Abstract Layers and Generic Elements as a Basis for Expressing Multidimensional Software Knowledge

Valentino Vranić[0000−0001−9044−4593] and Adam Neupauer

Institute of Informatics, Information Systems and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 84216 Bratislava 4, Slovakia
vranic@stuba.sk

Abstract. Enormous intellectual efforts are being invested into producing software in its executable form or, more precisely, a form from which this executable form can be automatically derived, commonly known as a source form (usually code, but may be a model, too). On the other hand, it is inherently complex to restore the ideas upon which software has been built. Moreover, it is usually not possible to produce software in its source form directly without producing a number of documents, diagrams, or schemes. All these artifacts, including the program code, represent software knowledge necessary for maintaining existing software and for building further systems in a given domain. But these software knowledge sources are disconnected from each other making it hard to navigate between them and to devise conclusions based on their relatedness, which is essential for their effective use. In this paper, a new approach to versatile graphical software modeling based on abstract layers and generic elements and its use in modeling multidimensional software knowledge and interrelating its pieces is proposed. The approach is supported by a prototype tool called InterSKnow, which targets mainly internal representation. This enabled to evaluate the approach from two perspectives: the efficiency of searching for software knowledge in software models and its comprehension. The results are generally plausible to the approach proposed here compared to Enterprise Architect as a representative of traditional, state-of-the-art software modeling tools.

Keywords: software artifacts · layers · UML · domain specific modeling · knowledge management

1 Introduction

Enormous intellectual efforts are being invested into producing software in its executable form [19,26] or, more precisely, a form from which this executable form can be automatically derived, commonly known as a source form (usually code, but may be a model, too [31,33]). On the other hand, it is inherently complex

Valentino Vranić and Adam Neupauer. Abstract Layers and Generic Elements as a Basis for Expressing Multidimensional Software Knowledge. In New Trends in Databases and Information Systems: ADBIS 2019 Short Papers, Workshops BBIGAP, QAUCA, SemBDM, SIMPDA, M2P, MADEISD, and Doctoral Consortium, CCIS 1064, Modelling is going to become Programming, a workshop at 23rd European Conference on Advances in Databases and Information Systems, M2P @ ADBIS 2019, Bled, Slovenia, 2019. Springer, 2019.

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-30278-8_26.

to restore the ideas upon which software has been built. Moreover, it is usually not possible to produce software in its source form directly without producing a number of documents, diagrams, or schemes. All these artifacts, including the program code, represent software knowledge necessary for maintaining existing software and for building further systems in a given domain. But these software knowledge sources are disconnected from each other making it hard to navigate between them and to devise conclusions based on their relatedness [22,25], which is essential for their effective use.

To overcome this, yet another model is necessary: a model that will interrelate pieces of software knowledge residing in different dimensions. However, this *multidimensional software knowledge* is maintained in different tools used to manage code and all kinds of models, including text descriptions and spreadsheets. Software development processes, as they are implemented, typically rely on these tools. Therefore, in many cases it is not feasible to extract software knowledge from its original sources and enforce maintaining it further only within a provided tool no matter how sophisticated it might be. It is hard to expect that such a supertool would ever support everything that is supported by dedicated tools. Thus, the purpose of the interrelating model would be to interconnect software knowledge as it is captured in the corresponding tools and to enable enhancing it with further details. It should be noted that the interrelating model need not be populated only manually. Code [5] and runtime information analyzers [1] could be used to generate parts of it.

The area of knowledge management in software engineering is very broad with hundreds of approaches [6,23,29,21]. The approach to expressing multidimensional software knowledge by an interrelating model proposed here stems in software modeling without enforcing any particular format for recording software knowledge or prescribing its taxonomy (such as the one in an earlier attempt at establishing a software development knowledge base [10]).

The rest of the paper is structured as follows. Section 2 analyzes what it would take to support modeling multidimensional software knowledge. Section 3 explains the approach to versatile graphical software modeling based on abstract layers and generic elements. Section 4 reports on the tool prototype and approach evaluation. Section 5 discusses related work. Section 6 concludes the paper.

2 Modeling Multidimensional Software Knowledge

UML was envisaged as the ultimate modeling solution applicable not only to software. As such, it was expected to put an end to the proliferation of different notations and establish mutual comprehensibility thus becoming the lingua franca of (at least) software modeling. However, this did not happen [30]. Other notations are in use, too, and new notations are being invented. Often, an ad hoc notation is a handy way to capture and express ideas. This may be at the level of diagram sketching [32], but taken in a more organized way, such notations may develop into domain specific modeling languages, which are being increasingly

recognized as a more flexible basis for model driven software development than (UML based) MDA is [9].

In a broader sense, a model is any artifact related to the software system being developed, including text documents (in different formats) and spreadsheets. Usually, code is distinguished from models, although it actually may be considered to be the final, executable model. Different models and code are related in complicated ways. Leaving this to depend on human memory is not feasible in the long run. It is not that only the new people joining the development process must discover the dependencies, but those that once knew the dependencies are forced to rediscover them once they forget them.

As it has been already pointed out in the introduction, to deal with multidimensional software knowledge, yet another model is needed: an interrelating model capable of expressing relationships between elements from the same or different models. Consider UML class diagram endless labyrinths. Understanding these often requires relating distant classes, as displayed in Figure 1.

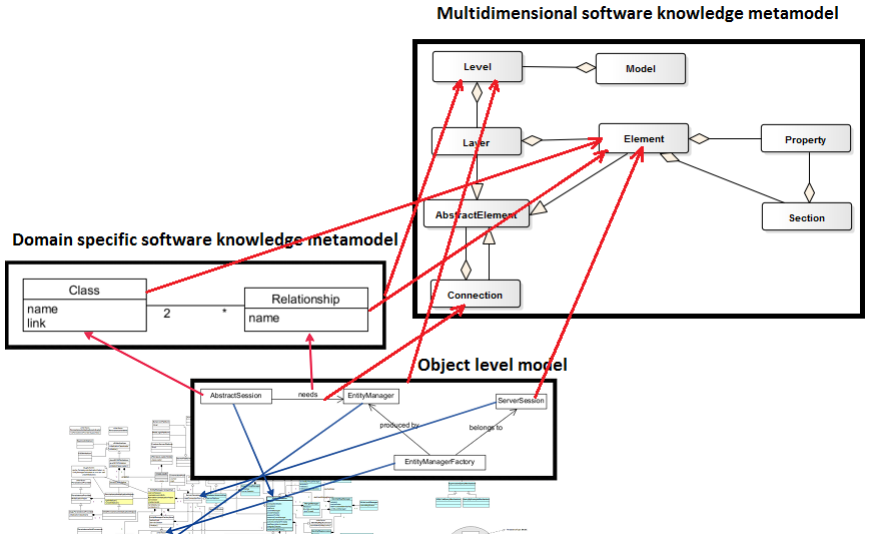

Fig. 1. Interrelating software knowledge: multidimensional software knowledge metamodel (the class diagram in the lower left part is a fragment from the EclipseLink 2.0 API UML class diagram[2]).

Relationships to external models and their elements may be expressed by using projections of these elements in additional models as in model federations [14]. In general, a set of such interconnecting models may be used allowing for multiple levels of interconnecting. Furthermore, the elements of these models may be not just projections of the elements from the models being interrelated,

[2] https://wiki.eclipse.org/EclipseLink/Development/Architecture/EclipseLink/ClassDiagram

but they may be enhancing the information contained in these models. This might even happen within the elements that are projections by introducing additional properties within these elements.

Semantics of the interrelating model may vary significantly. While capturing structural or static dependencies is very important, interrelating models could be very useful at expressing time dependencies or, in general, a behavioral or dynamic view. Providing a particular set of element types and connectors would be of limited use. For the full flexibility, this interrelating modeling framework has necessarily to be tailored to the context and this has to happen dynamically while performing application modeling as in free modeling [15]. Multiple levels of metamodeling are possible and it seems these are most transparently treated via the instantiation relationship as in deep modeling [3].

Some of the scenarios in which interrelating software knowledge may help include:

Change impact estimation. An envisaged specification change could be tracked through all the levels of models, code, and tests to estimate its complexity and cost, which would be used as a backing of the decision whether the change is viable.

Exploratory testing. The order of the steps the tester performs in the exploration test uncovers model and code interrelations. This can help in exploratory testing automation [13].

Fulfilling regular development tasks. A developer can more easily understand a software system via the interrelating model.

Tracking the intent of a software system to code. While intent is understandable in high-level specification and analytical artifacts, it easily gets lost in models and code [36]. Interrelating these could help and might be seen as a noninvasive alternative to preserving use cases in code [8].

3 Abstract Layers and Generic Elements

Graphical software modeling serves a broad spectrum of purposes ranging from conceptualization of ideas to executable models. Apart from UML as a general purpose modeling notation, there is a growing tendency towards domain specific modeling languages. Freedom to chose different kinds of *visualization* some of which may be even three-dimensional, seems to be essential. What models actually mean and how they can be used further constitutes another varying perspective: *interpretation*. To accommodate this versatility in visualization and interpretation, a sufficiently general *internal representation* is necessary.

Pages in a book, sheets of paper in general, blackboards in a classroom, or even diagrams in contemporary software modeling tools, they all indicate layering is a way of coping with complexity natural to humans. Making layers and elements they consist of, along with their relationships, abstract and generic and allowing them to be visualized and interpreted in different ways might be a key to the internal representation we look for. From this point of view, it seems that the main problem of common approaches to graphical software modeling is

interweaving visualization, interpretation, and internal representation, or giving priority to the former two over the internal representation.

Recall Figure 1 from Section 2. Its upper right part depicts the multidimensional software knowledge metamodel, This is, actually, a simplified version of the metamodel that we implemented in out prototype tool called InterSKnow. The metamodel enables creating a connection between any two abstract elements, which means that it is possible to create connections between layers and elements, including layer–layer and element–layer ones. An element consists of any number of properties, which may represent anything relevant to it, possibly organized into sections.

What the metamodel from Figure 1 does not show is that the elements can be of two kinds: native or proxy. While native elements can also be linked to other elements and, in some cases, represent them for the purposes of expressing further relationships between them, as depicted by the object level model in Figure 1, proxy elements are meant to reflect the original elements. The reflected elements need not be just elements of other graph based software models: they can be any software artifacts, such as text documents or spreadsheets.

In general, proxy elements should reflect all content of the original artifact or, more often, just a part of it. Clicking or otherwise invoking such a proxy element should open the original artifact for editing in the corresponding tool. InterSKnow provides this behavior for Microsoft Word and Excel files, but it does not reflect the content in proxy elements.

4 Tool Support and Evaluation

In order to enable the evaluation of the concept of using abstract layers and generic elements as a basis for expressing multidimensional software knowledge, we developed a prototype modeling tool called InterSKnow. The implementation is based on the Java 8 platform with JavaFX 2.0.

The evaluation embraced two perspectives: the efficiency of searching for software knowledge in software models, presented in Section 4.1, and its comprehension, presented in Section 4.2. Threats to validity are discussed in Section 4.3.

4.1 Efficiency of Searching for Software Knowledge in Software Models

The efficiency of searching for software knowledge in software models was assessed in the context of solving given tasks. The following set of metrics was used:

- Number of context switches during the course of solving a task (context switches cause distraction, which decreases efficiency)
- Number of files opened during the course of solving a task (the necessity to open many files complicates work making it less efficient)
- Time necessary to solve a task

Analogically to Bystrický and Vranić's interpretation [8] related to source code, by a context switch in modeling we mean a necessity to look elsewhere than at the current diagram (within or outside the current model). This breaks a thread of thought, which negatively affects efficiency.

The experiment involved six participants, each of which had to solve three tasks, either with the InterSKnow tool or with Enterprise Architect as a representative of traditional, state-of-the-art software modeling tools. Each participant was provided with an equal computer configurations. The subject of the experiment was a model of a web based insurance system consisting of three separate applications called Insurance, Customer, and Notifier, with a common front end. The tasks performed by the participants were:

- Task 1: Finding and opening the corresponding class to add a new insurance policy type
- Task 2: Finding the unimplemented components that need to be removed
- Task 3: Finding the description and the class that corresponds to the Customer application interface

Table 1 summarizes the results. In a traditional setting, task 1 involved sequentially going through the models and their diagrams since the participants had to find a class whose name wasn't known to them. The results regarding the number of files opened only slightly favor InterSKnow because task 1 was focused on the interconnectedness of artifacts within one software modeling tool. Nevertheless, InterSKnow saves some effort by enabling to open directly the source code of a given class. The difference in context switches is significant. InterSKnow context switches count only for switches between layers, which are straightforward. Enterprise Architect required opening several models with the necessity to get back to previous ones since the participants tend to forget the relationships between these models. The time perspective also favors InterSKnow, which correlates with context switches.

Table 1. Efficiency of searching for software knowledge in software models.

	Context switches			Files opened			Time		
	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3	Task 1	Task 2	Task 3
InterSKnow									
Participant 1	4	3	8	3	2	5	31	29	49
Participant 2	3	5	5	3	3	4	25	34	35
Participant 3	7	3	9	4	2	5	45	33	55
Enterprise Architect									
Participant 4	9	19	15	3	2	7	35	123	68
Participant 5	14	15	13	5	2	6	76	91	51
Participant 6	11	25	18	4	2	5	62	200	76

Task 2 results strongly favor InterSKnow in context switches and necessary time. This was expected, since Enterprise Architect does not express explicitly

the relationships between components and packages, nor between packages and classes, which made the participants systematically open diagram by diagram.

Task 3 was focused on working with external software artifacts. In a traditional setting, the participants had to open all incriminated files one by one. With InterSKnow, the files were available directly from the tool via proxy elements. In many cases, it wasn't even necessary to click on the proxy element, since the content reflected by it was sufficient to determine whether the file is relevant or not.

4.2 Comprehension of Software Knowledge in Software Models

The comprehension of software knowledge in software models was assessed by verifying how much participants were able to learn from a given software model within a limited time using a questionnaire that included the following questions:

1. What version of the Spring boot is used by the Insurance application components?
2. Which application communicates via RESTS?
3. Which internal application communicates via the AMQP message system?
4. What types of messages are sent by the Notifier application?
5. What infrastructure technology is used in the insurance system (deployment/configuration)?
6. In what format does the Insurance application communicate via the external API?

The experiment involved eight participants. Each participant was shown the questionnaire for twenty seconds. Afterwards, they had three minutes to study the model. The same models as in the first experiment were used. Finally, the participants had to fill in the questionnaire they had been shown initially.

As can be observed in Table 2, the results speak in favor of InterSKnow. It only failed to outperform Enterprise Architect with respect to question 5. This can probably be attributed to the Enterprise Architect GUI as InterSKnow is only a prototype.

4.3 Threats to Validity

The fact that, unlike InterSKnow, Enterprise Architect was known to all participants represents a threat to internal validity. This was an advantage to Enterprise Architect. The results for an unknown tool would have probably been worse. To mitigate this, the participant had a brief demonstration of InterSKnow (on a different model than the one that was used in experiments, of course).

A low number of participants represents a threat to external validity. While we haven't had a possibility to increase the number of participants, we've taken care not to engage the same participants in experiments with both InterSKnow and Enterprise Architect.

The model in Enterprise Architect could have been created with some of the external software artifacts made available within the model itself using, for

Table 2. Comprehension of software knowledge in software models (correct answers indicated by the check mark).

	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6
InterSKnow						
Participant 1	✓	✓			✓	
Participant 2		✓	✓	✓	✓	✓
Participant 3		✓	✓			
Participant 4		✓	✓	✓		✓
Enterprise Architect						
Participant 5		✓	✓			
Participant 6					✓	
Participant 7		✓			✓	
Participant 8			✓	✓	✓	

example, the UML note element. This can be viewed as another threat to external validity. However, an extensive use of UML notes to internalize external software artifacts is not a common practice.

5 Related Work

Melanee, Multilevel Modeling and Domain-Specific Language Workbench [18,2,3], is a workbench for so-called multilevel or deep modeling. It supports creating both graphical and textual domain-specific modeling languages clearly separating their metamodeling levels on the instantiation basis. Differently than InterSKnow, Melanee provides no support of connecting to external software artifacts. Similarly to InterSKnow, Melanee decouples visualization, interpretation, and internal representation of the model, but it does so only partially. This involves employing the concept of layers (sometimes called levels, as in MetaCase [27]), which can also be observed in other approaches as well [24,11]. However, the layers are used there only as a means of distinguishing the level of abstraction. InterSKnow makes no limitations to the interpretation of layers.

Openflexo is an environment that makes possible forming a federation of software models contained in different tools by providing connectors to these tools [28,14]. In some cases, it is possible to maintain a live connection, i.e., upon changing an element or value in the external tool, its proxy element reflects the change automatically. The idea behind OpenFlexo itself is similar to our idea of interrelating software artifacts maintained in various external tools. InterSKnow goes beyond this by employing multiple levels of modeling in the interrelating model.

EMF Views is a tool that enables combining the information from different models in one view in the database (SQL) view fashion [4,7]. Some of the views are live, i.e., changes to proxy elements in a view are reflected in the actual elements. Again, as with Openflexo, this is similar to our idea of interrelating software artifacts maintained in various external tools. InterSKnow goes beyond

this by making its interrelating model capable of introducing interconnections and further information not contained within the models being interrelated.

Among commercially available tools, IBM Rational Rhapsody Gateway enables elaborated connection to external software artifacts and their processing, mainly in the context of requirements engineering [20]. Different models and documents can be interconnected in this tool. Among supported ones are Microsoft Word, Microsoft Excel, and PDF formats. All these are transformed into a unified XML format within the import process that enables specifying what parts of these documents should be imported. Connected external software artifacts are available for opening directly from the graphical editor. However, all these documents are interpreted as sources of requirements, which makes it difficult to express other intentions. Moreover, Rhapsody Gateway itself does not support creation of software models, so even UML models have to be created elsewhere (usually in Rhapsody Modeler). InterSKnow, on the other side, aims at being a versatile software modeling tool.

6 Conclusions and Further Work

In this paper, a new approach to versatile graphical software modeling based on abstract layers and generic elements and its use in modeling multidimensional software knowledge and interrelating its pieces is proposed. The approach is supported by a prototype tool called InterSKnow, which targets mainly internal representation. This enabled to evaluate the approach from two perspectives: the efficiency of searching for software knowledge in software models and its comprehension. The results are generally plausible to the approach proposed here compared to Enterprise Architect as a representative of traditional, state-of-the-art software modeling tools.

The evaluation confirmed the importance of visualization which is largely underdeveloped in InterSKnow. A layered 3D visualization of software models [12,16,17] naturally fits the approach proposed in this paper. It could also be put into virtual reality [35,34]. Both possibilities would enhance opportunities for improving collaboration in distributed software development.

Acknowledgments

The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/0759/19 and by the Research & Development Operational Programme for the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Asadi, F., Di Penta, M., Antoniol, G., Guéhéneuc, Y.G.: A heuristic-based approach to identify concepts in execution traces. In: Proceedings of 14th Euro-

pean Conference on Software Maintenance and Reengineering, CSMR 2010. IEEE, Madrid, Spain (2010)
2. Atkinson, C., Gerbig, R.: Flexible deep modeling with melanee. In: Proceedings of Modellierung 2016. LNI, GI, Karlsruhe, Germany (2019)
3. Atkinson, C., Gerbig, R., Kühne, T.: Comparing multi-level modeling approaches. In: Proceedings of 1st Workshop on Multi-Level Modelling, co-located with 17th ACM/IEEE International Conference on Model-Driven Engineering Languages and Systems, MODELS 2014. vol. 1286. CEUR, Valencia, Spain (2014)
4. AtlanMod: EMF Views. `https://www.atlanmod.org/emfviews/` (2019)
5. Bavota, G., Laskowska, A., Chulani, I., De Nigro, A., Di Penta, M., Galletti, D., Galoppini, R., Gordon, T., Kedziora, P., Lener, I., Torelli, F., Pratola, R., Pukacki, J., Rebahi, Y., Garcia Villalonga, S.: The market for open source: An intelligent virtual open source marketplace. In: Proceedings of IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering, CSMR-WCRE 2014. IEEE, Antwerp, Belgium (2014)
6. Bjørnson, F.O., Dingsøyr, T.: Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. Information and Software Technology **50**(11), 1055–1068 (2008)
7. Bruneliere, H., Perez, J.G., Wimmer, M., Cabot, J.: EMF Views: A view mechanism for integrating heterogeneous models. In: Proceedings of 34th International Conference on Conceptual Modeling, ER 2015. LNCS 9381, Springer, Stockholm, Sweden (2015)
8. Bystrický, M., Vranić, V.: Preserving use case flows in source code: Approach, context, and challenges. Computer Science and Information Systems Journal (ComSIS) **14**(2), 423–445 (2017)
9. Dalgarno, M., Fowler, M.: UML vs. domain-specific languages. Methods & Tools **16**(2), 2–8 (2008)
10. Devanbu, P., Brachman, R., Selfridge, P.G., Ballard, B.W.: LaSSIE: A knowledge-based software information system. Communications of the ACM **34**(5), 34–49 (1991)
11. Englebert, V., Heymans, P.: Towards more extensible MetaCASE tools. In: Proceedings of 19th International Conference on Advanced Information Systems Engineering, CAiSE 2007. Springer, Trondheim, Norway (2007)
12. Ferenc, M., Polášek, I., Vincúr, J.: Collaborative modeling and visualisation of software systems using multidimensional UML. In: Proceedings of 5th IEEE Working Conference on Software Visualization, VISSOFT 2017. IEEE, Shangai, China (2017)
13. Frajták, K., Bureš, M., Jelínek, I.: Exploratory testing supported by automated reengineering of model of the system under test. Cluster Computing **20**(1), 855–865 (2017)
14. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Addressing modularity for heterogeneous multi-model systems using model federation. In: MODULARITY Companion 2016, Companion Proceedings of 15th International Conference on Modularity. ACM, Málaga, Spain (2016)
15. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Using free modeling as an agile method for developing domain specific modeling languages. In: Proceedings of ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS 2016. ACM, Saint-Malo, France (2016)
16. Gregorovič, L., Polášek, I.: Analysis and design of object-oriented software using multidimensional UML. In: Proceedings of 15th International Conference on Knowledge Technologies and Data-Driven Business. ACM, Graz, Austria (2015)

17. Gregorovič, L., Polášek, I., Sobota, B.: Software model creation with multidimensional UML. In: Proceedings of 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, part of WCC 2015. LNCS 9357, Springer, Daejeon, Korea (2015)
18. Group, S.E.: Melanee project website. University of Mannheim, `http://www.melanee.org` (2014)
19. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Computing Surveys **45**(1), Article No. 11 (2012)
20. IBM: Ibm rational rhapsody gateway. `https://www.ibm.com/support/knowledgecenter/SSB2MU_8.1.5/com.ibm.rhp.oem.pdf.doc/pdf/dassault/User Guide.pdf` (2014)
21. Indumini, U., Vasanthapriyan, S.: Knowledge management in agile software development – a literature review. In: Proceedings of 2018 National Information Technology Conference, NITC 2018. Colombo; Sri Lanka (2018)
22. Keivanloo, I., Forbes, C., Hmood, A., Erfani, M., Neal, C., , Peristerakis, G., Rilling, J.: A linked data platform for mining software repositories. In: Proceedings of 9th IEEE Working Conference on Mining Software Repositories, MSR '12. IEEE, Zurich, Switzerland (2012)
23. Khalil, C., Khalil, S.: Exploring knowledge management in agile software development organizations. International Entrepreneurship and Management Journal p. 15 (2019)
24. de Lara, J., Guerra, E.: Deep meta-modelling with MetaDepth. In: Proceedings of International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, TOOLS 2010: Objects, Models, Components, Patterns. LNCS 6141, Springer, Málaga, Spain (2010)
25. Makedonski, P., Sudau, F., Grabowski, J.: Towards a model-based software mining infrastructure. ACM SIGSOFT Software Engineering Note **40**(1), 1–8 (2015)
26. Matharu, G.S., Mishra, A., Singh, H., Upadhyay, P.: Empirical study of agile software development methodologies: A comparative analysis. ACM SIGSOFT Software Engineering Note **40**(1), 1–6 (2015)
27. MetaCase: MetaCase website. `https://www.metacase.com/` (2019)
28. Openflexo: Openflexo project website. `https://www.openflexo.org/` (2019)
29. Ouriques, R., Wnuk, K., Gorschek, T., Berntsson Svensson, R.: Knowledge management strategies and processes in agile software development: A systematic literature review. International Journal of Software Engineering and Knowledge Engineering **29**(3), 00153 (2018)
30. Petre, M.: UML in practice. In: Proceedings of 35th International Conference on Software Engineering, ICSE 2013. IEEE, San Francisco, CA, USA (2010)
31. da Silva, A.R.: Model-driven engineering: A survey supported by the unified conceptual model. Computer Languages, Systems & Structures **43**, 139–155 (2015)
32. Socha, D., Tenenberg, J.: Sketching software in the wild. In: Proceedings of 35th International Conference on Software Engineering, ICSE 2013. IEEE, San Francisco, CA, USA (2010)
33. Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G.: Relevance, benefits, and problems of software modelling and model driven techniques—a survey in the italian industry. Journal of Systems and Software **86**(8), 2110–2126 (2013)
34. Vincúr, J., Návrat, P., Polášek, I.: VR City: Software analysis in virtual reality environment. In: IEEE International Conference on Software Quality, Reliability and Security, QRS 2017. IEEE, Prague, Czech Republic (2017)

35. Vincúr, J., Polášek, I., Návrat, P.: Searching and exploring software repositories in virtual reality. In: Proceedings of ACM Symposium on Virtual Reality Software and Technology, VRST 2017. ACM, Gothenburg, Sweden (2017)
36. Vranić, V., Porubän, J., Bystrický, M., Frťala, T., Polášek, I., Nosáľ, M., Lang, J.: Challenges in preserving intent comprehensibility in software. Acta Polytechnica Hungarica **12**(7), 57–75 (2017)