Survey on the Applicability of Textual Notations for the Unified Modeling Language

Stephan Seifermann and Henning Groenda

FZI Research Center for Information Technology, Software Engineering
Haid-und-Neu-Str. 10-14, Karlsruhe, Germany
{seifermann,groenda}@fzi.de

Abstract. The Unified Modeling Language (UML) is the most commonly used software description language. Today, textual notations for UML aim for a compact representation that is suitable for developers. Many textual notations exist but their applicability in engineering teams varies because a standardized textual notation is missing. Evaluating notations in order to find a suitable one is cumbersome and guidelines found in surveys do not report on applicability. This survey identifies textual notations for UML that can be used instead of or in combination with graphical notations, e.g. by collaborating teams or in different contexts. Additionally, it rates the notation's applicability with respect to UML coverage, user editing experience, and applicability focused on engineering teams. Our results facilitate the otherwise unclear selection of a notation tailored for specific scenarios and enables trade-off decisions. We identified and characterized 21 known notations and 12 notations that were not covered in previous surveys. We used 20 categories to characterize the notations. Our findings show that a single notation does not cover more than 3 UML diagram types (mean 2.6), supports all surveyed state of the art editing features (only one notation supports all), and fits into existing tool chains.

Keywords: UML, Textual Notation, Survey, Editing Experience

1 Introduction

The Unified Modeling Language (UML) has become the de-facto standard for describing software systems. The specification defines a graphical but no textual notation for fully representing the model. Researchers such as Spinellis [34] argue that textual notations provide compact and intuitive alternatives. For instance, Erb represents UML activity diagram-like service behavior specifications textually in a developer-friendly way and more compact than graphics.

The absence of a standard leads to many textual notations that do not fully cover UML modeling partially but focus on supporting documentation, being compact, or serving as input for code generation. They largely differ in syntax, UML coverage, user editing experience, and applicability in engineering teams.

The latest surveys covering textual UML notations were performed by Luque, et al. [24,22,23]. The former two [24,22] focus on the accessibility of UML for blind students in e-learning and classrooms, respectively. The latter [23] surveyed tools for use-case and class diagrams used in industry at 20 companies in the state of Sao Paolo (Brazil). All surveys target notations used in practice. The literature studies rely on existing studies on the accessibility domain but do not search for scientifically published notations. The survey of Mazanec and Macek [25] focuses on textual notations in general but is a few years old and covers few notations. It does not represent the current development state and available variety of notations and modeling environments. The surveys illustrate the variety of specialized textual notations but do not analyze the editing experience in an objective way. The editing experience is, however, crucial for engineering teams and is hard to survey. The latter degrades the selection quality because it limits the amount of notations to be tested because of time constraints.

The contribution of this survey is the identification and classification of textual UML notations including the user experience. Engineering teams can use the classification for identifying appropriate notations for their usage scenarios. The classification scheme is tailored to support this selection. This survey examines usability of notations with respect to their syntax, editors, and modeling environment. Usability in realistic scenarios is determined by covered diagram types, supported data formats for information exchanges such as XMI, and synchronization approaches with other notations. It additionally evaluates whether non-necessary parts of the notation can be omitted. This support for sketching models eases low-overhead discussion and brainstorming. For instance, the UML specification allows to omit the types of the class attributes.

This survey extends the trade-off selection discussion and includes two additional notations with respect to our previously published survey [33]. The two new notations stem from the latest survey from Luque et al. [24] that we became aware of in the meantime. This adds two new notations that we reviewed with the same 20 categories covering applicability in engineering teams. Considering that survey, we identified 12 notations not covered in surveys of other authors. We rewrote and extended the discussion to identify drawbacks of the notations that limit applicability. This allows practitioners to focus their notation evaluations on critical aspects. Tool vendors can identify unique features. Researchers can develop approaches on how to make notations more applicable.

The remainder of this survey is structured as follows: Section 2 describes the survey's review method by defining objectives and the review protocol consisting of three phases. Section 3 describes the classification scheme based on the defined objectives. Section 4 presents the extended analysis results in terms of classified textual notations. Section 5 covers our new extensive discussion of the findings and discusses the validity of the results. Finally, Section 6 concludes the paper.

2 Review Method

The review process follows the guidelines of Kitchenham and Charters [20] for structured literature reviews (SLR) in software engineering based on the guidelines in the field of medical research. Their guidelines cover the planning, conduction, and writing of reviews. Planning involves defining research objectives and creating a review protocol describing the activities in each review step.

The following sections describe our implementation of the SLR and mapping to the proposed method. The results of our search activities are documented and available for reproducibility at `http://cooperate-project.de/CCIS2016`.

2.1 Objectives

Our objectives are to determine each notation's (O1) coverage of the UML, (O2) user editing experience and (O3) applicability in an engineering team. The reasoning requires an analysis of the textual notations and of the modeling environments. Section 3 presents the detailed classification scheme based on the objectives and instructions on information extraction from literature.

2.2 Review Protocol

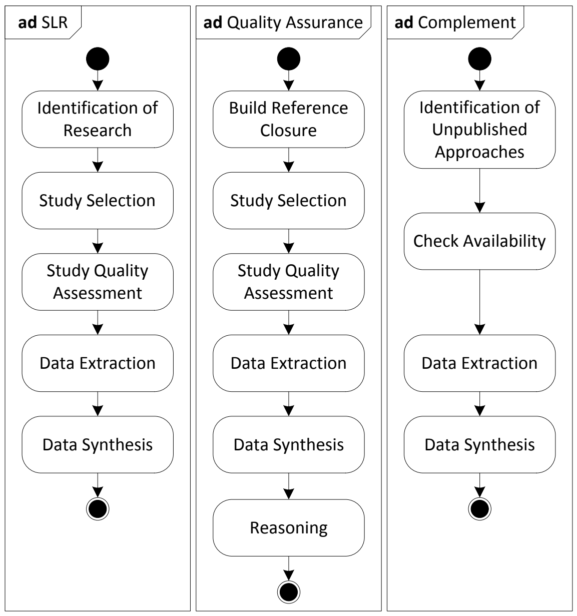

Fig. 1. The three phases of the review conduction process used in this survey.

Figure 1 shows an overview of our review protocol. We distinguish three phases during the conduction: classic SLR, Quality Assurance and Complement.

The classic *SLR* follows the guidelines of review conduction by Kitchenham, et al. [20]. We extend the SLR with two additional phases in order to increase the quality of the results and to take notations into account that are mainly used in (industrial) practice: The *Quality Assurance* phase focuses on incoming and outgoing literature references as suggested by the Snowballing search approach [40]. In contrast to the original proposal, we use Snowballing only to cross-check our SLR search strategy. The *Complement* phase focuses on textual notations that are available in practice but are not scientifically published.

2.3 Phase 1: SLR

Reviews according to [20] consist of the five activities marked as *SLR* in Figure 1.

The *Identification of Research* describes the search strategy for collecting literature. We chose a keyword-based search approach using the search engines ACM Digital Library, IEEExplorer, CiteSeer, ScienceDirect, SpringerLink and Google Scholar. These search engines cover relevant journals and are suggested by Kitchenham and Charters for the software engineering domain. We did not include EI Compendex and Inspec as we could not query these search engines without subscriptions. Their focus is on high-qualitative entries and metadata and they do not belong to a not-covered established publishing authority. We are confident that the selected search engines and their metadata are sufficient.

We defined a set of keywords T for identifying textual notations and another one U for identifying the usage of UML. Table 1 presents both sets as variations of our original terms *textual notation*, and *UML*. They are based on commonly used terminology in the modeling domain. A search query is given by $\vee_{t_i} \wedge \vee_{u_i}$ with $t_i \in T \wedge u_i \in U$. The query enforces the exact matching of keywords. It considers abstracts and titles because this restricts the search to literature that focuses on textual notations for UML. Google Scholar has API restrictions that limit queries on abstracts to papers that have been released at most one year ago. This restriction does not apply to our title-based search. We restrict ScienceDirect queries to computer science papers. We implemented a search on the SpringLink results enabling keyword identification in the abstract. After collecting the results of all search engines, we merge them and filter duplicates.

Table 1. Keyword groups used in search queries.

Group	Keywords
Textual T	CTS, textual modeling, textual modelling, text-based modeling, text-based modelling, textual notation, text-based notation, textual UML, text-based UML, textual syntax
UML U	UML, unified modeling language, unified modelling language

Study Selection covers a rough screening based on titles and abstracts to allow spending more time on relevant literature. We focus on textual notations for graphical parts of the UML specification [27, p. 683]. We exclude all textual notations only extending UML or its elements rather than expressing UML itself. We exclude all notations that are not related to UML. We exclude notations not intended for human usage such as data transfer containers, e.g. XMI serialization [28]. We include a) primary papers describing a single textual notation, and b) secondary survey-like papers including their references as primary sources.

The *Study Quality Assessment* considers title, abstract, and the content of the full paper. We decide on in-/exclusion of the remaining papers in this step.

Data Extraction is the process of determining the information required to judge about the fulfillment of the objectives. Section 3 shows the analyzed features of the notations, their hierarchy, and individual decision basis in detail. We reason on the modeling environment based on information found directly in literature, implemented prototypes, prototype websites, and source code. We identify prototypes, their website, and the source code by: a) following links in the papers, b) mining the website of the institute or company of the authors, c) and searching for the name of the notation (full name and abbreviation if used) via the Google search engine and on Githuband visit the first one hundred search results.Data extraction takes place for the declared primary editor. If there is more than one prototype, we use the declared primary editor and an IDE-integrated editor. We assume the latter to profit from advanced accessibility features of the IDE. If there are editors for several IDEs, we decide in favor of the Eclipse-based one because Eclipse is open source, highly extensible, and offers many accessibility features[1].

Data Synthesis summarizes the information. We show and summarize the analysis results according to the classification given in Section 3.

2.4 Phase 2: Quality Assurance

The Quality Assurance phase is based on the Snowballing approach [40] of Wohlin for literature identification. Wohlin suggests starting with an initial set of relevant literature and including relevant forward and backward references. We do not use Snowballing as primary source for relevant literature because its quality heavily depends on the initial literature set as described by Wohlin. Instead, we accept the overhead of a prior SLR phase with broad search terms and use Snowballing to verify the quality of our SLR phase as described below.

Build Reference Closure determines the completeness of results from the SLR phase. We collect all directly referenced and referencing literature for the analyzed papers. We derive the referenced literature from the references section of the paper. We use Google Scholar to determine incoming references.

The *Study Selection* and *Study Quality Assessment* from phase SLR are applied to identify additional notations.

We perform *Data Extraction* on selected papers as in the SLR phase and add the notation to our database.

Data Synthesis summarizes the information as carried out in the SLR phase.

Reasoning addresses why newly identified notations have been missed in the SLR phase. Section 5 presents the results. This phase is different from Wohlin's Snowballing approach and allows verifying the quality of our SLR phase.

[1] `https://wiki.eclipse.org/Accessibility`

2.5 Phase 3: Complement

Identification of Unpublished Approaches focuses on textual notations that are available in practice but are not scientifically published. We use the Google search engine to identify the top 5 pages for 'UML textual notation', 'UML textual notations', 'UML textual notations list'. We mine the resulting websites to identify new approaches. We follow the links from the identified websites looking for notations or comparisons of notations.

Additionally, we search for unrecognized scientific surveys or notation comparisons. We perform a full-text search via Google Scholar with the names of the three most popular non-scientific notations. We assume that recent surveys including non-scientific notations cover them and thereby will be included in the search results. We determine a notation's popularity by querying Google with the name of the notation and comparing the announced results with the amount of other notations. We only included notations that claim to relate to the UML.

In *Check Availability*, we filter all potential notations with dead links.

We perform *Data Extraction* for new notations, analyze the information, and add the notation to our database.

Data Synthesis summarizes the information as carried out in the SLR phase.

3 Classification

This section presents the classification and information extraction goals derived from the three objectives presented in Section 2.1. The objectives cover aspects of what can be edited based on the textual notation definition (O1, O2) as well as how it can be edited based on modeling environments (O2, O3). We use feature modeling to represent the evaluation classes, their hierarchy, and possible values. The resulting overview is depicted in Figure 2. The features themselves and how their values are evaluated for the notations are presented in the following.

Each *Textual Notation* is defined by a Language (O1, O2) and an optional Implementation (O2, O3) in a modeling environment.

The *Implementation* is optional and covers all aspects with respect to a modeling environment for a notation. It can have Recent Activity (O3), a License (O3), and can support Change Propagation (O2, O3) between different notations, data Format Exchange (O2), and Editor (O2) features.

We divide the classification of the implementation into two parts for a better overview: integration aspects, and the editor itself. The former covers the features relevant for integrating an implementation into a tool chain. The latter covers the editing experience of the editors.

The following subsections will cover the language, integration, and the editor in that order.

3.1 Language

The mandatory *Language* definition describes the language's syntax. It consists of UML Support (O1) for diagram types, can have Sketch Support (O2), integrated Layout Information (O2), and be Similar to UML Graphics (O2).

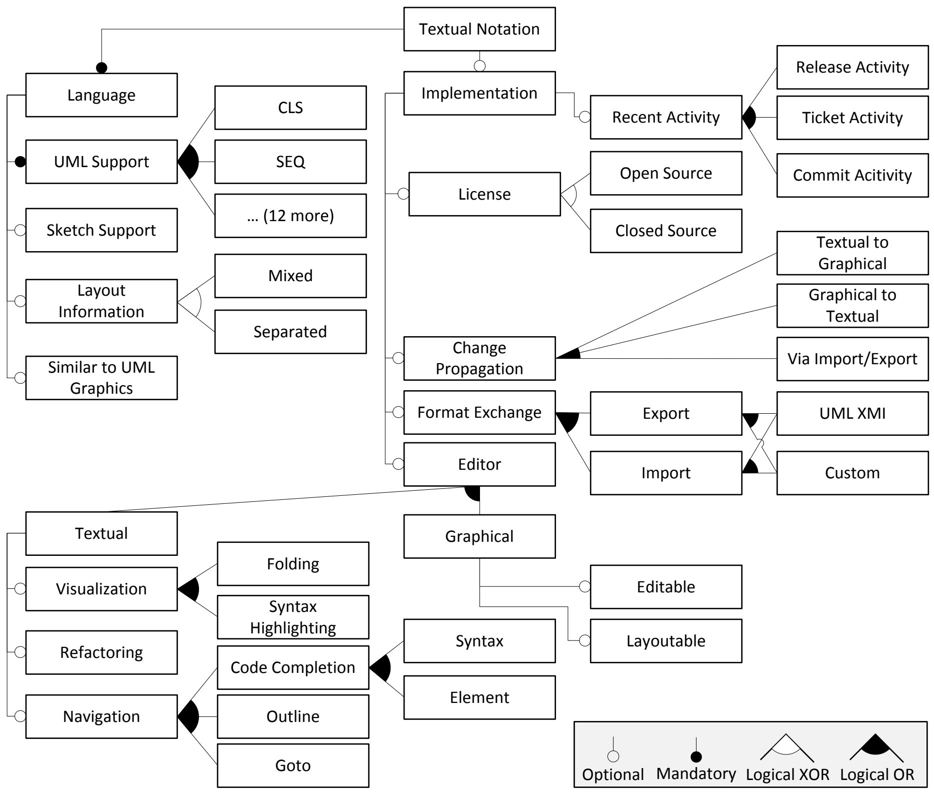

Fig. 2. Feature model for analyzed characteristics and their hierarchy.

UML Support is mandatory and describes the supported UML diagram types. At least one type has to be supported. A type is supported if the documentation states it to be supported or the modeling environment allows the creation of a corresponding type. The considered diagram types are based on the UML specification [27, p. 682]. The abbreviations are based on the official abbreviations from [27, p. 682], or self-made if there is no official one: Activity Diagram (ACT), Class Diagram (CLS), Communication Diagram (COM), Component Diagram (CMP), Composite Structure Diagram (COS), Deployment Diagram (DEP), Interaction Overview Diagram (INT), Object Diagram (OBJ), Package Diagram (PKG), Profile Diagram (PRO), Sequence Diagram (SEQ), State Machine Diagram (STM), Timing Diagram (TIM), and Use Case Diagram (UC).

Sketch Support is optional and can ease the notation's usage during discussions. Discussions benefit from quick interaction. Formal full-fledged modeling can extend the interaction time. There is support if only mandatory elements of UML's abstract syntax are required.

Layout Information is optional and states if the textual model can contain graphical layout information. This information allows to improve graphical presentations of textual statements. The information is irrelevant to describe the model itself. The interpretation is difficult as only graphic notations illustrate graphical positions. The information can be either *Mixed* with model elements or kept *Separated*. It is marked as *Mixed* if at least one element has mandatory layout information.

Similar to UML Graphics is optional and denotes if ASCII art memes graphical elements such as arrows in the textual notation. For instance, the characters `<>-->` are similar to the UML graphical representation for an aggregation. This can work well for people knowing the graphical representation but has adverse effects when typing or for people using accessibility tools like Braille displays. A notation is marked as similar if there is at least one ASCII art mapping.

3.2 Integration

The integration covers all features that are relevant for integrating an implementation into a tool chain. Such a decision is based on the costs, extensibility, support, maintainability and compatibility to existing tools. The following features cover these aspects in more detail.

The *Recent Activity* is optional and indicates the support status. In contrast to a maintained project, a discontinued project will not receive bugfixes and might be incompatible to recent software such as new versions of an IDE. We determine three activity dates that allow judging project activity. One of them has to be identifiable: *Release Activity* relates to the date of the last release. A release can be a proper release, snapshot, or nightly build. *Ticket Activity* is determined by the date of the most recently closed ticket. *Commit Activity* is given if we can determine the most recent commit.

The *License* is optional and can be crucial for using and maintaining the modeling environment. Open Source licenses allow own bug fixing and the development of extensions and adaptations. The individual requirements for a license depend heavily on the usage context of the modeling environment. An expert review is required to check for a notation of interest if it applies to the own use case. We therefore differentiate solely between Open Source and Closed Source licenses. We rely on the list of the Open Source Initiative [29]. If the license is listed on their website, we treat the project as open source. All other licenses are considered *Closed Source*.

Change Propagation can be supported and addresses transferring changes from one notation into another. The modification in the modeling environment for a textual notation can therefore result in an according change in a graphical notation of the same content. This targets a consistent view of the content and allows different team members to work with different notations during discussion. This can mean updates in real-time for close collaboration or based on exporting and importing models in different environments. We consider the three cases: Textual to Graphical, Graphical to Textual, and Via Import/Export propagation. *Textual to Graphical* and *Graphical to Textual* apply if the modeling environment includes a textual and a graphical editor. We consider it supported if changes in one editor are reflected in the other one. *Via Import/Export* applies if there is an import or export functionality and notations can be updated sequentially. It is marked if it provides import and export function for UML models in the standardized XMI data format.

Data *Format Exchange* is optional and allows integrating the modeling results into other tools or existing tool chains. We only consider fully-automated exchange procedures provided by the implementation itself. We do not consider other procedures such as the error-prone manual translation between notations or tools that is usually done by assistants. A modeling environment can support the *Import* or *Export* of a different set of data formats. This feature can have the value *UML XMI* as standardized UML data exchange and can list *Custom* formats supported by the tools. The values are selected based on the documentation or file extensions provided in the editing environment.

3.3 Editor

Editor categorizes properties related to user input, interaction, and presentation. They can be Textual (O2), Graphical (O2) or both. An editor is considered textual if it contains only text and no graphical elements. Text coloring may be used. This ensures that textual editors are accessible by accessibility techniques such as screen readers. Otherwise, it is treated as Graphical.

Textual editors address several features to increase user experience and accessibility. A textual editor can support Visualization (O2), Refactoring (O2) of the model, and user Navigation (O2) within the model. Previous surveys did not focus on the editing experience in detail. Therefore, we selected the features according to our objectives.

A *Visualization* is optional and allows focused presentation of content by means of information hiding. It can support Folding (O2), and Syntax Highlighting (O2).

Folding (un)hides selected partitions of the model, eases comprehension for complex models and focused presentation. It is selected if there is at least one partition in a model that can be hidden or shown based on the editor's UI.

Syntax Highlighting highlights keywords or important structural parts of the model. It eases comprehension and identifying the structure of models. It is selected if colors or text formats highlight at least one keyword of the language.

Refactoring is optional and addresses batch changes to the model. For instance, all occurrences of a model element can be replaced with another one in one single step instead of using a manual search and replace approach. This feature exists if there is at least one supported refactoring.

Navigation is optional and addresses navigation to model elements and providing an overview to users. There can be support for Code Completion (O2), overviews on model elements by Outline (O2), and model element navigation by Goto (O2). Navigation is selected if at least one of its child features is selected.

Code Completion is optional and provides completion of a language's syntax or referenced model elements. It can provide hints on keywords of the Syntax or model Elements allowed at the current position. It aids users in specifying correct models and speeds up changes. We consider two types of values: *Syntax*-based and *Element*-based completion. They are selected if there is at least one corresponding code completion feature in the editor.

Outline is optional and provides an overview of the elements in a model. This can include their hierarchical structure. It is selected if there is at least a list of all top-level elements in a model depicted in the editor.

Goto is optional and allows direct navigation or jumps to specific model elements. This eases comprehension and look-up of elements. It is selected if there is navigation or jump support for at least one element type. It is included if it is directly in the textual notation and excluded if its only in the Outline.

Graphical editors are optional and allow displaying and editing graphical version of the models. There are many advanced graphical UML editors available based on the formal UML specification. [18] gives a good overview in his survey of interoperability of UML tools. [38] illustrates the features of various UML tools. There are many comparisons between few selected tools such as IBM Rational Software Architect, MagicDraw, and Papyrus in [32] or between Rational Rose, ArgoUML, MagicDraw, and Enterprise Architect in [19]. This survey focuses on the synchronization aspect with textual languages and their editors (O3). Our categories show if the editor is mainly a pure static presentation of the model or allows interactions. We distinguish for Graphical editors if their content is Editable (O2) and Persistable (O2). This feature is selected if there is a graphical presentation of the model in the modeling environment.

Editable is optional and denotes if the graphical content can be modified, e.g. a user can rename elements. This feature is selected if at least some elements in the graphical editor can be modified.

Layoutable is optional and denotes if modifications to the graphical layout, e.g. the position of model elements, can be done. Users can structure the graphical representation in this way. This feature is selected if elements can be moved.

4 Analysis Results

This chapter presents the analysis results for all notations. Table 2 and Table 3 provide an overview and show the determined characteristics for all notations. The following paragraphs provide short notation descriptions. They point out features or provide comments, which are not already covered by the overview.

Alf [26] has been specified by the OMG and is the UML action language. It is based on Foundational UML (fUML). There is no official editor implementation.

Alloy [16] is a model finder and solver based on the Z notation [15] instead of UML. The author compares it to UML in sections 4.1 and 6.4 and states that "Alloy is similar to OCL, the Object *Constraint* Language (OCL) of UML"[2]. It provides a graphical and textual notation but no support for any UML diagrams. It has a MIT license and does not provide access to source code.

AUML [39] is an extension to UML SEQ diagrams. Winikoff defined a textual notation for AUML that has been included in the Prometheus Design Tool[3]. It provides a PNG export but no mechanism to import or export a model.

Ckwnc [36] is a web editor that allows specifying UML SEQ diagrams with a programming language-like syntax. Users can export graphics.

Clafer [41] is a modeling language for CLS diagrams and constraints. The online tool[4] provides no graphical view but offers a GraphViz export.

DCharts [10] specifies a meta-model in AToM^3 [5] and a graphical and textual notation. The textual notation is the leading one and the graphical implemented only partially [10, p. 35]. No tool or files could be found actually implementing the theoretical concept. We could not find an advanced textual editor with collaboration features for the self-defined language. The publication claims that there is a transformation from the meta-model to UML state charts.

Earl Grey [25] is a proof of concept for an accessible textual notation. The Eclipse implementation creates a model during editing but there is no export.

EventStudio [9] is a commercial tool suite for modeling object and message flows. It supports SEQ, STM, and UC diagrams and can generate images. The images, however, do not correspond to the official UML graphical syntax.

HUTN [35] is an OMG standard for text-based representation of MOF-based meta-models, which covers the UML meta-model. Humans can use it easier than XMI. There is no official reference implementation of an editor.

IOM/T [7] allows specifying protocols for agent communication. It covers AUML [39] sequence diagrams partially, which we consider as SEQ support. The notation seems to consist of two papers, the latest in 2007.

MetaUML [11] is a DSL leveraging TeX in the background. It creates graphics in UML style but no UML models.

[2] `http://alloy.mit.edu/alloy/faq.html`
[3] `https://sites.google.com/site/rmitagents/software/prometheusPDT`
[4] `http://t3-necsis.cs.uwaterloo.ca:8094`
[5] `http://atom3.cs.mcgill.ca/`

Table 2. Language and textual editor implementation characteristics of analyzed textual UML notations. Characteristics are: not extractable (-), given (✓), or not given (×). Layout information is: mixed (*m*) or separated (*s*).

	Language				Textual Editor						
					Vis.		Navigation				
Notation	UML Support	Sketch Support	Layout Information	Graph. Similarity	Syntax Highlighting	Folding	Compl. (Syntax)	Compl. (Element)	Outline	Goto	Refactoring
Alf	CLS, ACT, PKG	✓	×	×	-	-	-	-	-	-	-
Alloy	×	×	×	×	✓	×	×	×	×	×	×
AUML	SEQ	×	×	×	✓	×	×	×	×	×	×
AWMo	CLS	×	×	×	-	-	-	-	-	-	-
blockdiag: seqdiag, actdiag	SEQ, ACT	✓	m	✓	-	-	-	-	-	-	-
Clafer	CLS, OBJ	×	×	×	✓	×	×	×	×	×	×
cwknc	SEQ	×	×	×	✓	×	×	×	×	×	×
Dcharts	STM	✓	×	×	-	-	-	-	-	-	-
Earl Grey	CLS, SEQ, STM	×	×	×	✓	✓	✓	✓	✓	✓	✓
EventStudio	SEQ, STM, UC	×	s	✓	✓	×	×	×	×	×	×
Finite State Machine Diagram Editor	STM	✓	×	×	✓	×	×	×	×	×	×
HUTN	all	✓	×	×	-	-	-	-	-	-	-
IOM/T	SEQ	×	×	×	-	-	-	-	-	-	-
js-sequence-diagrams	SEQ	✓	m	✓	-	-	-	-	-	-	-
MetaUML	CLS, STM, ACT, UC, CMP, PKG	✓	m	×	-	-	-	-	-	-	-
modsl	CLS, COM	✓	×	×	✓	✓	✓	×	✓	✓	✓
Nomnoml	CLS, OBJ, STM, UC, PKG	✓	m	✓	-	-	-	-	-	-	-
pgf-umlcd	CLS	✓	m	×	✓	×	×	×	×	×	×
pgf-umlsd	SEQ	✓	m	×	✓	×	×	×	×	×	×
PlantUML	CLS, OBJ, SEQ, STM, ACT, UC, CMP, DEP	✓	m	✓	-	-	-	-	-	-	-
Quick Sequence Diagram Editor	SEQ	✓	×	×	-	-	-	-	-	-	-
TCD	CLS	✓	×	✓	-	-	-	-	-	-	-
TextUML	CLS, STM	×	×	×	✓	×	✓	×	✓	×	×
tUML	CLS, STM, COS	×	×	×	✓	✓	×	×	✓	✓	×
txtUML	CLS, STM, ACT	×	×	×	✓	✓	×	×	✓	✓	×
UML/P	CLS, OBJ, SEQ, STM, ACT	✓	s	✓	✓	✓	✓	×	✓	✓	×
UMLet	CLS, OBJ, UC, PKG	✓	m	×	✓	×	×	×	×	×	×
UMLGraph	CLS, SEQ	✓	m	×	-	-	-	-	-	-	-
uml-sequence-diagram-dsl-txl	SEQ	✓	m	✓	✓	✓	✓	×	✓	✓	×
Umple	CLS, STM, COS	✓	s	✓	✓	×	×	×	×	×	×
USE	CLS	×	s	×	✓	×	×	×	×	×	×
WebSequenceDiagrams	SEQ	✓	m	✓	-	-	-	-	-	-	-
yUML	CLS, ACT, UC	✓	×	✓	-	-	-	-	-	-	-

Table 3. Implementation characteristics (without textual editor) of analyzed textual UML notations. Characteristics are: not extractable (-), given (✓), or not given (×). The License is: open (*O*) or closed (*C*) source.

				Graph. Editor		Format Exchange	
Notation	Recent Activity	License	Change Propag.	Editable	Layoutable	Export	Import
Alf	-	-	-	-	-	-	-
Alloy	2015	O	×	×	✓	dot, xml	als
AUML	2014	-	T2G	-	-	png	×
AWMo	2013	C	T2G,G2T	-	-	×	×
blockdiag: seqdiag, actdiag	2015	O	T2G	-	-	png, svg, pdf	×
Clafer	2015	O	×	-	-	own, Python Z3, Choco JS, alf, dot	×
cwknc	2013	O	T2G	-	-	png	×
Dcharts	-	-	-	-	-	-	-
Earl Grey	2012	O	×	-	-	×	×
EventStudio	2016	C	T2G	-	-	pdf, emf, xml, html	×
Finite State Machine Diagram Editor	2015	O	T2G,G2T	✓	×	own	own
HUTN	-	-	-	-	-	-	-
IOM/T	-	-	-	-	-	-	-
js-sequence-diagrams	2015	O	T2G	-	-	svg	×
MetaUML	2015	O	T2G	-	-	×	×
modsl	2009	O	T2G	-	-	png, jpg	×
Nomnoml	2015	C	T2G	-	-	png	×
pgf-umlcd	2015	O	T2G	-	-	×	×
pgf-umlsd	2015	O	T2G	-	-	×	×
PlantUML	2015	O	T2G	-	-	uml, svg, eps, txt, html	×
Quick Sequence Diagram Editor	2015	O	×	-	-	pdf, (e)ps, svg, swf, emf, gif, jpg	×
TCD	-	-	IE	-	-	uml	uml
TextUML	2015	O	×	-	-	uml	×
tUML	-	-	T2G,IE	×	×	uml	uml
txtUML	2015	-	T2G	-	-	uml	×
UML/P	-	C	T2G	✓	×	×	×
UMLet	2015	O	×	✓	✓	bmp, eps, gif, jpg, pdf, png	uxf
UMLGraph	2014	O	T2G	-	-	png, svg, emf, ps, gif, jpg, fig	×
uml-sequence-diagram-dsl-txl	2009	×	T2G	-	-	xml, Code	×
Umple	2015	O	T2G,G2T	✓	✓	uml, tuml, uxf, als, use, emf, code, yUML	×
USE	2015	O	T2G	✓	×	pdf	×
WebSequenceDiagrams	-	×	T2G	-	-	×	×
yUML	-	-	T2G	-	-	png, pdf, jpg, json, svg	×

modsl[6] is a text to diagram sketch tool based on Java code specifications. The proposed default editing environment is Eclipse. It creates graphics in UML style but no UML models.

pgf-umlcd[7] and pgf-umlsd[8] are both based on PGF/TikZ. They leverage TEX interpreters. This has a major influence on its syntax and structure. They create graphics in UML style but no UML models.

PlantUML [31] is a textual notation to diagram tool. CLS diagrams can be exported as UML files for the StarUML and ArgoUML tools. Imports and synchronization mechanisms are not available. There are various standalone and integrated editor implementations.

Quick Sequence Diagram Editor[9] is a text to diagram sketch tool written in Java. It creates graphics in UML style but no UML models.

TCD [37] is an ASCII-art converter for CLS diagrams. It provides conversions from and to UML XMI representations. The implementation is not available.

TextUML [3] exports standard UML models but does not provide a graphical view. Services such as Cloudfier[10] use it as alternative for graphical modeling.

tUML [17] focusses on modeling for validation and verification purposes. The mentioned prototype is not available.

txtUML [5] uses regular Java syntax for modeling. Java Annotations provide additional information. There is no dedicated textual or graphical editor but a Papyrus model can be exported.

UML/P [12] is a textual notation claiming to merge programming and modeling by enriching UML models with Java expressions. The Eclipse plugin provides textual and graphical editors but no import or export.

UMLet[11] [1] is a graphical UML sketch tool. It provides graphical UML shapes. A selected shape is shown in a textual view, which allows to modify the element. The textual view covers only the selected element. It create graphics in UML style but no UML models.

UMLGraph [34] uses Java source files and customized JavaDoc comments to create diagrams. It creates graphics in UML style but no UML models.

uml-sequence-diagram-dsl-txl[12] is a command-line based text to diagram sketch tool written in the transformation language TXL. The Eclipse IDE plug-in was not available. The table lists the mentioned features of the guide[13]. It creates graphics in UML style but no UML models.

Umple [21] is a model-to-code generator with textual notations. UML elements not relevant for code generation such as aggregations are omitted. The online tool synchronizes the textual and graphical notation.

---

[6] `https://code.google.com/p/modsl/`
[7] `https://github.com/xuyuan/pgf-umlcd`
[8] `https://code.google.com/p/pgf-umlsd`
[9] `http://sdedit.sourceforge.net/`
[10] `http://doc.cloudfier.com/creating/language/`
[11] `www.umlet.com`
[12] `http://www.macroexpand.org/doku.php`
[13] `http://www.txl.ca/eclipse/TXLPluginGuide.pdf`

USE [41] aims for specifying systems with including OCL constraints. The official tool does not provide an editor but textual and graphical views.

AWMo [4][14] is a Web application targeting the collaboration of blind and sighted users. The Web tool does not work, there is no included documentation. The characteristics have been determined based on the source code, available presentations and the paper. They define their own simplistic meta-model inspired by CLS diagrams for their proof of concept. Collaboration is realized via store and load mechanism, which maps to Import and Export in the table.

blockdiag[15] has the subprojects seqdiag[16] and actdiag[17]. Both are written in Python and convert textual diagram descriptions to graphics. The syntax is Graphviz's DOT format. The code and release activities are taken from seqdiag only being representative. It creates graphics in UML style but no UML models.

Finite State Machine Diagram Editor and Source Code Generator[18] has an own XML Schema Definition, which defines their textual language called FsmML. Conforming XML documents can be Imported and Exported. Links to model elements are realized via String matching.

js-sequence-diagrams[19] is a text to diagram sketch tool written in Java Script. It is inspired by the commercial WebSequenceDiagram. It parses plain text and can report basic parsing errors. Its shared with an own license title as simplified BSD. It creates graphics in UML style but no UML models.

nomnoml[20] is a text to diagram sketch tool written in Java Script. The syntax is oriented at the graphical UML shapes. It creates graphics in UML style but no UML models.

WebSequenceDiagrams[21] is a text to diagram sketch tool written in Java Script. It creates graphics in UML style but no UML models. A free alternative is js-sequence-diagrams.

yUML [13] is a text to diagram sketch tool. It creates graphics in UML style but no UML models.

5 Discussion of Findings

This section discusses the results of the survey presented in the previous section. We use the results to reason about the applicability in engineering teams and especially identify open points and potential improvements. Additionally, we discuss threats to validity. Section 5.1 focuses on the UML coverage of the found notations. The quality of the provided user editing experience is covered in Section 5.2 and Section 5.3 illustrates the issues of using the notations in

[14] `http://garapa.intermidia.icmc.usp.br:3000/awmo/`
[15] `http://blockdiag.com/en/`
[16] `https://bitbucket.org/blockdiag/seqdiag`
[17] `http://blockdiag.com/en/actdiag/index.html`
[18] `http://www.stateforge.com/`
[19] `https://bramp.github.io/js-sequence-diagrams/`
[20] `https://github.com/skanaar/nomnoml`
[21] `https://www.websequencediagrams.com/`

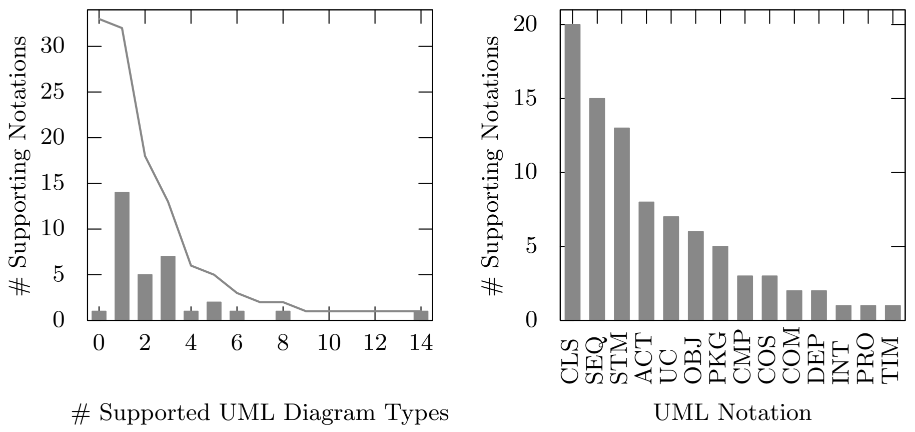

Fig. 3. UML coverage of the surveyed notations.

engineering teams. Threats to internal and external validity are discussed in Section 5.4.

5.1 UML Coverage

The benefit of a high coverage of UML diagram types is a wide range of applicable scenarios. This stems from an increased probability that a diagram type required for a scenario is supported by a notation. The results of our survey with respect to the UML coverage are shown in Figure 3. We discovered that most notations (14 out of 31) only support a single diagram type. This prohibits modeling different aspects of a system such as structure and behavior in a single model. Relations between elements describing different aspects are hard to express. The conceptional HUTN notation supports all diagram types but provides no implementation. In summary, only six notations support four or more diagram types and are, therefore, not restricted to specific application scenarios.

We found that the most supported diagram types are class (20) and sequence diagrams (15) as well as state machines (13). Only few notations support other diagram types as shown in Figure 3. No implementation exists for TIM, PRO, and INT diagrams. The focus of the notations is in line with research on graphical UML usage: Dobing and Parsons [6] as well as Erickson and Siau [8] already identified the class diagram as most commonly used diagram. Both consider sequence diagrams and state machines to be in the top five used diagram types. Reggio et al. [30] achieved similar results and stress that practitioners only use small subsets of the UML elements. As a consequence, vendors of textual notations tailor their notations to support the commonly used UML diagram types and elements in order to facilitate usage. Potential users of the notation have, nevertheless, to carefully check if it supports the elements required for the envisioned usage scenario.

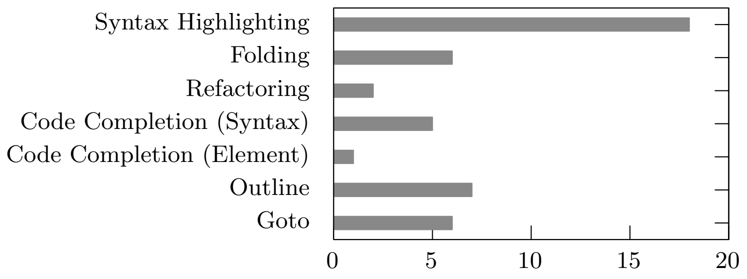

Fig. 4. Amount of notations that support specific textual editor features.

5.2 User Editing Experience

Even if using state of the art editors increases efficiency when working with textual representations, only about half of the notations (18 of 33) provide specialized editors. The support for specific features is visualized in Figure 4.

Basic Features All implementations support syntax highlighting. Around a third of the implementations provide navigation support including outlines and goto links. The same amount provides view customization such as folding. This most probably stems from textual editing frameworks such as Xtext[22] generating these features automatically and without additional effort.

Large Model Handling Most editors, however, lack features required for working with more complex models as given within industrial contexts. We consider code completion and refactorings to be such features because they free the user from knowing the whole model in order to finish their modeling tasks. The support for code completion is twofold: About 25 % of the implementations support code completion for syntactical elements such as keywords but lack support for code completion for elements. Therefore, a user has to remember all usable elements or has to look them up. Only one surveyed editor supports code completion for elements. Only two editors support refactorings such as renaming of elements. The results indicate that the models created with most of the surveyed notations have a limited maintainability: Refactorings ease restructuring or fixing typos but the majority of editors do not support them. Finding applicable elements becomes cumbersome without advanced code completion. Therefore, most notations should be used for small to medium sized models or for simple models without complex relationships between elements.

Coupling with Graphical Notation Seven implementations include a graphical editor to visualize the modeled UML diagram and five implementations allow editing it. In contrast, 22 implementations provide the export of graphics. This indicates that graphical representations are still in the focus but mostly for documentation purposes.

[22] `https://eclipse.org/Xtext`

5.3 Applicability in Engineering Teams

Active development is crucial to get bug fixes and helps when it comes to upgrading the editing environment. About half of our surveyed notations had recent activity in 2015 and later, which means they are actively developed. Unfortunately, only two-thirds of the notations provide a clear license statement, which is crucial for using a notation and its implementation in professional contexts.

The integration in existing tool chains mainly depends on supported import and export formats. Two-thirds (22) of the surveyed implementations provide exports in various formats and only five implementations support imports. Roundtrip engineering, however, requires both features and usage of well-structured formats. Only 8 out of 22 notations provide well-structured formats for exports. Four out of five notations allow the import of well-structured formats. The remainder uses graphics for information exchange. The most prominent well-structured exchange format are serialized UML models. Basically, this means that only four notations are ready for integration in existing tool chains that enable collaborative modeling in various notations, for instance. Information exchange is only partial and requires human intervention to reconstruct missing information including graphical positions, changes in one format or information not expressible in graphics.

5.4 Threats to Validity

We address four common threats to internal validity: incomplete selection, inconsistent measurements, biased experimenter, and incomplete information.

We addressed *incomplete selection* with two additional phases that check the completeness of search results. During the survey, we found a total of 33 textual UML notations. Two notations originate from including the latest survey of Luque et al. [24] in this extended version. These notations are not published scientifically and therefore did not originate from the first two phases that focus scientific notations. In addition, they are not popular enough to be listed in the very first Google search results that we used to find industrial notations. We would, however, have found the survey of Luque et al. in earlier phases if it had been published at the time we conducted our literature study.

We found half of the remaining 31 notations in the SLR phase. In the Quality Assurance phase, we found four new papers and three new notations. The first phase did not reveal three of these papers [16,14,7] because their main contribution was not about a textual UML notation. Therefore, they did not clearly indicate that they also cover a textual UML notation in their title or abstract. The remaining paper [5] is not indexed by the search engines that we used. The major new result of the completion phase was the textual UML tool list [2] provided by Jordi Cabot, a professor with research interests in model-driven software engineering at the ICREA research institute. We found eleven new notations compared to the previous phase. We did not find ten of them in earlier phases because of their scientific focus. The found notations of the third phase have not been scientifically published. The remaining notation [4] did use the term *textual language*, which we consider to broad for our research subject. Nevertheless, we consider the keywords of the SLR phase and the whole notation finding process to be successful and appropriate.

The Complement phase did not include an extensive search strategy because we focus on scientifically published notations in this survey. We complement previous intensive search strategies with the most common notations used in industry. To achieve this, we imitate the common search strategy that covers the very first popular results only. We included all notations of previous surveys [25,22,23] in the analysis. In total, we found 12 new notations compared to previous surveys: Alloy, AUML, Clafer, Dcharts, IOM/T, pgf-umlcd, pgf-umlsd, TCD, tUML, txtUML, UML/P, and uml-sequence-diagram-dsl-txl.

We addressed *inconsistent measurements* and *biased experimenter* with a rigorous review protocol and instructions for the characteristics extraction. The characteristics for the notations can be determined in an objective way. Mazanec et al. [25], however, used subjective characteristics such as *readability* or *simplicity* and did not mention how they have been determined.

We addressed *incomplete information* by using multiple information sources. We characterized all 33 notations by extracting information from the papers, and mining websites and source code (if possible). The former is the standard approach during a SLR but the two latter allow filling the gaps left by the scientific papers. Especially, the project's activity and editor features are most commonly not covered by publications. Only Alf, Dcharts, HUTN, IOM/T, and TCD did not provide sufficient information to determine these characteristics.

The external validity requires generalizable results. The survey results are applicable for scenarios that cover collaborative UML editing with textual notations in general because the characteristics do not focus on a specific scenario. This is a benefit over the previous surveys [24,22,23] that focused on teaching UML to visually impaired people or focused on specific UML diagram types in industry. The fuzzy characteristics in [25] lead to a limited generalization and applicability.

6 Conclusions

The Unified Modeling Language (UML) is the most commonly used modeling language. Its specification defines a graphical but no complete textual notation. Many specialized textual notations evolved but they are incompatible and highly fragmented with respect to UML coverage, editing experience, and applicability in engineering teams. There is no notation that clearly dominates the other notations in every aspect. Therefore, practitioners have to select a notation per usage scenario and do many trade-off decisions. This survey facilitates the selection of notations by providing a comprehensive list of 33 UML notations and their 20 characteristics related to applicability. The characteristics do not focus on a specific application domain but provide objective selection criteria.

The review method used in the survey produces reproducible and reliable results. We applied a classic systematic literature review in order to identify scientifically published approaches. In the second phase, we used snowballing to build a reference closure in order to find publications not covered by the keyword-based search from the first phase and to validate the keywords. In a third phase, we used Google searches to find not-scientifically published notations and complement our existing results. This approach is beneficial because we identified about half of the notations in the latter two phases.

The major insights we gained by analyzing our results are: a) Users have to know the UML diagram types they require in their scenarios because most notations only support a single diagram type and there is no single implemented notation that supports all types. b) Using the surveyed notations for complex UML models degrades the maintainability because almost all implementing tools do not provide editing support for complex tasks such as refactoring models or referencing existing elements. c) Teams can integrate the textual notation in existing tool chains mostly by using imports and exports of UML models but only few notations provide this feature.We could, however, not find a single notation that is applicable without restrictions and clearly dominates all other notations. Instead, many notations simply focus on graphics generation for documentation purposes and do not allow modeling and processing of the modeled information. A scenario-specific selection process is still necessary.

Practitioners, tools vendors, and researchers can benefit from this survey: Practitioners can focus on evaluating important characteristics of notations instead of struggling with finding notations and extracting the information with respect to UML coverage, editing experience, and applicability in engineering teams. Even if the survey does not cover all relevant aspects, it provides a considerable foundation for preselecting notations. This lowers the evaluation effort, allows to evaluate more notations within given time constraints, and therefore enables better selections.

Tool vendors for notations can identify seldom supported features and either advertise their support for these features or can try to integrate them in order to increase their market share.

Researchers can identify seldom supported features and can investigate the reason for the bad coverage. For instance, if tool vendors worry about the complexity of their notation when including further diagram types, researchers can develop approaches for integrating views in textual modeling frameworks.

We identified two tasks as future work: First, we see a need for notations that target proper UML modeling. This means a considerable UML diagram type coverage as well as support for import and export of standard UML models. Engineering teams cannot integrate other tools into their existing environments. Second, we need a systematic comparison and rating approach for the supported UML elements. This requires a definition of the UML elements usually contained in a UML diagram type and a set of sample models for elements. If the notation cannot represent the model, it does not support the corresponding element. We plan to develop guidelines and example models for assessing the UML coverage of UML notations.

Acknowledgements

This work is funded by the German Federal Ministry of Labour and Social Affairs under grant 01KM141108.

References

1. Auer, M., Tschurtschenthaler, T., Biffl, S.: A flyweight uml modelling tool for software development in heterogeneous environments. In: EUROMICRO'03. pp. 267–272. IEEE (2003)
2. Cabot, J.: Modeling languages – uml tools. `https://modeling-languages.com/uml-tools` (2015), accessed 04. August 2015
3. Chaves, R.: Textuml toolkit. `http://abstratt.github.io/textuml/readme.html` (2015), accessed 14. August 2015
4. Del Nero Grillo, F., de Mattos Fortes, R.: Tests with blind programmers using awmo: An accessible web modeling tool. In: UAHCI'14. pp. 104–113 (2014)
5. Dévai, G., Kovács, G.F., An, Á.: Textual, executable, translatable UML. In: OCL'14. pp. 3–12 (2014), `http://ceur-ws.org/Vol-1285/paper01.pdf`
6. Dobing, B., Parsons, J.: How UML is used. Commun. ACM 49(5), 109–113 (2006)
7. Doi, T., Yoshioka, N., Tahara, Y., Honiden, S.: Bridging the gap between AUML and implementation using IOM/T. In: ProMAS'04. pp. 147–162 (2004)
8. Erickson, J., Siau, K.: Can uml be simplified? practitioner use of uml in separate domains. In: EMMSAD'07. p. 8998 (2007)
9. EventHelix: Eventstudio system designer 6. `https://www.eventhelix.com/EventStudio` (2016)
10. Feng, H.: DCharts, a formalism for modeling and simulation based design of reactive software systems. Master's thesis, School of Computer Science, McGill University, Montreal, Canada (2004)
11. Gheorghies, O.: Metauml - github. `https://github.com/ogheorghies/MetaUML` (2015), accessed 14. August 2015
12. Grönniger, H., Krahn, H., Rumpe, B., Schindler, M., Völkel, S.: Text-based modeling. CoRR abs/1409.6623 (2014)
13. Harris, T.: Create uml diagrams online in seconds, no special tools needed. `http://yuml.me` (2015), accessed 14. August 2015
14. He, Y.: Comparison of the modeling languages alloy and UML. In: SERP'06. pp. 671–677 (2006)
15. Information technology – z formal specification notation – syntax, type system and semantics. Standard, International Organization for Standardization (2002)
16. Jackson, D.: Alloy: a lightweight object modelling notation. ACM TOSEM 11(2), 256–290 (2002)
17. Jouault, F., Delatour, J.: Towards fixing sketchy UML models by leveraging textual notations: Application to real-time embedded systems. In: OCL'14. pp. 73–82 (2014)
18. Kern, H.: Study of interoperability between meta-modeling tools. In: FedCSIS'14. pp. 1629–1637 (Sept 2014)
19. Khaled, L.: A comparison between uml tools. In: ICECS'09. pp. 111–114 (Dec 2009)
20. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (version 2.3). EBSE technical report, EBSE-2007-01, Keele University (2007)

21. Lethbridge, T.: Umple: An open-source tool for easy-to-use modeling, analysis, and code generation. In: MoDELS'14 (2014)
22. Luque, L., Brandão, L.O., Tori, R., Brandão, A.A.F.: Are you seeing this? what is available and how can we include blind students in virtual uml learning activities. In: SBIE'14 (2014)
23. Luque, L., Veriscimo, E., Pereira, G., Filgueiras, L.: Can we work together? on the inclusion of blind people in uml model-based tasks. In: Inclusive Designing, pp. 223–233. Springer (2014)
24. Luque, L., Brandão, L., Tori, R., Brandão, A.: On the inclusion of blind people in uml e-learning activities. RBIE'15 23(02), 18 (2015)
25. Mazanec, M., Macek, O.: On general-purpose textual modeling languages. In: Dateso'12. pp. 1–12 (2012)
26. OMG: Action language for foundational uml (alf). `http://www.omg.org/spec/ALF/1.0.1/PDF` (2013)
27. OMG: Unified Modeling Language (UML) – Version 2.5. `http://www.omg.org/spec/UML/2.5/PDF` (March 2015)
28. OMG: XML Metadata Interchange (XMI) – Version 2.5.1. `http://www.omg.org/spec/XMI/2.5.1/PDF` (June 2015)
29. Open Source Initiative: Licenses by name. `http://opensource.org/licenses/alphabetical` (2015), accessed 04. August 2015
30. Reggio, G., Leotta, M., Ricca, F., Clerissi, D.: What are the used UML diagram constructs? A document and tool analysis study covering activity and use case diagrams. In: MODELSWARD'14. pp. 66–83 (2014)
31. Roques, A.: Plantuml : Open-source tool that uses simple textual descriptions to draw uml diagrams. `http://plantuml.com/` (2015), accessed 14. August 2015
32. Safdar, S.A., Iqbal, M.Z., Khan, M.U.: Empirical evaluation of uml modeling toolsa controlled experiment. In: Modelling Foundations and Applications, LNCS, vol. 9153, pp. 33–44. Springer (2015)
33. Seifermann, S., Groenda, H.: Survey on textual notations for the unified modeling language. In: MODELSWARD'16. pp. 28–39. SciTePress (2016)
34. Spinellis, D.: On the declarative specification of models. IEEE Software 20(2), 94–96 (2003)
35. Vieritz, H., Schilberg, D., Jeschke, S.: Access to uml diagrams with the hutn. In: Automation, Communication and Cybernetics in Science and Engineering 2013/2014, pp. 751–755. Springer (2014)
36. Walton, D.: ckwnc - uml sequence diagram editor. `http://www.ckwnc.com` (2013)
37. Washizaki, H., Akimoto, M., Hasebe, A., Kubo, A., Fukazawa, Y.: Tcd: A text-based uml class diagram notation and its model converters. In: Advances in Software Engineering, CCIS, vol. 117, pp. 296–302. Springer (2010)
38. Wikipedia: List of unified modeling language tools. `https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools` (2015), accessed 04. August 2015
39. Winikoff, M.: Towards making agent UML practical: A textual notation and a tool. In: NASA / DoD Conference on Evolvable Hardware. pp. 401–412 (2005)
40. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: EASE'14. pp. 38:1–38:10. ACM (2014)
41. Zayan, D.O.: Model evolution: Comparative study between clafer and textual uml. `http://gsd.uwaterloo.ca/sites/default/files/Model%20Evolution;%20Clafer%20versus%20Textual%20UML.pdf` (April 2012), project Report