The Usage of Models in Model-Driven Software Engineering

Anissa D. Donkers
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.d.donkers@student.utwente.nl

ABSTRACT

During the Software Systems module of the Bachelor of Computer Science at the University of Twente, students learn how to produce code based on Model-driven Software Engineering (MDE). They are familiarized with different kind of Unified Modelling Language (UML) diagrams to model software systems. The topics that are taught are part of the knowledge base of MDE.
This research looks into the recognition of diagram types and which design models are actually used by practitioners. The result of this research could be used for an improvement to the Computer Science MDE curriculum, but also for other technical studies at the University of Twente (UT) where students learn how to develop a software system. The curricula could be adapted so that they are more fitting to what is used by practitioners in the field of software engineering. From the results it is concluded that practitioners prefer to use self designed syntax over the UML syntax. Generally, the use of design is not prominent. However, when there are diagrams, they mostly present the architecture of the system.

Keywords

Modelling Formalism, Model-Driven Software Engineering, Design Usage, Education

1. INTRODUCTION

Model-Driven Software Engineering (MDE) is a development methodology that focuses on creating and exploiting domain models. This methodology is taught in all computer science curricula, including the Bachelor and Master degrees. A core set of concepts and practices (i.e. a Body of Knowledge) was proposed during the MODELS 2018 Educators' Symposium [10], based on the experiences of people who are researchers in this domain. The main subjects for the curriculum were determined and the advice was that these concepts should be taught in the Software Engineering courses [9].

MDE courses teach how to formulate a design of a software system in a model with well-defined semantics. There are a lot of different graphical modelling languages. Each language in turn has different types of models or diagrams. For instance, the Unified Modelling Language (UML) has 14 different types of diagrams such as class and activity diagrams [8]

Although there is a great variety of design models which are accepted in the field, only a small portion is taught in a curriculum. Optimally, it would be good to have this part to be as relevant as possible for the practice. By doing so, graduated students are well prepared for working in the MDE field.

35th Twente Student Conference on IT July 2nd, 2021, Enschede, The Netherlands.

1.1 Research question

The goal of this paper is to investigate what design models are used in practice. In particular, the following questions will be investigated:

1. What are the inclusion and exclusion criteria for the contributions and their artefacts?
2. What are the characteristic components of a design model and how can these models be categorised?
3. Which design model components are used in practice and by whom?

This paper presents literature research and observation research. First, a schema was created for analysing the data-set and especially for the recognition of modelling types of design diagrams. Thereafter, the work of graduate students was analysed. For each present software design diagram, the components were noted and the diagram received a general type as a label. Next, published work from a software conference was analysed. This large data-set was filtered down by the criteria which were defined with the first research question.

The structure of the paper is as follows; This paper will first go through the first two research questions. After these questions are answered, a classification schema can be given. This schema will contribute to the recognition of the diagram type.
Next, the outcome of the observation is discussed per data-set as described before. The paper will be concluded with points of discussion, a recommendation for the curriculum for software engineering courses at the University of Twente and future work.

2. BACKGROUND

The module Software Systems of the Bachelor of Computer Science offers a good start for novice programmers on how to design and document systems using Software Engineering models [2]. Various components are introduced during this course and this knowledge can be extended during the

Master of Computer Science. A body of knowledge for MDE has been assembled by several researchers [10, 9, 13]. The body of knowledge specifies topics that are relevant for bachelor and master studies in the area of software engineering. The topics are very diverse since there is a lot to know about MDE.

To go further into depth about models and design, there are a lot of different types of models. Even only considering UML, not every diagram is as well known as the others.
In previous research, Reggio et al. [14] did a survey about the use of UML diagram types, to find out which diagrams were known and which were used. One of the outcomes of this survey was that several UML diagrams were known by approximately less than half of the participants (namely timing and interaction overview diagrams) and most had been produced by fewer than half of the industry participants. Academic participants in general knew and produced more of the diagram types. Additionally, another outcome was that the usage of the given diagrams was lower than the knowledge of these diagrams. So even when the developer is familiar with certain diagrams, they do not use them in their work.

In the research of Reggio, he collected the usage of design models from the developer's own point of view. Also, the research was focused on the knowledge and usage of UML. However, what a practitioner uses and thinks that they use could differ. Therefore, this research observed what practitioners actually use for the modelling of their software system. This research not only looks for UML diagrams in the work of practitioners but also looks for the general design diagrams and their layout and components. This gives more insight in software modelling than when only looking at

3. INCLUDING AND EXCLUDING CRITERIA

This section will answer the first research question of this paper. The question was formulated to set the restrictions on the data-set which is used for this research. On the ground of generalizability and representativeness not being the main concern for this research, the data-set has to fulfil certain requirements.
The answer to the questions was found by analysing the environment of the problem and by giving a definition to practitioners.

Since the term *practitioners* gives a broad result, a boundary has been set. The focus lies on which practitioners and their work are interesting for this research.
For this research, practitioners are limited to students from Technical Computer Science at the UT that have built a model-driven software system, and engineers that have built a model-driven software system which is presented at a software engineering conference. The motivation for these choices can be found below.

3.1 Venue

This research was motivated by the University of Twente. This means that there were some requirements regarding the data, keeping in mind that the result should be accurate for the University. For this reason, the research focused on practitioners' work within the Netherlands. This was done so that the UT can alter the curricula based on local experiences. Therefore, the criteria that the data should originate from a local source was determined. Local is defined as practitioners from the Netherlands or their work should be from a source where it would be likely that a Dutch developer would publish their work.

Interesting factors of this research are the use of modelling diagrams among students and software developers outside of the university. First of all, the knowledge base about model-driven software engineering of students from the UT is known. This gives an informative insight into what the students prefer to use. Additionally, practitioners outside of the university give a good insight into what is preferred in the field. Furthermore, it is good to reflect the results of the students on a larger scale. That is why for this research, the focus lies on students and developers in the field of software development.

In short, the source from where the data was received should be of interest for University of Twente to keep the research as relevant as possible. Also, it was determined that the data that was analysed must be from students of the UT or software developers.
Possible venues(i.e. sources) where the data could be collected from are named below;

- *A source where students' work is published.*
  This is for the reason that the knowledge of the student is predetermined by following the curriculum. This gives an interesting comparison to what is actually used.

- *A source where University lecturers' work is published*
  These employees of the university teach the students how to design software systems with design diagrams. It is interesting to know what they themselves use for their work. This also is a broad sample group since these teachers have different backgrounds.

- *Information-technology (IT) conferences*
  When looking at the participants of a general IT conference the backgrounds of these participants could also vary just like the background of the employees of the UT. Furthermore, this data-set could be quite large, which is preferable for this research.
  Another possibility is to look only at Software Engineering conferences. These are more specific and have a bigger probability of showing designs related to a software system.

Examples of venues are the EEMCS - Bachelor Showcase [1] for analysing the work of students and the research information web page from the University of Twente [6] for analysing published work of UT lecturers. There are several IT conferences, one of them is ICT Open [4]. Examples of conferences related to only software engineering are the International Conference on Software Engineering [7] and the Intelligent tutoring systems International Conference [12].

In the end, the decision has been made to observe 2 venues due to the scope of the paper. These were the Bachelor Showcase and the International Conference on Software Engineering 2020. The choice for these venues has been made because they show two extremes. Normally students and participants of such conference would not influence each others work.

3.2 Contributions

The venues, which were discussed previously, consist of contributions from practitioners. The next phase is to filter these contributions so that only the relevant papers for this

research would be analysed. Therefore, from the available data at a certain venue, only a sample was considered. The sample was selected based on the following predetermined criteria.

- The paper represents a software system.
- The paper should represent a self-made system such that the data that was gathered are developed by practitioners.
- In order to have a relevant result, the contribution should be no older than 5 years. This is because the field of Software Engineering and IT in general evolves quickly.
- The paper includes one or more software design diagrams

3.3 Artefacts

For this research the artefacts are the presented diagrams in a contribution that fulfilled the requirements as described above.

When looking at a diagram, there were several requirements that needed to be fulfilled to be considered for this research. A diagram had to be directly related to the software system of the contribution. It had to give a visual representation of a process within the software, or of the user-system interaction or of the architectural design of the software.
It was not relevant for this research when the diagram showed a workflow of the development (e.g. scrum diagram), a graphical user interface, the working of a different system than presented or any other diagram which is not related to the software system and its procedures. Also when the diagram showed either the in and/or output of the system, the diagram was considered meaningless for the research. A short overview of the artefact criteria is given below.

- The diagram must be related to a software system.
- The diagram shows a process of the system, user-system interactions or the architectural design.
- The diagram must not show any development related process.
- The diagram must not show a GUI or general input or output generated for/by the system.

The diagrams can be presented in a report or paper, these diagrams were not considered as a communication tool to the outside world but only as part of the development process. These diagrams were part of the research but will not be referred to as a communication tool.
The diagrams that are present on a poster or any other presentation were considered a communication tool. A presentation must clarify the system to outsiders. These people are not involved in the development process, but they are interested in the software system.

4. DIAGRAM CHARACTERISTICS

The goal of this part is to compose an overview of characteristic elements of design diagrams. When looking at a diagram, interpretation is the key. A design model will represent only a simplified version of the system. This means that the diagram should be understandable without any extra information. Just to give an example, Figure 1 is a diagram of the MOOColab system [11]. This system is not related to the project, its purpose is illustrative only. This image is a nice example of what can be found when looking at design diagrams of software systems. These kind of diagrams are supposed to give an overview of the system or its design. Later, an analysis is given on this diagram.

UML [5] has split up the overall design models into 3 different modelling types, namely Supplemental, Behavioural and Structural Modelling.
According to the UML documentation, a diagram that represents supplemental modelling will contain use cases or deployments that are connected with information flows. These deployments specify components that can be used to define the system architecture and assignment of software artefacts to system elements.
Behavioural Modelling shows what possible outcomes a system has. The diagrams can contain activities, common behaviour, interaction and statuses. These elements are linked with a triggering event or in other words a control flow.
Lastly, the documentation states that the Structural Modelling diagrams can contain system values, classifiers but also packages or the common structure. These components of the system are linked with associations.

With the schema that has been created for this research, the UML diagram specification is covered. A more detailed description of the schema with a definition of each element and the possible value of the outcome of the analysis is given below.

Software system
When the diagram shows important information about the software system, it will be denoted as “yes”. Otherwise, it can be described as workflow when the development process is represented. When it can neither be classified as software system or workflow the diagram is labelled as 'other'. When the value is other than yes, the rest of the elements are not considered during the observation.
Value: yes, workflow or other

Diagram type
When looking at elements of the diagram together with the given description, the type is determined. The diagram could be a representation of a process within the software or other design-related matter or the architecture of the system. These types do not yet correlate with the UML Modelling types as defined before.
Value: Design or architecture.

UML
When the diagram is a recognisable UML diagram, the type is given and the components do not need to be filled in. When this is not the case, the field will be left blank.
Value: Class diagram, use case diagram, etc or blank.

Components
Here, the focus is on the layout of a diagram. This will mostly be built out of blocks with connections. These blocks can describe software components or events/actions, but also a status of an object. In the case of an element that is anomalous, it is also mentioned here.
Value: (sub-)components, status, activities (or similar)

Connections
The connections between the components of a diagram can illustrate different communications. A link can show the data flow of the system. It can also show the triggering event to go from the first state to the second. This will be

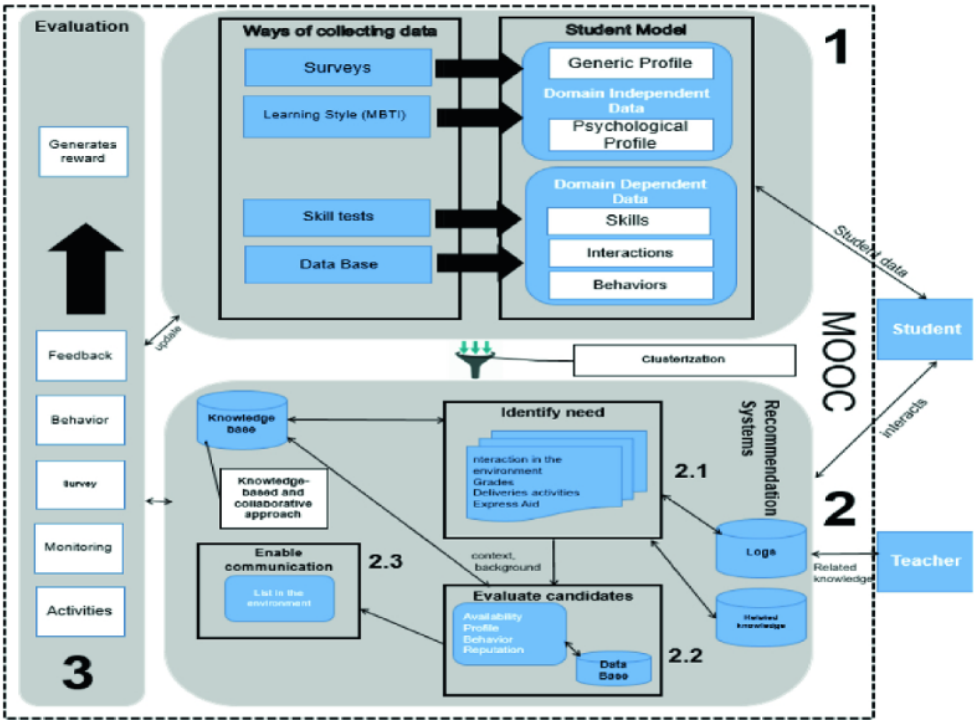

Figure 1. Example design model from MOOColab

named a control flow. A connection between components can also illustrate an association. Every connection which is not a data or control flow will be labelled as an association. Examples that will be considered as associations are aggregations, compositions and dependencies.
Value: Control flow, data flow, association

Layout
A diagram can have significant details in the layout which makes it less complicated to identify the type of the diagram. These details can vary but examples of these details are a sequence pattern or layers.
Value: sequence, layers, etc.

Completeness
Some diagrams are relatively big. This makes it hard to refer to and for the reader to read. That is why fragments are used.
Value: Complete, fragment, etc.

Location
It is noted down whether the diagram is found in a presentation or in the report/paper of the software system.Therefore the distinction is made whether the diagram was used as a communication tool or as part of the development process.
Value: Poster, presentation or report

These last 2 elements are only relevant for the research but not for identifying diagrams. The process of analysing with the help of this schema is presented in Figure 2. The first blocks make sure that the diagram represents a software system. When the diagram shows a workflow or something different, the analysis will not continue. In the next green section, more general features of the diagram are denoted. It ends with the question whether it is a UML diagram or not. When it is a UML diagram, it is noted and the analysis is already complete. Otherwise, the analysis will go into more detail. The blue section shows a brief explanation of the observation methodology for the diagrams which are not a UML.

4.1 Example observation

Figure 1 is an example of a design model which can be found during the observation part of this research. When looking at this diagram, all the aspects of the diagram characteristics, as defined above, are analysed. Below, an example of the methodology of the research is given with this diagram.

From the context of the diagram in the original paper, and by looking at the structure, it can be concluded that this diagram represents a software system.
The next remarkable segment of this diagram is the boxes. The smaller blue ones represent system components. Segments of a database are also represented on this level. But around these components, there are black lined boxes but also grey fields. So this diagram shows components and within areas, there are sub-components.
Additionally, segments are numbered. Numbering in general can indicate a sequence or an hierarchy. In this diagram this is not the case. The numbers are used to label the components and it sub-components.
On the right side, there are 2 actors stated. These are also connected to the system. These actors can perform an activity that influences the system or the system has specific functionality for a user. In this diagram, it is shown that the user can give a certain input and that the system keeps track of the data of the actors.

All in all, this diagram shows the architecture of the system. However, multiple elements of the system are combined within this diagram. This means that it does not only show an architecture but also possible activities or user-system interactions. From this, the conclusion can be made that there were different modelling types, described by the UML documentation, used within this one diagram.

5. RESULTS

The research is split up into 2 separate smaller researches

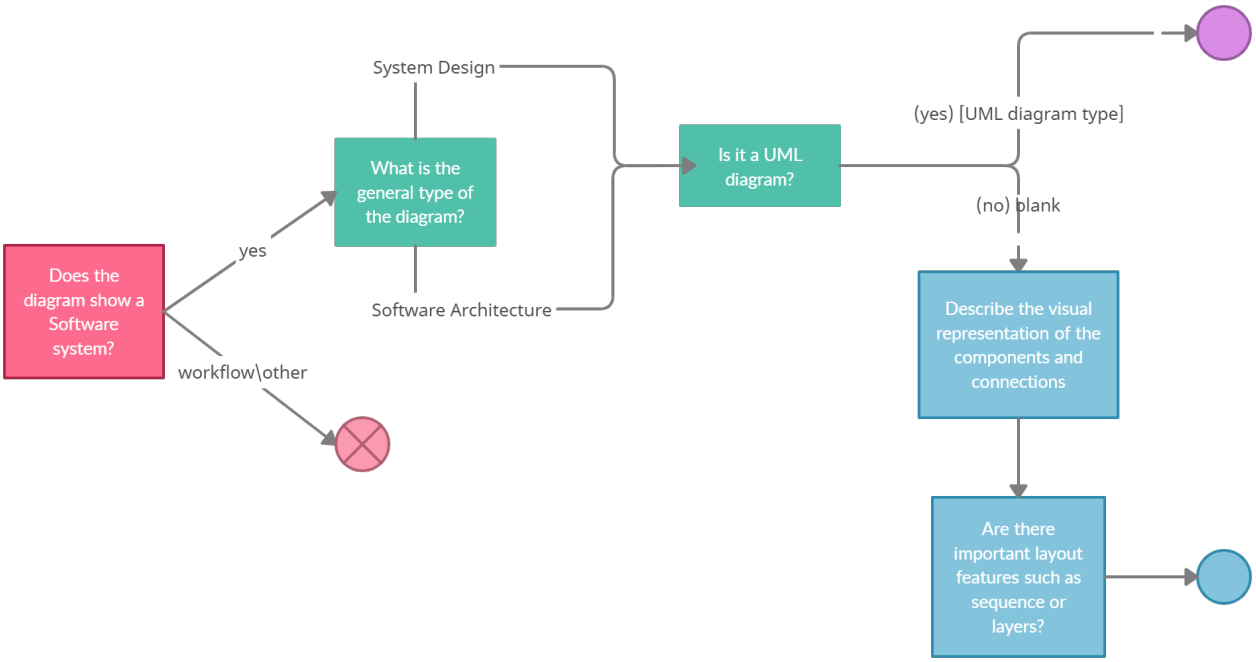

Figure 2. The process of an artifact observation analysis

where the collected data is from different sources. The results are also discussed per sub-research. In the end, the conclusion is given where the results are compared.

5.1 The Bachelor Showcase

The first data-set which is analysed is collected from the EEMCS Bachelor Showcase [1]. The students end their bachelor of computer science with a practical application of their knowledge and a research-oriented project. The results of these projects are presented at this Showcase.

33 projects were analysed. 9 of the 33 projects were research-oriented projects. 24 projects were the results of the Design project of Technical computer science.

Most students of the research projects did not create a software system. These students provided an extension or a theoretical approach towards a technical improvement. Therefore only two out of the nine observed research projects had one or more diagrams that were related to a system design.
Both had an architecture type of diagram which showed components and sub-components of the system. These components are connected with links that show the data flow between the components.
One of the researches showed a design diagram. In this diagram, a certain process was described with components and control flows.
No student showed a UML diagram.

For the Design Project, more software systems were created. These systems consisted mostly out of a web application. 25 projects were observed. 23 out of the 25 projects showed at least one diagram which is related to the design of a system. In these projects, there was a total of 75 design diagrams. Other present diagrams are related to the workflow of the development, used as a Graphical User Interface or related to the theory of the project.

18 out of the 75 diagrams showed the architecture of the system. These diagrams were similar to the architecture diagrams from the research projects. There were visible components of the system which are connected with a control or data flow.

54 of the 75 diagrams gave a more detailed description of the design of the system. 26% of these diagrams were similar to a UML activity or flow diagram. The diagrams showed activities and trigger events. The activities could either be executed by an actor or by the system itself.
A lot of these projects have a database incorporated into their systems. 14 of these 54 diagrams were a database schema that gave an overview of the database.
Additionally, 17% of the 54 diagrams are categorised as UML class diagram. These diagrams showed a detailed overview of the code of the system.

Although a variety of UML diagrams are used, 14 diagrams were not classified as UML. They mostly showed sub-components within components, namely 64%. The connections between these components differ. The preference is set on a data flow. 1 diagram even had a combination of a data flow and event flow. Other observed connections are control flows (2 diagrams) and associations (1 diagram).
Another noticeable fact is that the diagrams that are not in UML consisted of multiple characteristics in one diagram. For instance, components of the system are combined with user interactions in an architectural environment.

Lastly, the location of a diagram tells what the purpose of the diagram is. For example, when the diagram is shown on a poster it should be understandable for a novice. And when it is only shown in the appendix it could be considered as extra information which is not needed in the core of the paper or report. 105 diagrams were found in the design and research projects together. 81% of these diagrams were located within the core of the report. 15% were put in the appendix of the report. The posters were only part of the design project, therefore 4 out of the 54 were presented on a poster.

In short, there are students who use UML diagrams to describe their work. However, the majority chooses to use

components of their system together with a control or data flow. This gives a direct overview of the whole system.
Some students have put multiple aspects of the system within one diagram. These diagrams do not only represent the working of the system but also potential user interaction or an in/output. This construction does not benefit the understanding of the reader.

5.2 The 42nd International Conference on Software Engineering

The second data-set which is analysed is collected from the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE) [7]. ICSE is the premier software engineering conference, providing a forum for researchers, practitioners and educators to present and discuss the most recent innovations, research, experiences, trends and concerns in the field of Software engineering [3].

25 papers were observed. This is a subset of all the papers presented at the conference. Only 14 of these papers showed an overview of the system in one or more diagrams. Another noticeable fact is that there were some videos as extra presentation material. These videos mostly functioned as a demo of the tool that has been created. Still, 3 videos showed some design diagram. These were mostly architectural related and also present in the paper.

In total, the 14 papers (including the videos) showed 17 diagrams. Often, the papers include only 1 diagram which gives an insight into the architecture of the software system they have built. These systems were mostly tools that are used within a larger system. In the diagram, it was shown how the tool would interact with the system.

Only 1 diagram was a UML diagram, namely an activity diagram. The participants of the conference preferred to show the components of their tools together with either the input and output or a data flow with the information that is sent to the larger system. Also, different components of different UML diagrams were used in the same diagram. Components were combined with user-system interaction such as manually executed inputs and outputs for the system.

6. CONCLUSIONS

After looking at both groups, the students and the participants of a Software Engineering conference, it can be concluded that there are a lot of similarities but also quite some differences.

Both tend to create diagrams which are not specified in UML. They use their own way to describe the necessities of their system in order to understand it. The syntax of UML is not used for this, but several elements of different UML diagrams are used. They combined elements from the three different UML modelling types as discussed in section 4. Also for both, when they do use a UML specified diagram, the preference is to use an activity diagram.

The students had more variety of diagram types. although an architectural pattern was very common, diagrams that show any kind of event flow were also frequently present. Also for the design projects, when a project had a diagram, they often had more than one diagram included.
The participants of the conference often used only one diagram, which showed the architecture of the system. This was sometimes extended with the links to the larger system where their system is an extension for.

6.1 Recommendation

The combination of elements from different modelling types within one diagram is not preferred, since this makes a design model more difficult to interpret.
So firstly, the focus within the curriculum must be more on the importance of the different design models and modelling types. When the core ideas of the modelling types is more clear, the structure would be better even when combining elements from these different types.
Next, architecture was very common in use. However this was not by the use of class diagrams or any package structure. This does not comply with the course material of the Software Systems module.

From a personal point of view, the syntax of UML is clear and should be used more often. However, in the field it is shown that this is not preferred. That is why a deeper understanding of design elements is important. Then elements can perhaps be combined in a single diagram and still be clear. Inferring from the results, there is a demand for knowledge about architectural design models. UML offers more diagrams than those which are taught during the Software System module. However, those other diagrams still do not cover the needs of practitioners.

7. DISCUSSION

Since it was unknown what diagrams were used by practitioners, the schema was defined broadly. As shown, it still covers the characteristic components which are described in the UML specification. For a more detailed research it is advised to work out the schema into more detail. Though, the schema is sufficient for the goal of recognising modelling types and diagram elements.

For this research, only a small data-set is observed. This research could be fairly easily extended by increasing the size of the data-set. This will benefit the results of the research.

All in all, all the goals of this research are accomplished.

8. REFERENCES

[1] Bachelor Project Showcase - University of Twente. TCS/BIT bachelor projects. https://bachelorshowcase-eemcs.apps.utwente.nl/. Accessed: May 2021.

[2] EEMCS - Computer Science: Software Systems. https://www.utwente.nl/en/education/exchange-students/programmes/computer-science/courselink:201700117. Accessed: April 2021.

[3] ICSE 2020. https://conf.researchr.org/home/icse-2020. Accessed: June 2021.

[4] ICT.OPEN2021. https://www.ictopen.nl/. Accessed: June 2021.

[5] Omg unified modeling language - version 2.5.

[6] Research Information - University of Twente. https://research.utwente.nl/en/publications/. Accessed: June 2021.

[7] ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, 2020. Seoul, South Korea.

[8] D. Baisley, M. Björkander, C. Bock, S. Cook, P. Desfray, N. Dykman, A. Ek, D. Frankel, E. Gery, Øystein Haugen, S. Iyengar, C. Kobryn, B. Møller-Pedersen, J. Odell, G. Övergaard, K. Palmkvist, G. Ramackers, J. Rumbaugh, B. Selic,

T. Weigert, L. Williams, et al. OMG unified modeling language (OMG UML), superstructure. version 2.4.1. *Object Management Group*, Apr 2014.

[9] L. Burgueño, F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, et al. Contents for a model-based software engineering body of knowledge. *Software and Systems Modeling*, 2019.

[10] F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay, G. Taentzer, et al. Towards a body of knowledge for model-based software engineering. *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2018.

[11] A. C. A. Holanda, P. A. Tedesco, E. H. T. Oliveira, and T. C. S. Gomes. MOOCOLAB - A Customized Collaboration Framework in Massive Open Online Courses. 2020.

[12] V. Kumar and C. Troussas. International Conference on Intelligent Tutoring Systems (16th : 2020 : Online). *Intelligent Tutoring Systems: 16th International Conference Proceedings, Its 2020, Athens, Greece*, 2020.

[13] A. Pierantonio et al. A body of knowledge for model-based software engineering. https://modeling-languages.com/body-of-knowledge-model-based-software-engineering/, May 2020.

[14] G. Reggio, M. Leotta, and F. Ricca. Who knows/uses what of the UML: A personal opinion survey. *Lecture Notes in Computer Science Model-Driven Engineering Languages and Systems*, 2014.