UML Modelleme Araçlarının Pratik Kullanım için Analizi

Mert Ozkaya[1] and Ferhat Erata[2]

[1] Yeditepe Üniversitesi , Ataşehir, İstanbul
mozkaya@cse.yeditepe.edu.tr
[2] UNIT Bilgi Teknolojileri R&D Ltd., Bornova, Izmir
ferhat.erata@unitbilisim.com

Özet. Günümüzde, Unified Modeling Language(UML) pratisyenler tarafından en sık tercih edilen yazılım sistemi modelleme ve tasarlama notasyonu olarak kabul edilmektedir. UML, aynı zamanda, birçok yazılım modelleme aracı tarafından desteklenmektedir, ve bu araçlar sayesinde, pratisyenler yazılım sistemlerini kolayca UML notasyonunu kullanarak modelleyebilir ve analiz, yazılım kodu üretme, ve işbirliği gibi birçok faydalı değişik işlemler gerçekleştirebilirler. Bu çalışmada, tanınan 11 farklı UML modelleme aracını pratisyenlerin UML'i benimsemeleri açısından önemli olduğunu düşündüğümüz bir grup gereksinim bakımından analiz ettik. Bu gereksinimler başlıca, modellerin tasarımı, model analizi, modelden kod üretme, iş-birliği halinde modelleme, ve genişletilebilirlik olmaktadır. Model tasarımı gereksinimi, modelleme araçlarının UML diyagramlarına olan destekleri, yazılım modelleme bakış-açılarına olan destekleri, ve büyük ve karmaşık yazılım modellerinin tasarımına olan destekleri açılarından ele alınmaktadır. Model analizi gereksinimi, simülasyon ve doğrulama (hem önceden tanımlanmış doğrulama hem de kullanıcı tanımlı doğrulama) gereksinimlerine olan destek bakımından incelenmektedir. İş-birliği halinde modelleme gereksinimi ise, senkron ve asenkron olarak çoklu kullanıcı desteği, görev yönetimi, paydaşlar arası iletişim, ve versiyonlama destekleri açısından incelenmektedir. Çalışmamızın sonuçları ile, UML modelleme araçlarının güçlü ve zayıf yönlerinin kolayca anlaşılması ve aynı zamanda hangi UML modelleme aracının (ya da araçlarının) göz önünde bulundurulan gereksinimler bakımından daha iyi olduğunun ve hangi gereksinimlerin az yada çok ilgi gördüğünün ortaya çıkarılması hedeflenmektedir.

Anahtar Kelimeler. UML, Modelleme Araçları, Analiz, Otomatik kod üretme, İş-birliği halinde modelleme, Genişletilebilirlik

Analysing UML Modeling Tools for Practical Use

Mert Ozkaya[1] and Ferhat Erata[2]

[1] Yeditepe University, Atasehir, Istanbul
mozkaya@cse.yeditepe.edu.tr
[2] UNIT Information Technologies R&D Ltd., Bornova, Izmir
ferhat.erata@unitbilisim.com

Abstract. Unified Modeling Language (UML) is nowadays one of the top used software modeling languages by practitioners. UML is supported by many modeling tools through which practitioners can use UML for modeling their software systems and also perform many useful operations such as analysis and code generation. In this paper, we analysed 11 different well-known UML modeling tools for a set of requirements that we believe are highly important for the practical adoption of the tools in software modeling. These requirements are concerned with modeling, analysis, code-generation, user-collaboration, and tool extensibility. Modeling herein is considered in terms of the supported UML diagrams, viewpoint management, and large view management. Analysis is considered in terms of the support for simulation and validation (i.e., the support for pre-defined and user-defined rules). Also, collaboration is considered in terms of multi-user support for the synchronous and asynchronous collaboration, task management, communication, and versioning. The analysis results of the UML tools shed light on which UML modeling tool(s) provide better support in terms of those requirements considered and which requirement(s) are shown a lack of interest by the modeling tools.

Keywords: UML, Modeling Tools, Analysis, Code Generation, Collaboration, Extensibility

1 Introduction

Unified Modeling Language (UML) [6, 7, 18] has been proposed in the early nineties as the notation set for specifying and designing software systems. Nowadays, UML is considered as the de-facto standard for software modeling and widely used by practitioners in several industries [11, 13]. UML offers various types of visual diagrams that can be used for modeling software systems from different perspectives. That is, practitioners may use UML diagrams for documenting the logical structure of systems, behaviours and interactions, software architectures, the physical and deployment structures. The UML diagrams may also be categorised as the static and dynamic diagrams where the static diagrams are concerned with the structural modeling of systems from different perspectives and the dynamic diagrams are concerned with the behavioural modeling of systems. The structural diagrams are represented with UML's class diagram, object

diagram, composite structure diagram, component diagram, package diagram, and deployment diagram. The dynamic diagrams are represented with UML's sequence/communication diagram, state machine diagram, timing diagram, and activity diagram.

Several software modeling tools have been developed for UML, through which practitioners can design their software systems in UML and perform many other operations such as analysis, code generation, team work, task management, etc. However, it is not so easy for practitioners to understand which UML software modeling tool(s) are better for their needs. Therefore, in this paper, we aim at analysing the well-known UML tools for a set of requirements that we believe are highly important for practitioners. The UML software modeling tools that we consider in our analysis are Visual Paradigm[3], MagicDraw[4], StarUML[5], Modelio[6], Enterprise Architect[7], ArgoUML [8], BoUML [9], Obeo UML Designer[10], Eclipse Papyrus[11], IBM Rational Rhapsody[12], Umbrello UML[13]. The requirements for which we plan to analyse those UML tools are concerned with the *(i)* modeling capabilities, *(ii)* analysis capabilities, *(iii)* code-generation, *(iv)* collaboration, and *(v)* extensibility.

Modeling. The tool support for modeling software systems is considered in terms of three requirements: *(i)* the supported UML diagrams, *(ii)* the viewpoint management, and *(iii)* large view management. The supported UML diagrams aid in understanding the static and dynamic diagrams of UML that the UML tools support. The software viewpoints are intended for modeling software systems from different perspectives that each deal with a particular aspect of the software system modeled. So, the viewpoint management management is concerned with understanding the different viewpoints that the tools support for the modeling of software systems using UML. Kruchten [9] has initially proposed the logical, process, development, and physical viewpoints. Later on, Rozanski et al. [17] have proposed a comprehensive work on software viewpoints and proposed the functional, information, concurrency, development, physical, deployment, and operational viewpoints. Lastly, the large view management requirement is concerned with modeling large software systems with many components that are hierarchically composed of other components in an effective way for the eased understandability and analysis. So, herein, the UML modeling tools are intended to be analysed for determining whether they offer any means of handling this complexity. For instance, it would be desirable to specify the

[3] https://www.visual-paradigm.com/
[4] https://www.magicdraw.com/
[5] http://staruml.io/
[6] https://www.modelio.org/
[7] http://sparxsystems.com/products/ea/
[8] http://argouml.tigris.org/
[9] http://www.bouml.fr/
[10] http://www.umldesigner.org/
[11] https://www.eclipse.org/papyrus/
[12] https://www.ibm.com/us-en/marketplace/rational-rhapsody
[13] https://umbrello.kde.org/

sub-architecture for any complex component by clicking on the component that opens up a new sub-editor for the sub-architecture modeling.

Analysis. We consider model analysis in two aspects: simulation and validation. The model simulation is concerned with executing software system behaviours specified using UML's dynamic diagrams (e.g., state diagram and activity diagram) and analysing whether the system behaves as expected or not (e.g., taking the system actions in the right sequence). The model validation is concerned with checking the UML models against the well-definedness rules of UML (e.g., model completeness, correctness, consistency, and compatibility) and any other rules that practitioners deem important for their domain.

Code-generation. Code-generation is concerned with the ability of transforming UML models into software implementation. So, practitioners can automatically have the executable software code that precisely reflect their UML models.

Collaboration. Collaboration is concerned with the collaborative modeling of software systems that promote fast and high-quality modeling. We consider collaboration in terms of multi-user support for synchronous (i.e., at the same time) and asynchronous (i.e., offline) collaborations, *(i)* task management (e.g., assigning/removing tasks to/from the developers, prioritising the tasks, etc.), *(ii)* communications among stakeholders (e.g., instant messaging for discussing over UML diagrams), and *(iii)* versioning.

Extensibility. Extensibility is concerned with the ability of modifying the software tool with some extra capabilities that practitioners need (e.g., translator for some additional programming languages, translators for model checker tools, new tool windows, and menu bars).

2 Research Methodology

In this study, 11 different UML modeling tools are considered, which have been determined from the list of the UML modeling tools maintained by Wikipedia[14]. Due to the space limit, we were not able to consider every single UML modeling tool in the list, and instead, focussed on the UML tools that are highly popular among the practitioners and academics. Indeed, based on wide our experiences on UML modeling [12, 14] and the connections that we maintain with the industry and academia, the UML modeling tools considered in this study are among the top preferred tools used for the UML-based software modeling by practitioners and academics.

To analyse the UML modeling tools, we considered Lago et al.'s language requirements [10], which are grouped into the language definition, language features, and tool support. In this study, we focussed on the tool support requirements, which are concerned with the large-view management, the model analysis, code-generation, collaboration, and extensibility. Moreover, we also extended the list with the support for UML diagram types and multiple-viewpoint

[14] The list of the existing UML modeling tools: `https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools`

Table 1. The UML tools and their support for the static UML diagrams

UML Tools	Class Diagram	Package Diagram	Object Diagram	Component Diagram	Composite Structure Diagram	Deployment Diagram	Profile Diagram
Visual Paradigm	Yes	Yes	Yes	Yes	Yes	Yes	Yes
MagicDraw	Yes	Yes	Yes	Yes	Yes	Yes	Yes
StarUML	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Modelio	Yes	Yes	Yes	Yes	No	Yes	No
Enterprise Architect	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ArgoUML	Yes	No	No	No	No	Yes	Yes
BoUML	Yes	No	Yes	Yes	No	Yes	Yes
Obeo UML Designer	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Eclipse Papyrus	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Rational Rhapsody	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Umbrello UML	Yes	No	Yes	Yes	No	Yes	No

Table 2. The UML tools and their support for the dynamic UML diagrams

UML Tools	Activity Diagram	Sequence Diagram	Use Case Diagram	State Diagram	Comm. Diagram	Timing Diagram
Visual Paradigm	Yes	Yes	Yes	Yes	Yes	Yes
MagicDraw	Yes	Yes	Yes	Yes	Yes	Yes
StarUML	Yes	Yes	Yes	Yes	Yes	No
Modelio	Yes	Yes	Yes	Yes	Yes	No
Enterprise Architect	Yes	Yes	Yes	Yes	Yes	Yes
ArgoUML	Yes	Yes	Yes	Yes	Yes	No
BoUML	Yes	Yes	Yes	Yes	Yes	No
Obeo UML Designer	Yes	Yes	Yes	Yes	Yes	No
Eclipse Papyrus	Yes	Yes	Yes	Yes	Yes	Yes
Rational Rhapsody	Yes	Yes	Yes	Yes	Yes	Yes
Umbrello UML	Yes	Yes	Yes	Yes	Yes	No

management support. Indeed, these two requirements are also highly crucial for practitioners and, in our experience, not all the UML modeling tools provide full support for them.

In our analysis of the tools for the pre-determined sub-set of requirements, we initially determined the resources for each tool that consist of the tool web-site, user-manual, tutorials, and any publications (if available). Each of us has analysed the tools using the resources available separately and recorded the results in an excel file. Then, we compared our analysis results to determine any discrepancies. For each discrepancy determined, we re-analysed the tools in question and conducted discussions together until no more conflicts left.

3 The Summary of the Analysis Results

3.1 Modeling

In Tables 1 and 2, the UML tools are analysed with regard to their support for the UML's static and dynamic diagrams respectively. Note that the static and dynamic UML diagrams have been determined from UML's OMG specification [15]. Concerning the static diagrams, all the UML tools support the UML class and deployment diagrams. UML's component and object diagrams are only omitted by ArgoUML. The rest of the static UML diagrams are just omitted a few UML

[15] UML's OMG specification version 2.5.1 can be accessible via the following link: `https://www.omg.org/spec/UML/About-UML/`.

Table 3. The UML tools and their support for the viewpoint management

UML Tools	Viewpoint Management
Visual Paradigm	Yes
MagicDraw	No
StarUML	Yes
Modelio	No
Enterprise Architect	Yes
ArgoUML	No
BoUML	No
Obeo UML Designer	No
Eclipse Papyrus	No
Rational Rhapsody	Yes
Umbrello UML	No

Table 4. The UML tools and their support for the large-view management

UML Tools	Components	Connectors
Visual Paradigm	Yes	Yes
MagicDraw	Yes	No
StarUML	Yes	Tags
Modelio	Yes	Attributes
Enterprise Architect	Yes	Constraints
ArgoUML	No	No
BoUML	No	No
Obeo UML Designer	No	No
Eclipse Papyrus	No	No
Rational Rhapsody	Yes	No
Umbrello UML	No	No

tools. Concerning the dynamic diagrams, all the UML tools support the activity, sequence, use case, state, and communication diagrams. However, many UML modeling tools ignore the UML timing diagram.

As shown in Table 3, the viewpoint management is supported by a few UML tools only. Enterprise Architect offers the use-case, dynamic (i.e., behaviour), class/component (i.e., logical), and deployment views for the UML modeling of software systems. Also, Enterprise Architect allows practitioners to create their own views that meet their domain constraints better. IBM Rational Rhapsody offers the use case, requirement, component, unit, and properties views. Also, Rational Rhapsody offers some add-ons for modeling software systems from additional viewpoints, including operational, strategic, acquisition, and technical viewpoints. Visual Paradigm offers practitioners a set of pre-defined viewpoints that are specified in terms of the stakeholders involved, the purpose of the viewpoint, the concerns, the particular aspects of the software architecture, and any visual elements supported. Practitioners may modify the viewpoints, e.g., introducing new types of stakeholders. Also, practitioners may associate their models with a viewpoint to restrict the access to the model with the viewpoint stakeholders only. Lastly, StarUML adopts Kruchten's 4+1 view model and promotes the specifications of software systems in terms of the logical, process, development, and deployment aspects. Practitioners are offered with a different set of notations for each view and also allowed to document any viewpoint-related information textually.

The large-view management is supported by Visual Paradigm, MagicDraw, Enterprise Architect, StarUML, Modelio and Rational Rhapsody via sub diagramming. That is, for any system unit (e.g., component, class, package, interface, activity, and state) specified, practitioners may click on it that opens up a new sub-editor for specifying the internal structure of that element as a new UML diagram. In Table 4, the large view management has been considered in terms of the component and connector units composing the systems. Component herein may be perceived as any computational unit specified in a UML model (e.g., class) and connector perceived as the point of interaction between the components (e.g., the class associations). While the tools support the management of large components, they mainly ignore the need for modeling the large connector units. Indeed, connectors may represent complex interaction

Table 5. The UML tools and their analysis support

UML Tools	Simulation	Validation	
		Pre-defined Properties	User-defined Properties
Visual Paradigm	No	No	No
MagicDraw	Yes	Completeness, correctness, and well-definedness rules	OCL properties
StarUML	No	Well-definedness rules	No
Modelio		Well-definedness rules	Module definition in Java
Enterprise Architect	Yes	Well-definedness	OCL constraints
ArgoUML	No	No	No
BoUML		No	No
Obeo UML Designer	No	Well-definedness rules	Validation rules in Acceleo
Eclipse Papyrus	No	Well-definedness rules	Validation rules in OCL
Rational Rhapsody	Yes	Completeness, correctness, and well-definedness rules	Validation rules in API
Umbrello UML	No	No	No

Table 6. The UML tools and their collaboration support

UML Tools	Multi-user Support	Task Management	Communication	Versioning
Visual Paradigm	-Sync. and async collaboration	Yes	Yes	Cloud-based
MagicDraw	-Sync. and async collaboration	No	Yes	Cloud-based
StarUML	No	No	No	No
Modelio	Sync. and async collaboration	No	No	Server-based
Enterprise Architect	-Sync. and async collaboration	Yes	Yes	Server-based
ArgoUML	No	No	No	No
BoUML	No	No	No	No
Obeo UML Designer	No	No	No	No
Eclipse Papyrus	-Sync. and async collaboration	No	No	Server-based (GIT)
Rational Rhapsody	-Sync. and async collaboration	No	No	Server-based
Umbrello UML	No	No	No	No

mechanisms (e.g., adapters and distributors) that need to be modeled in terms of sub-components and connectors. Only, Visual Paradigm offers the option for attaching sub-diagrams for the connectors. StarUML and Modelio offer the specifications of mere attributes for the connectors. Enterprise Architect offers the specifications of OCL constraints for connectors.

3.2 Analysis

Table 5 shows UML tools and their support for the model analysis, which are considered in terms of the simulation and validation support. Note that the validation support is considered in terms of the pre-defined validation rules and user-defined rules. According to the results, Visual Paradigm and Umbrello do not provide any support for the model simulation and validation. MagicDraw and Enterprise Architect are the only UML tools that support the simulation and analysis requirements together at the same time. Note Visual Paradigm supports the simulation of behavioural models, which is however restricted with the Business Process models and not applicable for UML diagrams.

The simulation requirement is supported by MagicDraw, Enterprise Architect, and Rational Rhapsody, which essentially allow practitioners to simulate the behaviour of their software systems specified via the UML state diagram or activity diagram. Concerning the model validation, all the UML tools (see Table 5) support the pre-defined validation properties and at the same time allow practitioners to formulate their own validation properties that can be checked automatically - except StarUML. However, as given in Table 5, the notations used for defining the user-defined validation properties differ among the tools.

Table 7. The UML tools and their code generation support

UML Tools	Programming Languages
Visual Paradigm	Java, C#, VB.NET, PhP, ODL, ActionScript, IDL, C++, Delphi, Perl, XML, Phyton, ObjectiveC, Ada, Ruby
MagicDraw	Java, C++, C#, Corba, WSDL, DDL, XML
StarUML	Ruby, PhP, TypeScript, XMI, Phyton, C++, C#, Java
Modelio	Java
Enterprise Architect	Action Script, C, C#, C++ , Delphi, Java, PHP, Python, Visual Basic, Visual Basic .NET
ArgoUML	Java, C#, C++, and PHP
BoUML	Java, C, C++, Phyton, SQL, PhP, IDL
Obeo UML Designer	Java
Eclipse Papyrus	Java and C++
Rational Rhapsody	Java, C, C++
Umbrello UML	ActionScript„Ada, C++, C#, D, IDL, Java, JavaScript, MySQL and Pascal.

3.3 Collaboration

Table 6 shows the UML Tools and their support for user collaboration, which is considered in terms of multi-user support, task management, communication, and versioning. Concerning the multi-user support, all the tools that support collaboration enable both the synchronous and asynchronous collaborations. So, practitioners can work on the same UML diagrams at the same time (i.e., synchronously), or they may choose to work offline (i.e., asynchronously) on the UML diagrams. Any changes made by the practitioners are merged by the UML tools, and in case of conflicts, the warning messages will be given to the practitioners. The task management is supported by Visual Paradigm and Enterprise Architect, which offer the necessary facilities for determining the design and development tasks, assigning the tasks to the stakeholders, decomposing tasks into sub-tasks, and prioritising the tasks. The communication facility is provided only by the Visual Paradigm, MagicDraw, and Enterprise Architect tools, which enable the stakeholders to communicate with each other via some messaging mechanisms and leave comments on the model diagrams. Lastly, the UML tools that enable the model versioning offer their own (sub-)tools that allow for managing the model versions securely and effectively. The versioning tools provide many facilities such as secure authenticated version access, high-performance versioning, working offline, and web access. Note that Eclipse Papyrus adapts the Git version control system in the Eclipse environment rather than being supported with a new tool. The tools store the versions in two ways, i.e., either using a server or cloud. While Visual Paradigm and MagicDraw offer cloud-based repositories, the other tools with the versioning offer server-based repositories. So, Visual Paradigm is the only UML tool that supports all those requirements at the same time. Note that StarUML, ArgoUML, BoUML, Obeo UML Designer, and Umbrello UML do not support any of the collaboration requirements.

3.4 Code-generation

Table 7 shows the UML tools that support code generation and gives the programming languages used for code generation. So, according to the results, all the UML tools enable the automatic code generation from UML models. Java is the top popular programming language that is supported by all the UML

Table 8. The UML tools that support extensibility

UML Tools	Extensibility
Visual Paradigm	Plug-in development in Java API
MagicDraw	Plug-in development in Open Java API
StarUML	Plug-in in COM-compatible languages
Modelio	Module definition
Enterprise Architect	ActiveX Plug-in
ArgoUML	No
BoUML	Plug-in in Java/C++
Obeo UML Designer	No
Eclipse Papyrus	No
Rational Rhapsody	Plug-in Java API
Umbrello UML	No

tools, and it is followed by C++. Visual Paradigm offers the greatest number of alternative programming languages for code generation.

3.5 Extensibility

Table 8 shows the UML tools' support for extensibility. So, ArgoUML, Obeo UML Designer, Eclipse Papyrus, and Umbrello UML cannot be extended with some new features. Visual Paradigm, MagicDraw, Rational Rhapsody, and BoUML offer Java APIs for developing plug-in applications that can be used for extending the tools. StarUML offers an API for developing plug-ins in COM-compatible languages (e.g., C++, C#, Visual Basic, etc.). Modelio offers its own module notation for extending the UML tool, where modules are developed in the combination of Java and XML languages. To facilitate developing tool extensions, Modelio also provides template code. Lastly, Enterprise Architect prompts the practitioners to develop their extensions as ActiveX COM objects in any supporting platform, e.g., Microsoft Visual Studio.

4 Related Work

The literature essentially includes many empirical studies on UML. Most of those approaches focus on the UML language itself. Indeed, some approaches (e.g., [1, 5]) compare UML with other software modeling languages (e.g., architecture description languages), and some (e.g., [2, 3]) compare the UML-based software modeling languages that extend UML for different purposes (e.g., the needs of particular domains, non-functional properties) with each other. However, none of those studies consider the existing UML modeling tools. Among the very few empirical studies that compare the UML modeling tools, none of them consider *(i)* the requirements focussed in this study (i.e., multiple viewpoints, large view management, model analysis, code generation, collaboration, and extensibility) and *(ii)* as many UML tools as is considered in this study.

In [19], Safdar et al. surveyed among the university students to understand how productive the students are in their usage of the three UML tools (i.e., Rational Software Architect, MagicDraw, and Papyrus) for their software modeling. Safdar et al. compared the three tools based on the students' effort required for performing a UML modeling correctly, the tool learnability, the number of clicks required during the modeling, and the memory consumed by the tool.

In [8], Khaled compared four well-known UML tools (i.e., Rational Rose, ArgoUML, MagicDraw, and Enterprise Architect) for three features, which are the support for HTML documentation, UML notation, and reverse engineering.

In [15], Rajoo et al. compared four well-known UML tools (i.e., Dia, UMLet, MagicDraw, and Rational Rose) with regard to their support for the health informatics domain. Rajoo et al. focus on the key properties that health informatics systems are concerned with, which are performance, security, usability, and reliability, and analysed the tools for modeling these properties.

In [4], Cabot et al. compared five different UML tools (i.e., Poseidon, Rational Rose, MagicDraw, Objecteering/UML, Together) to understand how well they support the software code generation and the transformation of the integrity constraints specified in the models into software implementation code.

In [16], Rani et al. analysed four UML tools (i.e., ArgoUML, StarUML, Umbrello UML, and Rational Rose), discussing their features in general and determining the advantages and disadvantages of the tools depending on the supported features (e.g., particular languages for code generation, UML notation support, and formatting options).

5 Discussions and Conclusion

We have analysed 11 well-known UML modeling tools for a sub-set of requirements that we determined using Lago et al.'s framework [10]. The analysis results are expected to be useful for the practitioners and academics who wish to use the UML modeling tools for modeling their software systems in UML. Also, the tool developers may use the results for developing a UML-based software modeling toolset that bridges the gap(s) of the existing tools. The results clarify on the tools' level of support for *(i)* different types of UML diagrams, *(ii)* different software viewpoints, and *(iii)* managing the large software systems. Moreover, the results also give clues about the tool support for model analysis, model-to-code transformation, collaborative modeling, and tool extensibility.

Among the analysed UML tools, Obeo UML Designer, ArgoUML, and Umbrello are accessible as free and open-source. Also, BoUML and Eclipse Papyrus are free but their source-code is not open. The rest of the UML tools considered offer a free evaluation version with limited period of use (mostly 30 days) and only certain basic facilities are made available in the evaluation version.

Enterprise Architect is the only UML tool that supports all the requirements considered in our study. Most of the tools support UML's all static and dynamic diagrams. So, the UML tools (except one or two) allow for choosing among the alternative diagrams depending on their domain constraints. However, many of the UML tools ignore the separation of UML models into different viewpoints for modular and understandable model specifications. The tools (except Visual Paradigm, StarUML, Enterprise Architect, and Rational Rhapsody) lead to the UML models where it is difficult to understand which diagram describe the system from which perspective(s). Managing large UML models is quite popular, supported by more than half of the UML tools. The tools sup-

port sub-diagramming for the model components (e.g., classes, packages, and activities). The tools however do not support sub-diagramming for the connectors (e.g., class associations) - except Visual Paradigm, which might be needed when complex interactions were employed. Concerning the model analysis, the model simulation is supported by MagicDraw, Enterprise Architect, and Rational Rhapsody only. Most of the UML tools support the model validation - except ArgoUML, BoUML, and Umbrello UML. Those UML tools define their well-definedness rules for UML that can be checked automatically and allow for modeling the system requirements in the form of OCL or some other notation and checking the UML models for those requirements. Note here that none of the UML tools enable the formal verification of UML models for proving the model correctness. Another requirement considered is the user collaboration, which is supported by most of the tools in terms of the synchronous/asynchronous user collaborations and the model versioning via their cloud-based or server-based repository systems. ArgoUML, BoUML, Obeo, and Umbrello do not provide any collaboration support. Other collaboration requirements such as task management and an instance messaging system are rarely supported though. The automatic generation of software code from UML models is also supported by all the UML tools, and Java is the top popular programming language. Lastly, many of the UML tools offer the plug-in development support for extending the tools with the new features of their interests.

While the analysis results shed light on the weak and strong points of the well-known UML modeling tools, the results may essentially be considered as the preliminary work that is to be followed by a more comprehensive analysis. Indeed, due to the space limitation, we were not able to consider every single UML modeling tool existing. Moreover, the requirements for the tool analysis can be extended with some other expectations of the practitioners following a survey that may be later on conducted on practitioners with the goal of understanding their expectations from the UML modeling tools.

References

1. Andersson, P., Höst, M.: UML and SystemC – A Comparison and Mapping Rules for Automatic Code Generation, pp. 199–209. Springer Netherlands, Dordrecht (2008). https://doi.org/10.1007/978-1-4020-8297-9$_1$4, https://doi.org/10.1007/978-1-4020-8297-9_14
2. Bendraou, R., Jezequel, J., Gervais, M., Blanc, X.: A comparison of six uml-based languages for software process modeling. IEEE Transactions on Software Engineering **36**(5), 662–675 (Sept 2010). https://doi.org/10.1109/TSE.2009.85
3. Brisolara, L., Becker, L., Carro, L., Wagner, F., Pereira, C.E., Reis, R.: Comparing high-level modeling approaches for embedded system design. In: Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005. vol. 2, pp. 986–989 Vol. 2 (Jan 2005). https://doi.org/10.1109/ASPDAC.2005.1466505
4. Cabot, J., Teniente, E.: Constraint support in mda tools: A survey. In: Rensink, A., Warmer, J. (eds.) Model Driven Architecture – Foundations and Applications. pp. 256–267. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)

5. Dori, D., Wengrowicz, N., Dori, Y.J.: A comparative study of languages for model-based systems-of-systems engineering (mbsse). In: 2014 World Automation Congress (WAC). pp. 790–796 (Aug 2014). https://doi.org/10.1109/WAC.2014.6936160
6. Eriksson, H.E., Penker, M., Lyons, B., Fado, D.: UML 2 Toolkit. Wiley Publishing (2003)
7. Fowler, M., Scott, K.: UML distilled - a brief guide to the Standard Object Modeling Language (2. ed.). notThenot Addison-Wesley object technology series, Addison-Wesley-Longman (2000)
8. Khaled, L.: A comparison between uml tools. In: 2009 Second International Conference on Environmental and Computer Science. pp. 111–114 (Dec 2009). https://doi.org/10.1109/ICECS.2009.38
9. Kruchten, P.: The 4+1 view model of architecture. IEEE Software **12**(6), 42–50 (1995). https://doi.org/10.1109/52.469759, http://dx.doi.org/10.1109/52.469759
10. Lago, P., Malavolta, I., Muccini, H., Pelliccione, P., Tang, A.: The road ahead for architectural languages. IEEE Software **32**(1), 98–105 (2015). https://doi.org/10.1109/MS.2014.28, http://dx.doi.org/10.1109/MS.2014.28
11. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: A survey. IEEE Transactions on Software Engineering **99** (2012)
12. Ozkaya, M.: Analysing uml-based software modelling languages. Journal of Aeronautics and Space Technologies **11**(2), 119–134 (2018), http://www.rast.org.tr/JAST/index.php/JAST/article/view/326
13. Ozkaya, M.: Do the informal & formal software modeling notations satisfy practitioners for software architecture modeling? Information & Software Technology **95**, 15–33 (2018). https://doi.org/10.1016/j.infsof.2017.10.008, https://doi.org/10.1016/j.infsof.2017.10.008
14. Ozkaya, M., Kose, M.A.: Sawuml – uml-based, contractual software architectures and their formal analysis using spin. Computer Languages, Systems Structures **54**, 71 – 94 (2018). https://doi.org/https://doi.org/10.1016/j.cl.2018.04.005, http://www.sciencedirect.com/science/article/pii/S1477842417301550
15. Rajoo, M., Noor, N.M.M.: Important evaluation factors of uml tools for health informatics. Journal of Telecommunication, Electronic and Computer Engineering **9**(3), 191–195 (2017)
16. Rani, T., Garg, S.: Comparison of different uml tool - tool approach. International Journal Of Engineering And Computer Science **2**(6), 1900–1908 (June 2013)
17. Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley Professional, 2 edn. (2011)
18. Rumbaugh, J.E., Jacobson, I., Booch, G.: The unified modeling language reference manual. Addison-Wesley-Longman (1999)
19. Safdar, S.A., Iqbal, M.Z., Khan, M.U.: Empirical evaluation of uml modeling tools–a controlled experiment. In: Taentzer, G., Bordeleau, F. (eds.) Modelling Foundations and Applications. pp. 33–44. Springer International Publishing, Cham (2015)