The process starts with requirements elicitation and ends after system validation within the system development process. Component development is done after conceptual modeling and reusable components are saved into the repository with all their associated models including the conceptual models. System design process employs component based approach whereas component search covers related model search as well. Repositories contain both models and implementations where various models and component source code are clearly labelled to identify their type. Component integration step in system design helps to specify a system model by integrating the adapted component models. System implementation stage includes the system integration with both reusable and non-reusable components. Each stage is explained in detail in the following section from a project-oriented system development perspective.

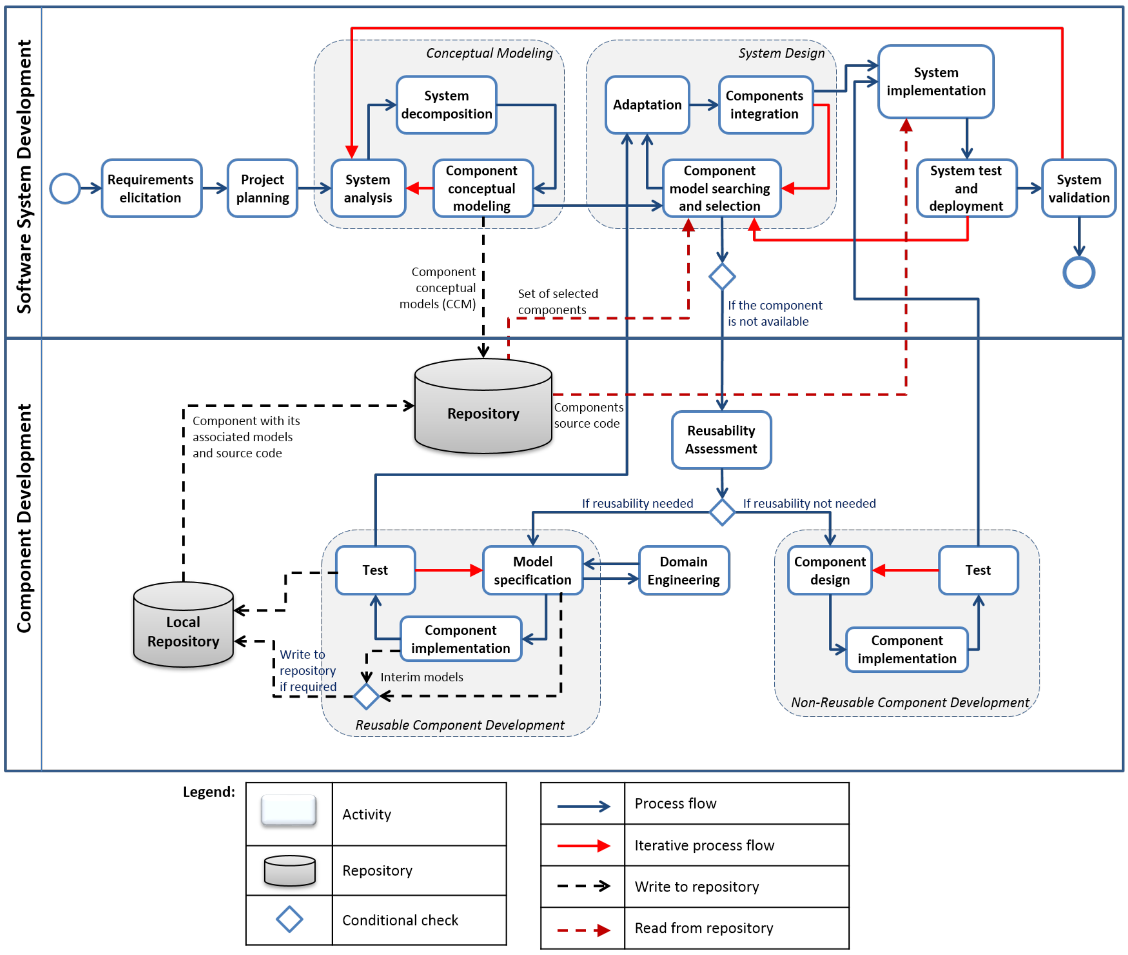


Figure 1. CompoMDD process model.

5. Description of the CompoMDD Process Model

This section explains the aim and activities of each stage defined in the CompoMDD process model. When we look at the overall process model, we see that software system development process has two sub-processes which are conceptual modeling process and system design process. Component development also has two sub-processes which are reusable component development and non-reusable component development. For each component, only one of them should be executed according to the reusability assessment result. There are two repositories, one of them is the local one and the other one is the general repository. Although the general repository is shown as a single resource in the diagram in Figure 1, it can represent multiple distributed repositories over a network. The repository should have efficient classification structures and retrieval methods. Additionally, advanced algorithms to identify ill-defined queries or to propose alternative search suggestions are needed. Existing search tools and methods in the literature may be used, but there may be issues specific to the component repositories.

5.1. Software System Development

This sub-process deals with the software system development and aims to satisfy overall project requirements.

5.1.1. Requirements Elicitation

The first step in software system development is requirements elicitation as common in most software development models. In this step, the requirements are collected from the users and stakeholders, and are documented. Although a detailed requirement specification document does not exist at this stage, it provides a basis for planning.

5.1.2. Project Planning

Next, the purpose and the boundaries of the project are defined based on the customer requirements to understand the main characteristics of the system. At this step, the project plan should be defined with a list of activities, deliverables, time and budget constraints, resources, etc.

5.1.3. System Analysis

After the project plan is agreed, the analysts understand and analyse the system in-depth. They define functional and non-functional requirements of the system and start thinking about the conceptual models of sub-systems. This step includes requirement analysis and specification. The detailed requirements specification document should be officially agreed at this stage.

5.1.4. System Decomposition

Component-based approaches focus on dividing the system into logical or functional components which are connected to each other by well-defined interfaces. In the CompoMDD model, after the system is analysed and the essential characteristics are defined, the next step in the lifecycle is system decomposition. At this step, the problem is decomposed into sub-problems, the system is decomposed into sub-systems, and the sub-systems are decomposed into components to facilitate separation of concerns. The outcome of this step is the System Decomposition Model (DM) that describes the system's conceptual design and shows the components of the system and their relations. This step includes the discussions and decisions about the system architecture and existing available solutions since they may affect the system decomposition [38]. In the CompoMDD model, developers should use a system modeling language to define the DM.

5.1.5. Component Conceptual Modeling

After the system decomposition model is designed, a high-level abstraction should be made for each component in the system in order to prepare a conceptual model for each component. We will call it Component Conceptual Model (CCM), and it should be defined according to the requirements, system boundaries, constraints, etc. A CCM does not include any platform specific details so they are not executable. In the CompoMDD model, developers should use a conceptual modeling language to define the CCMs, and all CCMs should be uploaded to the general repository. After this step, analysts and modellers can follow an iterative process to improve the system analysis and decomposition to reflect the issues or challenges during conceptual modeling.

5.1.6. Component Model Searching and Selection

This step is all about searching and selecting components. However, in order to be able to search for components, the general repository should have component models and their associated implementations. Component models can be platform independent or platform specific models which will be developed during the component development process. Identifying and selecting a subset of component models that satisfy the requirements to construct the target system is a challenging task, and it requires advanced mechanisms and techniques such as keyword search, semantic mapping, meta-information search, etc. At this step, the modellers should aim for high rate of reusing existing components. During the system design process, components can be searched according to their meta information and existing model definitions in the repository. Component libraries or standardised interfaces can be used to satisfy interoperability. In CompoMDD model, a component does not only have a specific implementation but instead has a conceptual model, a design model, and one or more platform specific implementations. For example, if a suitable component conceptual model is found in the repository then it will be included in the system design whereas its associated models and platform specific implementation will be included into the system in later stages by using MDD approach.

5.1.7. Adaptation

Once a set of component models are found and selected from the repository, the next step will be the adaption. Each component should be evaluated for adaptation and should be tailored according to the system requirements. Again, this is another challenging part of the component-based approaches since the granularity of the components varies a lot. The modellers should note that the components adaptation cost and effort should not be more than the cost and effort of building a new component. If a component is not available in the repository and required in the DM, then the flow continues with the component development process to develop the component models and the associated implementation from scratch. Component development process is explained in Section 5.2 where MDD approach is adopted during the reusable component development process. During adaptation, platform specific details can be identified for selected components so that components integration can be done in the next step.

5.1.8. Components Integration

After all needed components become available by either selecting from a repository or by building them from scratch, a composite model containing all selected component models with their relations should be constructed. This model will represent the design model for the system and will be called as System Design Model (SDM). SDM can include both platform independent and platform specific component models, but they need to be clearly labelled. If there is any need to return to the component selection or development steps, then it is possible to return back to the component model searching and selection step. Otherwise, the flow continues to system implementation. When moving from design level to implementation, model-to-model transformations are used to refine the higher level design models into more detailed ones. During this stage adapted components are integrated to the system by using their associated models and meta information.

5.1.9. System Implementation

At the implementation stage, available source code in the repository for the used component models is automatically extracted and the source code is generated. This can be provided with defining links to the related source code in the platform independent and platform specific component models. Alternatively and preferably, models can be transformed to source code via formal model transformations and the component source code can be generated automatically. If the components are service level components then the parameters to use the services should be set up according to the component models. Model-to-text transformations are used for code generation. In addition, model-to-text transformations can be utilised to generate configuration files, documents, reports or test cases [39].

5.1.10. System Test and Deployment

Finally, the system should be ready and executable at this level, which however does not happen very often in practice. If the system is ready and executable, it should be deployed and tested to see that the functions of the system meet the requirements. If there are any problems with the system at this level, the process flow goes back to the component model searching and selection step, not to the implementation step. Any defects in the models or bugs in the code should be corrected, and any missing components should be implemented.

5.1.11. System Validation

Once, the system is tested then validation tests should be done to ensure that the system provides a solution to the initial problem. If system validation is successful then it can be said that the development process finishes and the maintenance process can start. Otherwise, analysts and modellers go back to the system analysis stage.

This section explained the software system development process. To understand it fully, the component development process should also be understood well. So, the next section explains the component development process.

5.2. Component Development

This sub-process deals with the software component development and aims to satisfy individual component requirements.

5.2.1. Reusability Assessment

Before starting to build a new component it should be decided whether we want to create this component as a reusable or non-reusable component because developing a reusable component is more costly [40]. This is the reason that the amount of the component's potential reuse should be estimated to allow the estimation of the reuse effectiveness. Reuse effectiveness is defined as "the ratio of reuse benefits to reuse costs" [41,42]. It can be measured by multiplying the amount of reuse by the ratio of the difference between what the development of a new component would have cost, and what it costs to reuse the component to the investments costs to develop the reusable component [41].

Reusable Component Development:

If it is decided to develop the component as reusable then a new component development lifecycle starts which adapts an MDD approach, and it consists of the sub-stages model specification stage, component specification stage, and test stage, which are described in the following sections.

5.2.2. Model Specification

If a required component was not found in the repository, it should be built. When we decide to build a new component we need expert knowledge related to the domain of the component to have a better understanding and to make the component more reusable. This experience can be gained from domain engineering which "deals with collecting, organizing and storing past experience in building system or part of system in particular domain" [43]. Using the domain knowledge and the conceptual model, a Component Platform Independent Model (CPIM) should be defined by transforming the CCM into a formal model by using a system specification language. Model driven approach and domain specific modeling techniques are used at this stage to develop metamodels and metamodel-based model transformations. The specification model should describe the functionality, behaviour and potential interaction of the component to be realized in the implementation model. Platform independence concept is adopted from MDD so a CPIM model represents a component from a platform independent point of view and hides implementation specific details.

5.2.3. Component Implementation

At this stage, the CPIM is transformed into the implementation model. The implementation tools, techniques and the platform should be specified. An essential concern of the component implementation is how to develop a component that is reusable as well as interoperable. MDD solves this issue with platform specific models. Hence, in the MDD context, we say that the Component Platform Specific Model (CPSM) should be defined at this stage. Then, the source code of the component is generated from the CPSM. For each CCM, it is possible to define more than one CPIM and similarly for each CPIM, more than one CPSM can be defined. The links between these models can be pre-defined manually entered links, or more preferably model transformation rules can be defined from the CPIM to the CPSM. CPIM and CPSM models should be defined with well-defined modeling languages. Metamodeling can be used to define new modeling languages and formal model transformations can be defined. If existing modeling languages are used then existing model-to-model or model-to-text transformations can be used to facilitate component development. Section 5.3 discusses the metamodels in the CompoMDD model.

5.2.4. Test

The test stage is used for verifying and validating the component models and the component source code. A test plan and test cases should be defined and run to get the results. The results will be analysed to fix the bugs and defects. If the component passes the test successfully, then the component with its associated models will be stored in a general repository and adapted to integrate within the software system. If required, model specifications and the code can be uploaded to a local repository during the component development lifecycle for versioning and backup purposes. Local repository should be used during development and for testing purposes. MDD can help to generate documentation or test plans by using the local repository. On the other hand, main repository should be used for actual system design and implementation. Please note that, this model can be applied in different organisations and each of them may have their own main repository. In this case, common or shared repositories can be defined as well.

Non-Reusable Component Development:

If it is decided to develop the component as non-reusable then a new component development lifecycle starts which adapts a traditional software development approach. The stages of developing a non-reusable component can be similar to the stages of developing traditional software because the number of potential reuse of this component is low and so extra effort for making the component reusable is not needed. It is still possible to develop a non-reusable component using the same steps of reusable component development without taking into consideration the reusability requirement. The resulting component will be integrated directly into the system and will not be stored in the repository.

The component development process is part of the system development process. So, once all of the required components are developed then the process flow continues with the system implementation stage which is defined in the previous section.

5.3. Metamodeling

Metamodeling is a modeling process to generate a model that defines the abstract syntax of a modeling language, which includes concepts, relationships and well-formedness rules [11]. The output of this process is a metamodel and we use a metamodeling language such as MOF or Ecore to define the metamodel [44]. In CompoMDD, since we adapt an MDD approach, there are a number of metamodels that need to be generated within the development lifecycle. Figure 2 shows the models and metamodels in the CompoMDD process model. The metamodels are:

- *Component conceptual metamodel (CCmeta):* This metamodel is a high level description of the components, inputs, outputs, attributes and methods and their relations without technical details. It will be used to define component conceptual models.
- *Component platform independent metamodel (CPImeta):* This metamodel is a description of the grammar of the component specification language. It will be used to define component platform independent models.
- *Component platform specific metamodel (CPSmeta):* The entities and relations in this metamodel correspond to the entities and relations in CPImeta and there may be additional entities or additional features as well. CPSmeta describes these entities and relations for a specific platform

implementation. It will be used to define component platform specific models and the final source code will be automatically generated from these models.

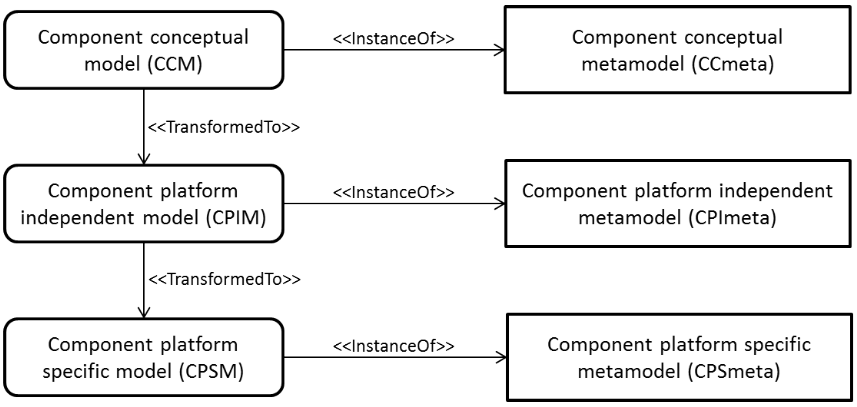

Component conceptual model (CCM)
<<InstanceOf>>
Component conceptual metamodel (CCmeta)
<<TransformedTo>>
Component platform independent model (CPIM)
<<InstanceOf>>
Component platform independent metamodel (CPImeta)
<<TransformedTo>>
Component platform specific model (CPSM)
<<InstanceOf>>
Component platform specific metamodel (CPSmeta)

Figure 2. Metamodels and models in CompoMDD process model.

The generated CPIMs will have more potential reuse and more impact if domain knowledge is added to the models. However, that will make the model transformations more complex, time consuming and the probability of error will be high. Metamodel-based formal model transformations will help to make model transformations faster and easier.

5.4. Outputs of the CompoMDD Process Model

This section provides a list of outputs of the proposed process model by summarising the stages explained in previous sections. Table 2 represents all of the outputs of in the process model. Table shows the outputs of each sub-process separately. Outputs of the Software System Development process are a follows:

Table 2. The outputs of the CompoMDD process model.

Lifecycle Stage	Output
Software System Development	
Requirements elicitation	User requirements
Project planning	Project plan
System analysis	Software requirements specification
System decomposition	Decomposition model
Component conceptual modeling	Component conceptual models (CCM)
Components searching and selection	Set of selected components
Adaptation	Adapted components
Components integration	Integrated system model
System implementation	Source code
System test and deployment	Deployed system
Component Development	
Model specification (only for reusable components)	Component platform independent model (CPIM)
Model implementation (only for reusable components)	Component platform specific model (CPSM) and source code
Model specification (for non-reusable components)	Design document
Model implementation (for non-reusable components)	Source code

User requirements are gathered during the requirements elicitation stage. This includes a requirements document that contains information about the desired application such as the problem, the customer's expectations and the boundaries of the system. After that, project plan is produced. A project plan is defined as "a formal, approved document used to guide both project execution and project control" [45]. A project plan includes scope, cost, and schedule of the project as well as assumptions, constrains and decisions.

During system analysis, more detailed software requirements specification is defined. This information represents a high level description of the system and does not contain any details about architecture or implementation. In system decomposition, the output should be a decomposition model that describes conceptual design and divides the system into components. Next, a number of conceptual models of components that are suggested by decomposition model will be developed in component conceptual modeling step.

If the searching process succeeds in finding components in the repository that match the required conceptual models then the output will be the selected components, otherwise the process will move to component development activities. These selected components are adapted in the adaptation step and the adapted components are included into the system. The output of the implementation stage is the source code of the system. As explained in previous sections an iterative process can be followed before moving to the next stage. Finally, after testing and deployment, final system is generated.

Outputs of the Component Development process are design models and source code for the new component. The outputs of the reusable component development are the CPIM in the model specification stage, and the CPSM and source code in the component implementation stage. These models are stored in the local repository. After the test stage, these models are stored in the main repository. The outputs of the non-reusable component development are the design document and source code of the component which are directly included into the system and not stored in the repository.

5.5. A Development Process Workflow

We proposed the CompoMDD process model which merges CBSE and MDD approaches. CompoMDD provides a high level methodology and has great promise to realize the goals of both approaches. In this section, we provide a sample workflow for our proposed model. This workflow illustrates how software development can be performed according to the CompoMDD. Figure 3 shows the sample workflow for the CompoMDD model. This workflow suggests a possible implementation of the proposed process model to illustrate its applicability whereas Figure 1 shows the overall process from a generic point of view. Diamond shapes represent Yes/No questions and rectangles represent the tasks as common in flowcharts.

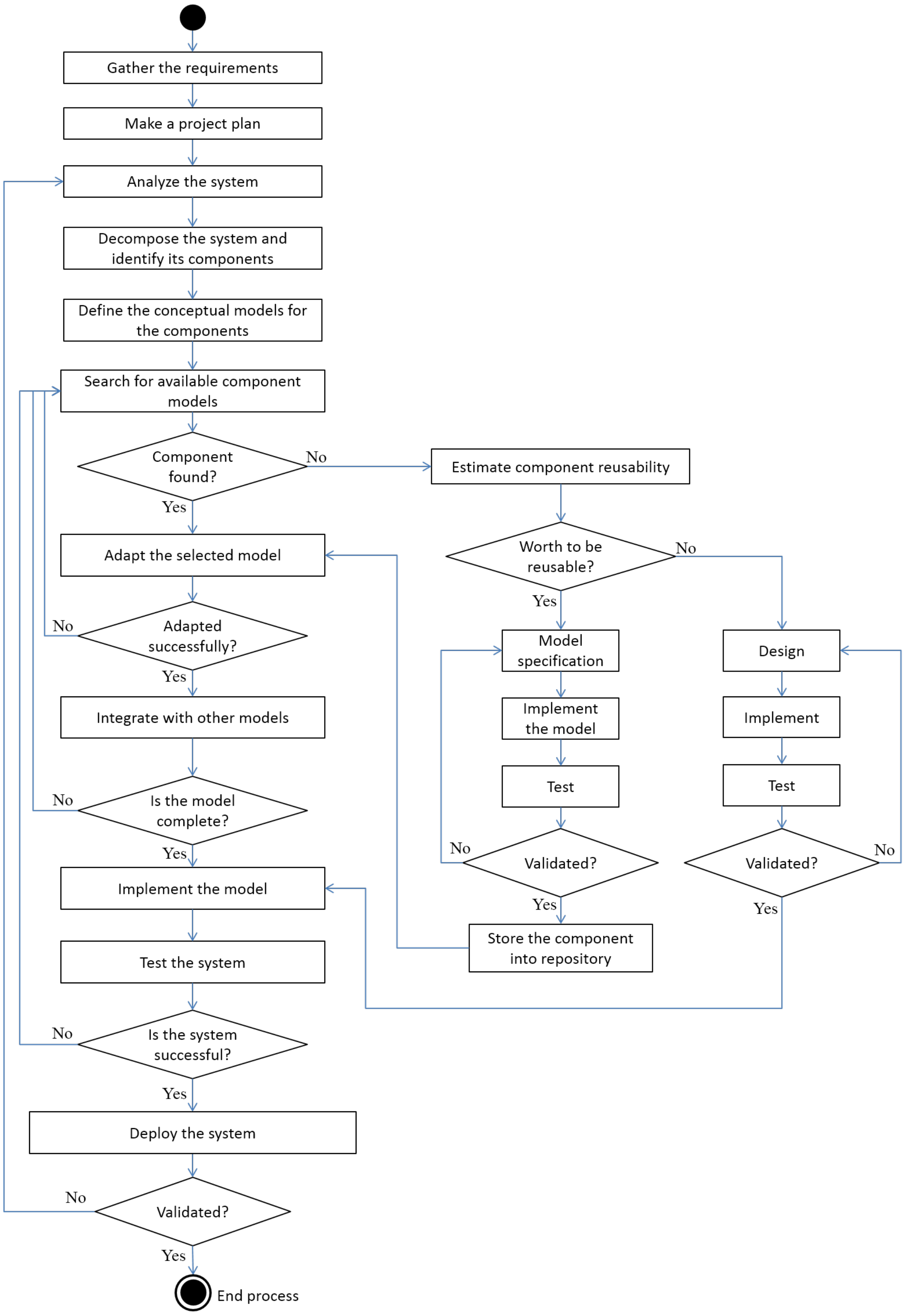


Figure 3. A sample workflow for the proposed process model.

6. Application to an Exemplar Case Study: An Online Quiz System

There are different ways that one can use to validate the proposed approach. This includes subject matter expert feedback, controlled experimentation validation, and also application to a case study, just to name a few. To validate the CompoMDD process model, we used it to develop an online quiz system as a working example. An online quiz system is a web application that allows users to take quizzes or tests online and provides an online learning experience. This example is chosen because it is a general subject and has potential to be reused for development of similar systems. A web based application is preferred to minimise platform related issues. The question types and categories could be different. For simplicity, we used text based questions in our study. Questions and answers are stored in a MySQL database and Apache is used as a web server. MAMP is used as an all-in-one solution.

6.1. Using CompoMDD Process Model

First of all, we identified a set of requirements and user scenarios to define the scope of the project. Then we analysed the system and the system was decomposed into its essential parts as shown in Figure 4 which represents the DM. The target users of the system are learners and specialists from different fields so the interfaces must be user friendly and easy to use. Anyone can use the system without registration to get a quiz. Users should register to add a question, but the system is free and publicly available.

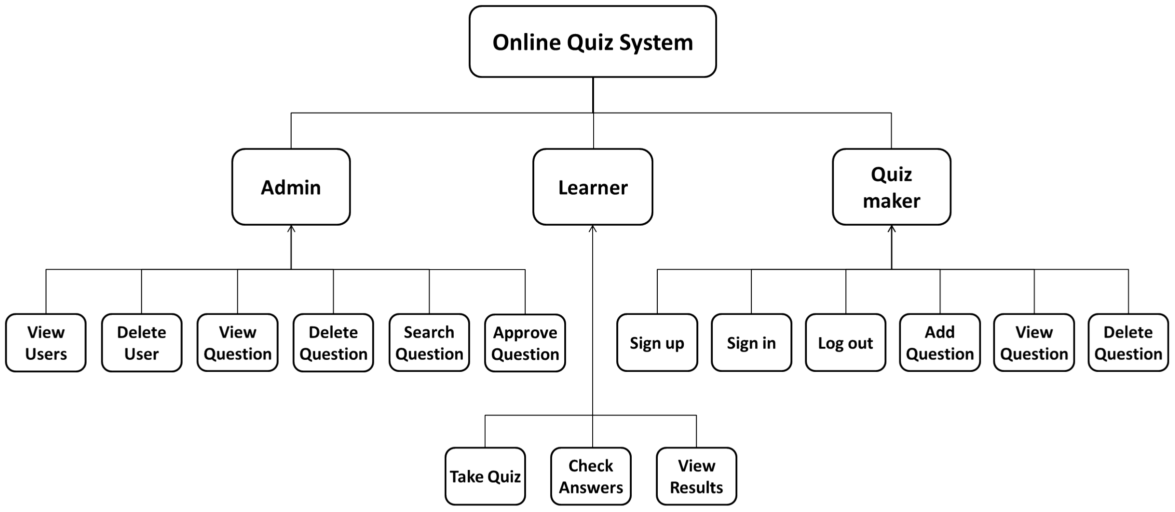
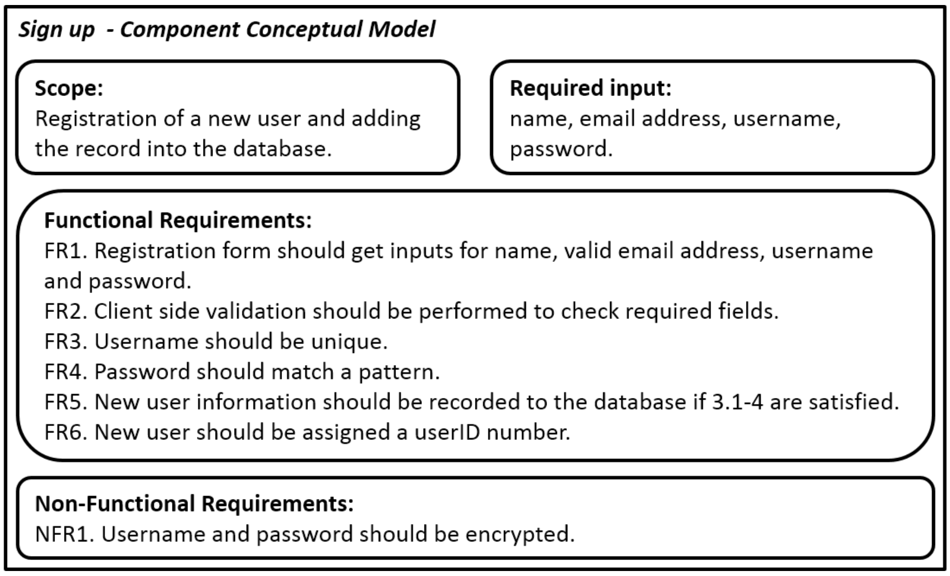


Figure 4. Decomposition of web-based online quiz system.

In this system, there are three main actors who are admin, quiz maker and learner. The quiz maker is the person who can add the questions and their answers to the database. The quiz maker should create an account, i.e., sign up, to be able to add a question and they have to sign in their account every time they want to add a new question or edit a question that they added before. Quiz makers can delete the questions that they added before. The learner is the person who can take a quiz without registration. Learners can check their answers and see their results. The admin is the person who checks the questions and the answers to ensure that they are correct and suitable. Admin can delete any users or any questions. All users can search for questions.

Depending on the potential reuses and the difference in complexity between creating a reusable component and a non-reusable component, we can assess whether the desired component is worth to be a reusable component. Sign up, sign in, add, view, search and delete components were assessed as reusable components. Other components like taking a quiz or viewing results were assessed to be non-reusable components. All components are developed from scratch according to the relevant component development lifecycle.

A conceptual model is an abstract representation of concepts and ideas to explain a system in early stages. It can be defined by using text, mathematical concepts, diagramming techniques, etc. In this case study, component conceptual models are defined with semi-structured text focusing on the requirements. Figure 5 shows an example for the sign up component.



Sign up - Component Conceptual Model

Scope:
Registration of a new user and adding the record into the database.

Required input:
name, email address, username, password.

Functional Requirements:
FR1. Registration form should get inputs for name, valid email address, username and password.
FR2. Client side validation should be performed to check required fields.
FR3. Username should be unique.
FR4. Password should match a pattern.
FR5. New user information should be recorded to the database if 3.1-4 are satisfied.
FR6. New user should be assigned a userID number.

Non-Functional Requirements:
NFR1. Username and password should be encrypted.

Figure 5. Sample CCM definition for sign up component.

6.2. Metamodeling for Component Platform Independent Modeling

Component models can be defined as a graph, mathematical equation, text or other types. In this study, we developed a metamodel for component platform independent modeling. Figure 6 shows the CPImeta metamodel for our online quiz system. This metamodel proposes a modeling language for component-based web application development. The metamodel is developed with GME (Generic Modeling Environment). The main modeling element is the component which can have inputs, outputs and parameters. There are also web pages which are HTML files that may or may not use components. Components are developed as PHP scripts. The connections between the components and web pages can be as follows: link, include or require. The representations for the connections are shown in Table 3.

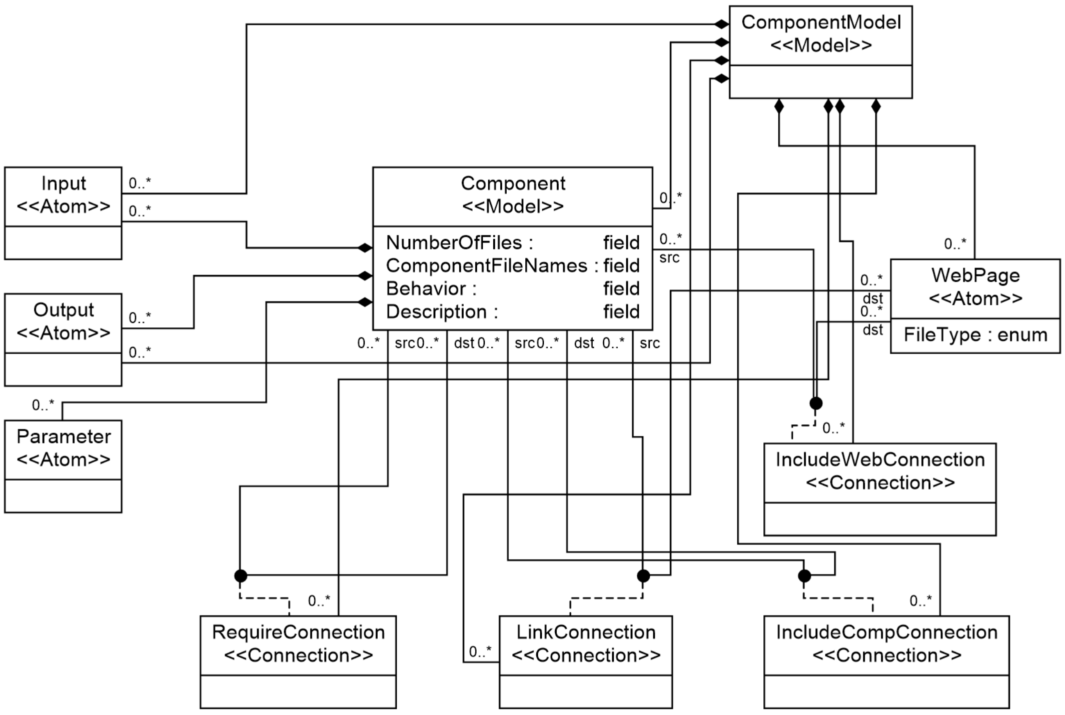


Figure 6. CPImeta: component platform independent metamodel.

Table 3. The connection types in CPImeta.

Connection Type	Representation	Description
Link connection	- - - - - - - - - ->	Link connection is used to refer the web page links.
Include connection	——————→	Include connection is used for including other elements as composition. Include connection can be two types as include web page or include component. But, they are represented same.
Require connection	◆——————	Require connection is used for embedding a component into another component.

CPIM models for the reusable components were generated according to CPImeta. Figure 7 shows the sign-up component model and Figure 8 shows the add component model as an example.

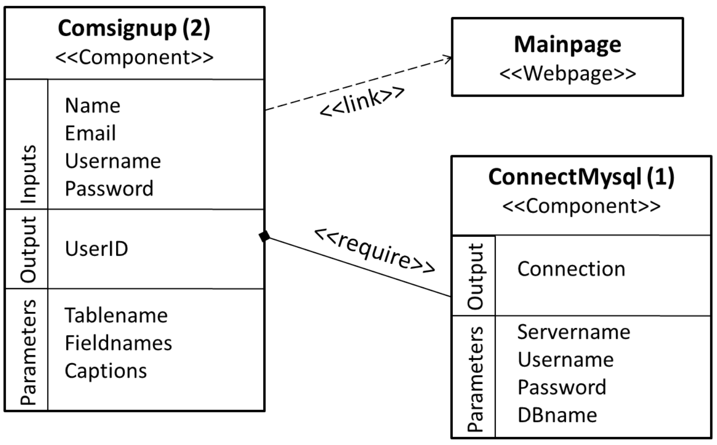


Figure 7. CPIM for sign up component.

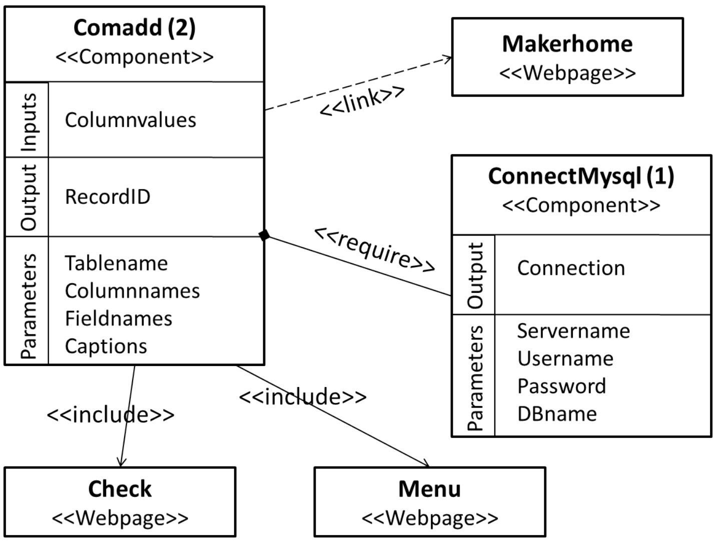
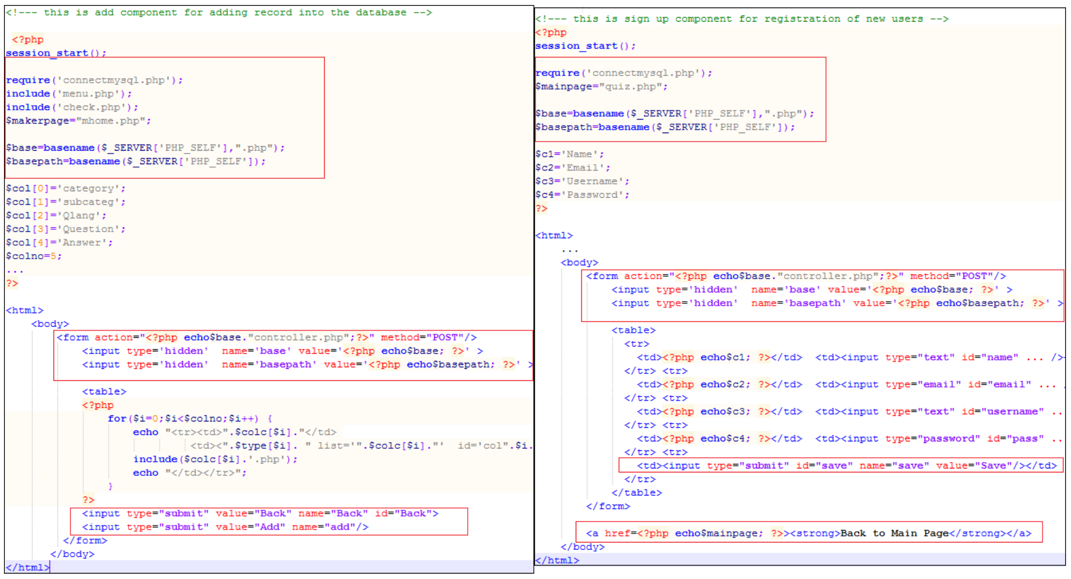


Figure 8. CPIM for add component.

All components were implemented using PHP and HTML. Hence, platform specific models include code snippets of these languages. Two examples of CPSMs (for the given examples in Figures 7 and 8) are shown in Figure 9 next to each other to show the similarities and emphasize the generated text. In this case study, platform specific models are defined as code templates. Code templates are generated from the CPIM models then they are completed. These models were integrated to construct the integrated system model that is shown in Figure 10.



```
<!--- this is add component for adding record into the database -->

 <?php
session_start();

require('connectmysql.php');
include('menu.php');
include('check.php');
$makerpage="mhome.php";

$base=basename($_SERVER['PHP_SELF'],".php");
$basepath=basename($_SERVER['PHP_SELF']);

$col[0]='category';
$col[1]='subcateg';
$col[2]='Qlang';
$col[3]='Question';
$col[4]='Answer';
$colno=5;
...
?>

<html>
    <body>
        <form action="<?php echo$base."controller.php";?>" method="POST"/>
            <input type='hidden'  name='base' value='<?php echo$base; ?>' >
            <input type='hidden'  name='basepath' value='<?php echo$basepath; ?>' >

            <table>
            <?php
                for($i=0;$i<$colno;$i++) {
                    echo "<tr><td>".$colc[$i]."</td>
                              <td><".$type[$i]. " list='".$colc[$i]."'  id='col".$i.
                    include($colc[$i].'.php');
                    echo "</td></tr>";
                }
            ?>
            <input type="submit" value="Back" name="Back" id="Back">
            <input type="submit" value="Add" name="add"/>
        </form>
      </body>
</html>
```

```
<!--- this is sign up component for registration of new users -->
<?php
session_start();

require('connectmysql.php');
$mainpage="quiz.php";

$base=basename($_SERVER['PHP_SELF'],".php");
$basepath=basename($_SERVER['PHP_SELF']);

$c1='Name';
$c2='Email';
$c3='Username';
$c4='Password';
?>

<html>
    ...
    <body>
        <form action="<?php echo$base."controller.php";?>" method="POST"/>
            <input type='hidden'  name='base' value='<?php echo$base; ?>' >
            <input type='hidden'  name='basepath' value='<?php echo$basepath; ?>' >

            <table>
              <tr>
                <td><?php echo$c1; ?></td>  <td><input type="text" id="name" ... />
              </tr> <tr>
                <td><?php echo$c2; ?></td>  <td><input type="email" id="email" ...
              </tr> <tr>
                <td><?php echo$c3; ?></td>  <td><input type="text" id="username" ..
              </tr> <tr>
                <td><?php echo$c4; ?></td>  <td><input type="password" id="pass" ..
              </tr> <tr>
                <td><input type="submit" id="save" name="save" value="Save"/></td>
              </tr>
            </table>
        </form>

        <a href=<?php echo$mainpage; ?>><strong>Back to Main Page</strong></a>
    </body>
</html>
```

Figure 9. Sample code templates for sign up and add components.

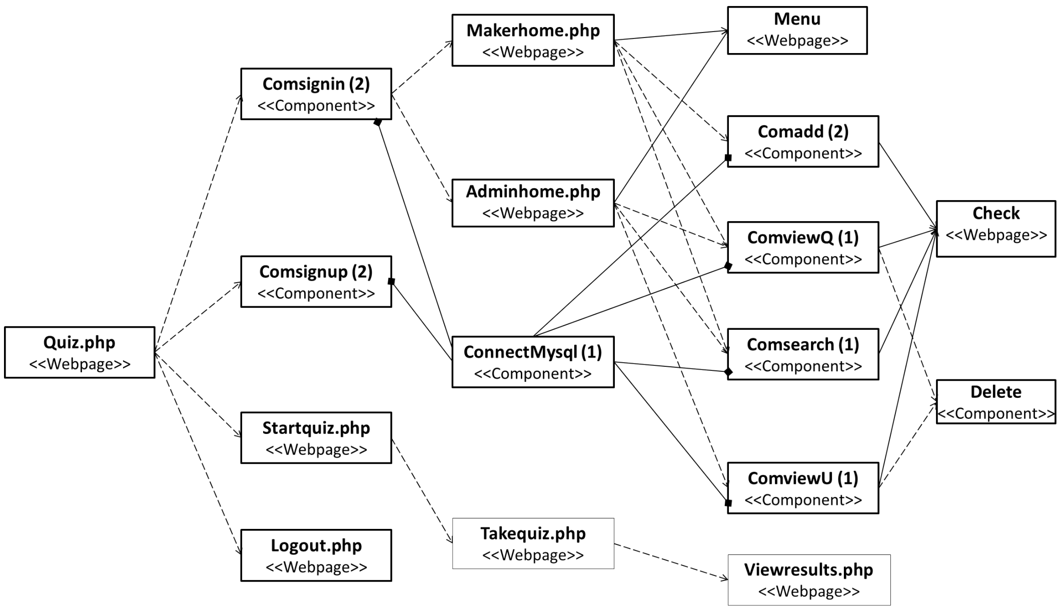
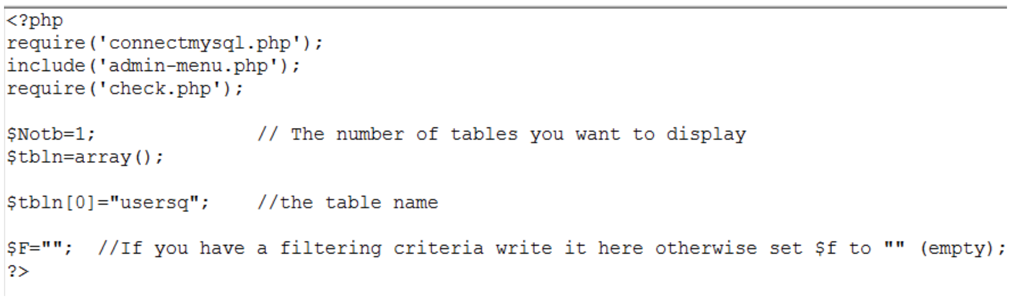


Figure 10. Integrated system model.

After the components are developed, we can adapt the components by modifying the parameters file, which can be considered as the component's interface. The parameters can be the names of linked pages or components, number of inputs, table name, column names or captions. We modify them to conform to the requirements.

We used direct link inclusion for the models such that each model is linked to its source code by a link. So, when a model is included in the design then its associated source code is included to the implementation. If we look at the metamodel given in Figure 6, each component has two attributes which show the number of files and the list of component file names. After component implementation, CPIM is updated to have the file names so that they can be included in the system integration and implementation. So, when a model is included in the design then its associated source code is included to the implementation. For example, for sign up component there are three files which are *signup.php* (main component file), *signupcontroller.php* (adding the logic) and *signupparams.php* (for setting the parameters). After testing, these are included in the repository.

Some components are used more than once in the system, such as view component, which was used in the view questions function and the view users function. As an example, Figure 11 shows the parameters file for the view users function while Figure 12 shows the parameters file for view questions function. Both of them are using the same component, which is view records, but by modifying the parameters file, we can use it for different tasks. Finally, to implement the system, we combine the code of reusable and non-reusable components to construct the final system and test it.



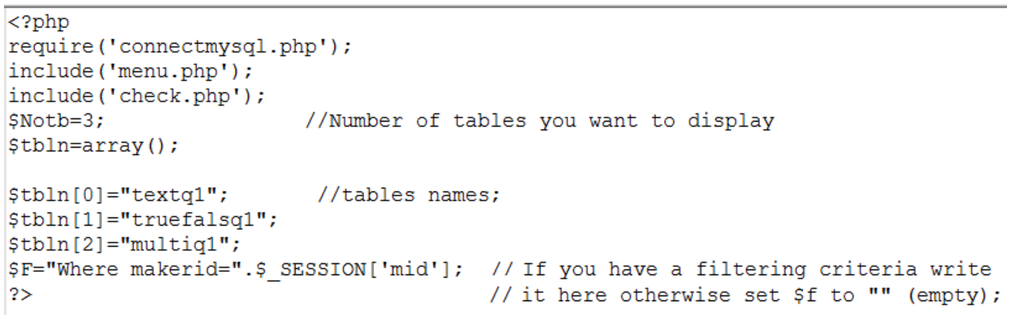
```
<?php
require('connectmysql.php');
include('admin-menu.php');
require('check.php');

$Notb=1;              // The number of tables you want to display
$tbln=array();

$tbln[0]="usersq";    //the table name

$F="";  //If you have a filtering criteria write it here otherwise set $f to "" (empty);
?>
```

Figure 11. The parameters file for view users component.



```
<?php
require('connectmysql.php');
include('menu.php');
include('check.php');
$Notb=3;                //Number of tables you want to display
$tbln=array();

$tbln[0]="textq1";       //tables names;
$tbln[1]="truefalsq1";
$tbln[2]="multiq1";
$F="Where makerid=".$_SESSION['mid'];  // If you have a filtering criteria write
?>                                     // it here otherwise set $f to "" (empty);
```

Figure 12. The parameters file for view questions component.

6.3. Discussion

Although there are different ways to validate the proposed approach, we decided to apply the CompoMDD process model to an exemplar case study, i.e., online quiz systems, in order to illustrate the applicability of the model through a working example. CompoMDD process model presents a practical lifecycle for component based software development which also brings the interim models at different levels of abstraction together. The goal of the validation study was to show the applicability of the proposed model and testing it with a real life example. The validation study is limited to the selected example. We will involve potential users to improve our evaluation. Hence, future validation studies will utilise controlled experiments and expert feedback in industrial settings.

In this study, we evaluated our work with expert opinion and we set up small-scale experiments with mid-level experienced web developers. The most challenging aspect during the experiments was understanding the different modeling and metamodeling layers. We needed to provide a training of the CPIMMeta tool to the users. Developers said that they needed to change their way of thinking to understand metamodeling at different stages. Both experts and developers agreed that the proposed approach was effectively applied in the case study, code is well-structured and reusable components can be used while developing a similar web application. The proposed CompoMDD model supports the following common features shared by existing component based software development approaches (see Table 1 to compare):

- Components can be deployed independently,

- Components can be composed with other components,
- Components can interact with other components,
- There is a separation between component development and system development activities in the lifecycle,
- There is an adaptation step that allows modifying the components,
- Components are verified and validated before they are stored into the repository,
- Domain specific modeling is used to define the metamodels.

In addition, CompoMDD brings the following benefits:

- It embeds the model-driven software development approach into the component based software development lifecycle,
- It introduces a step to consider the potential reuse of components before developing them,
- It facilitates component and system integration by using interim models at different levels.

Although the results were supporting our hypothesis more research should be done for evaluating the applicability at large scale. Moreover, more experiments with real-life scenarios are needed to support our research. Our evaluation did not include the productivity aspect due to there is already evidence in the literature about it which are discussed in Section 3.

7. Conclusions and Future Work

This paper presents CompoMDD, a software development process model that merges CBSE and MDD as a solution to the problems of the development of software systems and increases the productivity. Our proposed model splits software lifecycle into two processes, the system development process and the component development process. MDD is embedded in both development processes. Moreover, the model reduces unnecessary cost of developing non-reusable components with reusability concerns (although it does not have reuse potential) by merging CBSE with a traditional software development approach. A component can be developed as a reusable component using the component-based approach or as a non-reusable component using a traditional approach depending on reusability assessment result.

The CompoMDD process model was used to develop an online quiz system to evaluate its applicability and practicality. The study showed that CompoMDD is applicable and easy to follow. Using CBSE approach to develop a new system is justifiable if the cost of reusing pre-build components is less than the cost of developing new components otherwise it is better to develop the system from scratch or using another software engineering approach. In our case, CBSE was beneficial since the effort needed to develop reusable components was less than the reduced effort in design and implementation stages. However, more tests needed to measure the results in different application scenarios. CompoMDD introduces interim models and embeds MDD approach into the development process as compared to other existing CBSE process models. In this way, it has more potential for model and component reusability.

This study presents an applicable and effective software development process model, and shows that the interplay of software components and model-driven approaches is possible and beneficial. As a future work, we will work on designing and implementing an online component repository compatible with CompoMDD process model. We think that interoperability is still a challenge in CBSE and so we will focus on formal definition of models and facilitating MDD methods to overcome interoperability issues.

In addition, we would like to improve the system validation step, and analyse the traceability between the requirements and the components. In this way, CompoMDD model can better contribute to change management in software development projects. Existing modeling languages such as UML, SysML, Web Modeling Language (WebML) [46], UML-based Web Engineering (UWE) [47], etc., can be used during the early stages of the development process and model continuity can be achieved throughout the software development lifecycle. Hence, future validation studies will cover and monitor project management activities during the experiments. Finally, as discussed in the previous section, more validation experiments will be set up to evaluate the proposed model at larger scale and in industrial settings.

Author Contributions: This work was done mostly during the post graduate studies of the first author (A.U.A.) under the supervision of the second author (D.C.). The paper has been improved with the third and fourth authors' review and contributions. A.U.A. did the research. D.C. provided supervision and guidance as well as reviewed the paper. A.U.A. and D.C. have written the manuscript. G.L. and H.D. contributed to the outline and hence coherence of the paper. G.L. and H.D. also reviewed the paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to thank the Software Engineering Department at Atılım University for supporting this work during the Master of Science studies of the first author.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CBSE	Component-Based Software Engineering
MDD	Model-Driven Software Development
CompoMDD	Proposed Component-Based Model-Driven Software Development Process Model
MDA	Model Driven Architecture
DM	System Decomposition Model
CCM	Component Conceptual Model
SDM	System Design Model
CPIM	Component Platform Independent Model
CPSM	Component Platform Specific Model
CCmeta	Component conceptual metamodel
CPImeta	Component platform independent metamodel
CPSmeta	Component platform specific metamodel

References

1. Szyperski, C. *Component Software: Beyond Object-Oriented Programming*; ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 1998.
2. Sommerville, I. *Software Engineering*, 10th ed.; Pearson: London, UK, 2016.
3. Emmerich, W.; Kaveh, N. Component Technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA Component Model. In Proceedings of the 24th International Conference on Software Engineering, ICSE '02, Orlando, FL, USA, 25 May 2002; ACM: New York, NY, USA, 2002; pp. 691–692. [CrossRef]
4. Khemakhem, S.; Drira, K.; Jmaiel, M. Chapter 8: Description, classification and discovery approaches for software components: A comparative study. In *Modern Software Engineering Concepts and Practices: Advanced Approaches*; Information Science Reference: Warszawy, Poland, 2010; pp. 196–219. [CrossRef]
5. Crnkovic, I.; Chaudron, M.; Larsson, S. Component-Based Development Process and Component Lifecycle. In Proceedings of the International Conference on Software Engineering Advances, Tahiti, France, 29 October–3 November 2006; pp. 44–60. [CrossRef]
6. Bakshi, A.; Singh, R. Component Based Development in Software Engineering. *Int. J. Recent Technol. Eng. (IJRTE)* **2013**, *2*, 48–52.
7. Chatterjee, R.; Rathi, H. A prolific approach towards automating component repository search. In Proceedings of the Seventh International Conference on Contemporary Computing (IC3), Noida, India, 7–9 August 2014; pp. 547–552.

8. Baker, P.; Harman, M.; Steinhofel, K.; Skaliotis, A. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In Proceedings of the 22nd IEEE International Conference on Software Maintenance, ICSM '06, Chicago, IL, USA, 11–14 September 2006; IEEE Computer Society: Washington, DC, USA, 2006; pp. 176–185. [CrossRef]
9. Becker, S.; Brogi, A.; Gorton, I.; Overhage, S.; Romanovsky, A.; Tivoli, M. Towards an Engineering Approach to Component Adaptation. In *Proceedings of the International Conference on Architecting Systems with Trustworthy Components*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 193–215.
10. Brogi, A.; Canal, C.; Pimentel, E. Measuring Component Adaptation. In *Coordination Models and Languages*; De Nicola, R., Ferrari, G.L., Meredith, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; pp. 71–86.
11. Çetinkaya, D.; Verbraeck, A.; Seck, M.D. Model Continuity in Discrete Event Simulation: A Framework for Model-Driven Development of Simulation Models. *ACM Trans. Model. Comput. Simul.* **2015**, *25*, 17:1–17:24. [CrossRef]
12. Syriani, E.; Gray, J.; Vangheluwe, H. Domain Engineering: Product Lines, Languages, and Conceptual Models. In *Domain Engineering: Product Lines, Languages, and Conceptual Models*; Chapter Modeling a Model Transformation Language; Springer: Berlin/Heidelberg, Germany, 2013; pp. 211–237.[CrossRef]
13. Badampudi, D.; Wohlin, C.; Petersen, K. Software component decision-making: In-house, OSS, COTS or outsourcing—A systematic literature review. *J. Syst. Softw.* **2016**, *121*, 105–124. [CrossRef]
14. Alfraihi, H.; Lano, K. The Integration of Agile Development and Model Driven Development—A Systematic Literature Review. In Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development, Porto, Portugal, 19–21 February 2017; INSTICC, SciTePress: Setúbal, Portugal, 2017; pp. 451–458. [CrossRef]
15. Alrubaee, A.U. A Component Based Model Driven Software Development Framework for Web-Based Applications. Master's Thesis, Software Engineering MSc Program, Graduate School of Natural and Applied Sciences, Atilim University, Ankara, Turkey, 2017.
16. Brambilla, M.; Cabot, J.; Wimmer, M. *Model-Driven Software Engineering in Practice*, 2nd ed.; Morgan & Claypool Publishers: Mountain View, CA, USA, 2017.
17. Brown, A.W.; Conallen, J.; Tropeano, D. Introduction: Models, Modeling, and Model-Driven Architecture (MDA). In *Model-Driven Software Development*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–16. [CrossRef]
18. Capretz, L. Y: A new component-based software life cycle model. *J. Comput. Sci. Sci. Publ.* **2005**, *1*, 76–82. [CrossRef]
19. Lau, K.K.; Taweel, F.M.; Tran, C.M. The W Model for Component-Based Software Development. In Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications, Oulu, Finland, 30 August–2 September 2011; pp. 47–50.
20. Tomar, P.; Gill, N.S. Verification amp; Validation of components with new X Component-Based Model. In Proceedings of the 2010 2nd International Conference on Software Technology and Engineering, San Juan, PR, USA, 3–5 October 2010; Volume 2, pp. V2-365–V2-371.
21. Vale, T.; Crnkovic, I.; de Almeida, E.S.; da Mota Silveira Neto, P.A.; Cavalcanti, Y.C.; de Lemos Meira, S.R. Twenty-eight years of component-based software engineering. *J. Syst. Softw.* **2016**, *111*, 128 – 148. [CrossRef]
22. Rodrigues da Silva, A. Model-driven Engineering. *Comput. Lang. Syst. Struct.* **2015**, *43*, 139–155. [CrossRef]
23. Kapteijns, T.; Jansen, S.; Brinkkemper, S.; Houet, H.; Barendse, R. A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. In Proceedings of the 4th European Workshop on from Code Centric to Model Centric Software Engineering: Practices, Implications and ROI, Enschede, The Netherlands, 24 June 2009.
24. Bucchiarone, A.; Cabot, J.; Paige, R.F.; Pierantonio, A. Grand challenges in model-driven engineering: An analysis of the state of the research. *Softw. Syst. Model.* **2020**, *19*, 5–13. [CrossRef]
25. Mussbacher, G.; Amyot, D.; Breu, R.; Bruel, J.M.; Cheng, B.H.C.; Collet, P.; Combemale, B.; France, R.B.; Heldal, R.; Hill, J.; et al. The Relevance of Model-Driven Engineering Thirty Years from Now. In *Model-Driven Engineering Languages and Systems*; Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 183–200.

26. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. Model-Driven Distributed Simulation Engineering. In Proceedings of the 2019 Winter Simulation Conference (WSC), National Harbor, MD, USA, 8–11 December 2019; pp. 75–89.
27. Bocciarelli, P.; D'Ambrogio, A.; Falcone, A.; Garro, A.; Giglio, A. A model-driven approach to enable the simulation of complex systems on distributed architectures. *SIMULATION* **2019**, *95*, 1185–1211. [CrossRef]
28. Atkinson, C.; Paech, B.; Reinhold, J.; Sander, T. Developing and applying component-based model-driven architectures in KobrA. In Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, Seattle, WA, USA, 4–7 September 2001; pp. 212–223.
29. Weiss, K.A.; Ong, E.C.; Leveson, N.G. Reusable specification components for model-driven development. In Proceedings of the International Conference on System Engineering (INCOSE'03), Portland, OR, USA, 3–10 May 2003.
30. Phung-Khac, A.; Beugnard, A.; Gilliot, J.M.; Segarra, M.T. Model-driven Development of Component-based Adaptive Distributed Applications. In *Proceedings of the 2008 ACM Symposium on Applied Computing*; ACM: New York, NY, USA, 2008; pp. 2186–2191. [CrossRef]
31. Kainz, G.; Buckl, C.; Sommer, S.; Knoll, A. Model-to-Metamodel Transformation for the Development of Component-Based Systems. In *Model Driven Engineering Languages and Systems*; Petriu, D.C., Rouquette, N., Haugen, Ø., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 391–405.
32. Liu, Z.; Morisset, C.; Stolz, V. rCOS: Theory and Tool for Component-Based Model Driven Development. In *Fundamentals of Software Engineering*; Arbab, F., Sirjani, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 62–80.
33. Clemente, P.J.; Hernández, J.; Conejero, J.M.; Ortiz, G. Managing crosscutting concerns in component based systems using a model driven development approach. *J. Syst. Softw.* **2011**, *84*, 1032–1053. [CrossRef]
34. Kathayat, S.B.; Le, H.N.; Bræk, R. A Model-Driven Framework for Component-Based Development. In *SDL 2011: Integrating System and Software Modeling*; Ober, I., Ober, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 154–167.
35. Agustin, J.L.H.; del Barco, P.C. A model-driven approach to develop high performance web applications. *J. Syst. Softw.* **2013**, *86*, 3013–3023. [CrossRef]
36. Mizuno, T.; Matsumoto, K.; Mori, N. Applying Component-Based Technologies to Model-Driven Software Development. *Electron. Commun. Jpn.* **2015**, *98*, 24–31. [CrossRef]
37. Ciccozzi, F.; Carlson, J.; Pelliccione, P.; Tivoli, M. Editorial to theme issue on model-driven engineering of component-based software systems. *Softw. Syst. Model.* **2017**. [CrossRef]
38. Allen, R.; Garlan, D. A Formal Basis for Architectural Connection. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **1997**, *6*, 213–249. [CrossRef]
39. Javed, A.Z.; Strooper, P.; Watson, G. Automated Generation of Test Cases Using Model-Driven Architecture. In Proceedings of the Workshop on Automation of Software Test (AST'07), Minneapolis, MN, USA, 26 May 2007. [CrossRef]
40. Shiva, S.G.; Shala, L.A. Software Reuse: Research and Practice. In Proceedings of the 4th International Conference on Information Technology, Las Vegas, NV, USA, 2–4 April 2007; pp. 603–609.
41. Sametinger, J. *Software Engineering with Reusable Components*; Springer: Berlin/Heidelberg, Germany, 1997.
42. Desouza, K.C.; Awazu, Y.; Tiwana, A. Four Dynamics for Bringing Use Back into Software Reuse. *Commun. ACM* **2006**, *49*, 96–100. [CrossRef]
43. Councill, B.; Heineman, G.T. Definition of a Software Component and Its Elements. In *Component-Based Software Engineering*; Heineman, G.T., Councill, W.T., Eds.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 2001; pp. 5–19.
44. Emerson, M.; Neema, S.; Sztipanovits, J. Metamodeling Languages and Metaprogrammable Tools. In *Handbook of Real-Time and Embedded Systems*; Lee, I., Leung, J.Y.T., Son, S.H., Eds.; Taylor & Francis Group: Abingdon, UK, 2008; Chapter 33, pp. 1–16.
45. Institute, P.M. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*, 6th ed.; Project Management Institute, Inc.: Newtown Square, PA, USA, 2017.

46. Acerbis, R.; Bongio, A.; Brambilla, M.; Butti, S.; Ceri, S.; Fraternali, P. Web Applications Design and Development with WebML and WebRatio 5.0. In *Objects, Components, Models and Patterns*; Paige, R.F., Meyer, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 392–411.
47. Kroiss, C.; Koch, N.; Knapp, A. UWE4JSF: A Model-Driven Generation Approach for Web Applications. In *Web Engineering*; Gaedke, M., Grossniklaus, M., Díaz, O., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 493–496.