Software and Systems Modeling
https://doi.org/10.1007/s10270-024-01176-y

THEME SECTION PAPER

Generating domain models from natural language text using NLP: a benchmark dataset and experimental comparison of tools

Fatma Bozyigit[1,2] · Tolgahan Bardakci[1] · Alireza Khalilipour[1,2] · Moharram Challenger[1,2] · Guus Ramackers[3] · Önder Babur[4,5] · Michel R. V. Chaudron[5]

Received: 14 August 2023 / Revised: 10 January 2024 / Accepted: 14 March 2024

Abstract

Software requirements specification describes users' needs and expectations on some target system. Requirements documents are typically represented by unstructured natural language text. Such texts are the basis for the various subsequent activities in software development, such as software analysis and design. As part of software analysis, domain models are made that describe the key concepts and relations between them. Since the analysis process is performed manually by business analysts, it is time-consuming and may introduce mistakes. Recently, researchers have worked toward automating the synthesis of domain models from textual software requirements. Current studies on this topic have limitations in terms of the volume and heterogeneity of experimental datasets. To remedy this, we provide a curated dataset of software requirements to be utilized as a benchmark by algorithms that transform textual requirements documents into domain models. We present a detailed evaluation of two text-to-model approaches: one based on a large-language model (ChatGPT) and one building on grammatical rules (txt2Model). Our evaluation reveals that both tools yield promising results with relatively high F-scores for modeling the classes, attributes, methods, and relationships, with txt2Model performing better than ChatGPT on average. Both tools have relatively lower performance and high variance when it comes to the relation types. We believe our dataset and experimental evaluation pave to way to advance the field of automated model generation from requirements.

Keywords Software functional requirements · Software models · Text-to-model transformation · Benchmark dataset

1 Introduction

Software functional requirements or functional case descriptions can sometimes be ambiguous, and this ambiguity leads to many different perceptions. By creating software models, such as diagrams, flowcharts, or UML diagrams, it is possible to provide a clearer and more concrete representation of the functional requirements. This helps stakeholders,

Communicated by Lano, Kolahdouz-Rahimi, Yassipour-Tehrani, Burgueño, and Uma.

Tolgahan Bardakci, Alireza Khalilipour, Moharram Challenger, Guus Ramackers, Önder Babur, and Michel R. V. Chaudron have contributed equally to this work.

✉ Fatma Bozyigit
fatma.bozyigit@uantwerpen.be

Tolgahan Bardakci
tolgahan.bardakci@uantwerpen.be

Alireza Khalilipour
alireza.khalilipour@uantwerpen.be

Moharram Challenger
moharram.challenger@uantwerpen.be

Guus Ramackers
g.j.ramackers@liacs.leidenuniv.nl

Önder Babur
onder.babur@wur.nl

Michel R. V. Chaudron
m.r.v.chaudron@tue.nl

[1] Department of Computer Science, University of Antwerp, Antwerp, Belgium

[2] AnSyMo/Cosys core lab, Flanders Make Strategic Research Center, Lommel, Belgium

[3] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands

[4] Information Technology Group, Wageningen University and Research, Wageningen, The Netherlands

[5] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Published online: 08 May 2024

including developers, testers, and clients, better understand the intended functionality and behavior of the software. Although the analysis phase is generally considered a manual task, the automatic generation of software models from text-based functional requirements documents has become a promising area of interest for researchers.

Recent studies have demonstrated that artificial intelligence (AI) can automatically identify and categorize key domain concepts from textual functional requirements using techniques like machine learning and text mining [1–4]. Since Natural Language Processing (NLP) is the fundamental basis of today's research on textual documents, NLP is contributing to an enormous amount of work related to this area of study.

Figure 1 shows a high-level overview of an NLP procedure to create a domain model from textual requirements documents [5]. This general procedure has an initial analysis phase in which various analytical approaches such as Lexical, Syntactic, Semantic, Discourse, and Pragmatic are applied to the main text, yielding intermediate data. A final domain model is then created from the intermediate data by utilizing various heuristic-based transformations such as Standard Rule-based, Enhanced Rule-based, Ontology-based, and Pattern Rule-based.

One can come up with intelligent tools that can be used as an intelligent assistant alongside an expert to better analyze given the advancements in artificial intelligence and text processing, especially in the areas of software engineering where text is directly involved. More specifically, directions including functional requirements are beneficial. Similar to numerous other intelligent systems, this one also needs sufficient data in this case, requirement texts—for the intelligent engines to be trained and prepared for use. The absence of appropriate datasets in this field is the research's most difficult obstacle.

Currently, the evaluation of text-to-model studies is limited and ad hoc due to the small number of requirements in the evaluation datasets used. The experimental evaluation of such approaches will be more convincing and realistic when performed on larger datasets. In particular, the existing datasets should be extended with gold annotations (i.e., solution models) and should be extended with more cases so that they can be more useful for further studies.

An additional challenge is that many datasets that are reported in published studies are not publicly shared. The public sharing of datasets provides researchers with the opportunity to evaluate and compare their proposed approaches in a more objective and comparable manner. Consequently, we anticipate that the outcomes of our research in this paper will aid in dataset-sharing activities.

The main contributions of this paper are:

1. Publication of a dataset that is created using a systematic approach for use as a benchmark for text-to-model approaches. Concretely, we created a well-defined experimental dataset containing 120 software functional requirements in English and have made it publicly available on IEEEDataPort[1] to be used by other researchers. Moreover, the provided dataset includes gold annotation and statistic files that describe various characteristics of the functional requirements such as the number of sentences, words, classes, and so on.
2. An empirical comparison of two tools, txt2Model and ChatGPT, based on the presented dataset and the proposed evaluation method encompassing different criteria.

The rest of this paper is structured as follows: Sect. 2 reviews the related work on the available datasets and approaches for text-to-model transformation. Section 3 illustrates the systematic methodology to create our benchmarking dataset. We propose our evaluation method and elaborate on the experimental results in Sect. 4. Finally, Sect. 5 concludes and suggests future work.

2 Related work

With the penetration of AI into many scientific and industrial fields, software engineering is also exploring the use of intelligent methods for supporting software development activities. Hence, the artificial intelligence for software engineering (AISE) research area has recently emerged. In the case of automated software development, we would like to employ AI to reduce the time and effort required in software development [6].

Qualified efforts have been made in AISE, such as for cost management and estimation [7], effort estimation [8], fault prediction [9], intelligent model management [10, 11], and so on. In each of these areas, experimental data makes it possible to develop descriptive or predictive models, enabling an intelligent description of the data or future predictions. Hence, the availability of appropriate datasets is crucial to these activities.

Requirement formalization and requirement engineering using NLP have been thoroughly studied in the literature, with two review articles presenting a comprehensive classification in this area [12, 13]. Falessi et al. [14] investigated the impact of applying language processing techniques to save the human effort required for the requirements document classification. Based on BERT and graph attention networks, Li et al. [15] provided a model for the classification of require-

[1] https://ieee-dataport.org/documents/dataset-text-requirements-models.

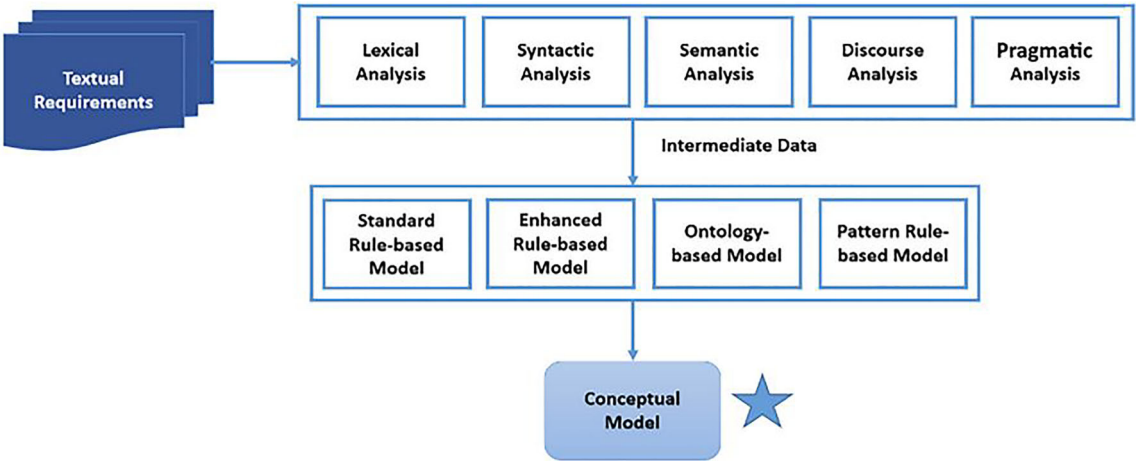

Fig. 1 General framework of approaches in the reviewed concept identification studies [5]

ments. Since sentence structure and syntactic information in requirements have also been taken into consideration, this research stands out notably taking into account the sentence structure and syntactic information in requirements as well.

One of the major challenges in the field of AISE is the initial phase of requirement analysis. In this phase, requirement documents have the least structure and may have noise in terms of grammar or terminology [16]. Accordingly, there is a need for appropriate datasets to evaluate the approaches in this area. There is a necessity to use text-to-model transformation tools to benchmark and validate the proposed datasets. Considerable effort has been made to extract knowledge from requirements documents [17]. Due to the textual nature of the data in this research field, representation learning and NLP techniques [18] are employed with some degree of success. However, approaches for automatically extracting domain models from requirements need further research. Hereupon, we present the related work separately on *Datasets* and *Text to Model* tools.

2.1 Datasets

A comprehensive dataset for benchmarking provides many benefits for researchers. First of all, it can encourage the study of a research problem. Moreover, many academics may have an opportunity to compare their approaches with the state of the art using common and well-designed input data—thus improving objectivity. Therefore, benchmark datasets are significant in building scientific studies and experiments.

With the advance of AI, applying such methods in the software engineering domain, like extracting system functional requirements from textual descriptions, domain model generation, requirement categorization, and finding similar requirement documents to enable reuse, has become more popular. In Table 1, we present an overview of approaches for extracting domain models from natural language text requirements. All datasets used in these studies are quite small and not close to realistic datasets.

This table was created based on a systematic search (using a keyword list—see Sect. 4) to find publicly available datasets of functional requirements. We will investigate further (Table 2) the study with the most extensive dataset, the PURE dataset [30], which includes 79 requirements documents. By reviewing the PURE dataset in a detailed manner, we find that its requirements make it difficult to automatically generate well-formed structural models.

Large heterogeneity within the documents is one of the complications. For example, some documents include images and include other types of explanations and descriptions besides the systems' functionality. Other forms of heterogeneity are that some cases follow the format of use-case templates while others are descriptions in natural language, and yet others mix present and past tense.

2.2 Text-to-model approaches

The early studies of model extraction from functional requirements documents were based on grammatical rules and heuristics—similar to other text-processing research utilizing rule-based techniques [27, 28]. More recently, large language models (LLMs) have been successfully employed for this task. Bragilovski et al. [29] developed an approach to generate domain models from user stories. User stories are one particular style of addressing the expression of requirements for systems. While "traditional" approaches more or less factually state the functionalities expected from a system, user stories describe the use of and interaction with a system from the perspective of users that use one particular feature of the system. Several templates exist for describing user stories. Such templates provide a slightly more standardized input which may be a modest advantage for the automated extraction of domain models. This study focuses on improving the

Table 1 Studies for generating UML Diagrams from requirements

Paper	Supported Languages	Method	UML Diagram	Source code	No of Reqr's	Avg. No of words	Evaluation
[5]	English & Turkish	Rule-based	Object	C#	20	94	Pr, Re, Fm
[19]	English	Rule-based	Object	–	1	94	–
[2]	English	Rule-based	Class	–	3	87	Pr, Re
[20]	English	Ontology-based	Class	–	1	97	–
[21]	English	Pattern-based	Class	–	1	78	–
[22]	English	Rule-based	Class	Java	5	102	Pr, Re
[23]	English	Rule-based	Class	Java, VB	1	85	Pr, Re
[24]	English	Rule-based	Object	Java	6	–	–
[25]	English	Rule-based	Class	Java	–	–	–
[26]	English	Rule-based	Class	Java	–	–	–
[27]	English	Rule-based	Conceptual models	–	2	–	Pr, Re
[28]	English	Rule-based	Conceptual models	–	49	16	Pr, Re, Fm
[29]	English	Rule-based	Domain models	–	7	–	Pr, Re, Fm

The abbreviations are as follows—Pr: precision, Re: recall, Fm: F-measure

detection of relations between concepts, as this seems to be the aspect where most improvement is needed. They applied their approach to seven case studies. These case studies come from assignments in software engineering courses at Utrecht University. They conducted a detailed quantitative evaluation but also proposed comparison with and feedback from human users as future work.

Ramackers et al. [31] presented a vision for automatically generating a UML specification from natural language requirements—either in text or in spoken form. The noteworthy point in this article is that by using components, the user can interact with the transformation system to support a "human-in-the-loop" approach. In this way, the user can make changes in the created (visual) model or inform the main component of the transformer through feedback. In addition, it supports both behavioral and structural aspects of modeling.

Sedrakyan et al. [32] presented a method (called TeToMo) that integrates machine learning-based methods, word embeddings, heuristic rules, statistical and linguistic knowledge to solve functional requirements analysis, and modeling problems. They evaluated their output with models designed by an expert.

Hamza et al. [33] generated use case diagrams from four case studies of English requirements, considering NLP techniques. In the proposed method, using a spell checker, only spelling mistakes are recognized and warned to the user for modification.

Arora et al. [34] developed an automated generator of the domain model that uses extraction rules and combines those rules with complementary rules from the information retrieval literature and rules obtained from modern NLP dependency parsers. They applied their model generator to four industrial requirements documents. As is typical for industrial documents, these are not shared publicly. They evaluated their approach by interviewing one expert on the results produced, yielding qualitative insights into the strengths and weaknesses of their approach.

The incorporation of AI into the field of software modeling is covered by Camara et al. [35]. This study has identified several flaws, including grammatical issues, incorrect semantic interpretation, inconsistent responses, and unscalability in well-known systems like ChatGPT. In the work of Lano et al. [36], an approach for automating the extraction of software specifications from the requirements is described to make model-based software engineering agile. The study has employed text processing and machine learning techniques to automatically identify the required data and system components, hence facilitating their utilization in subsequent development phases, particularly model-driven engineering.

In addition to the research done in the field of transforming requirements into models, some researchers have also exposed the results of their research as available tools. We were able to access two tools, and since one of our goals in this article is to benchmark datasets based on real tools, we will discuss these tools in the next section.

2.2.1 txt2Model tool

To evaluate the quality of our benchmark dataset, we realize an experimental study by applying two tools to the proposed benchmark dataset. One of these tools, txt2Model, was implemented by Bozyigit et al. [37] who is an author in this study. txt2Model transforms functional software requirements into UML class diagrams by employing NLP techniques and a modified rule set containing twenty-six rules

to find object-oriented design elements from requirements. The current state of the user interface can be seen in the screenshot provided in Fig. 2.

2.2.2 ChatGPT tool

The LLM-based tool that we use in this paper is from the OpenAI organization and is called "ChatGPT" [38]. ChatGPT is a language model developed by OpenAI, built upon the GPT (Generative Pre-trained Transformer) architecture. It is designed to generate human-like text based on any textual input it receives. ChatGPT has been trained on a wide variety of text sources from the internet, books, articles, and more, up until its knowledge cutoff in September 2021.

The model's primary strength lies in its ability to generate coherent and contextually relevant responses to a wide range of prompts, making it suitable for tasks like answering questions, providing explanations, offering creative writing suggestions, and engaging in natural language conversations. It does not possess genuine understanding or consciousness but employs patterns it has learned from its training data to produce text that often appears human-like. Some of the things it can do include, but not limited to: answering questions, translating text, generating text, summarizing text, and providing definitions.

We utilize these capabilities of ChatGPT, specifically version 3.5, to compare it with txt2Model [37]. For this, we instruct ChatGPT to produce UML diagrams. In principle, ChatGPT can produce class-like diagrams using characters to create boxes and lines. However, we instruct ChatGPT to generate textual lists of classes, associated attributes, and relationships from the given requirements documents. This allows us to quickly and accurately create a domain model that represents the requirements. Moreover, this allows us to process it efficiently, without the need for parsing UML exchange formats such as XMI.

3 Creating the benchmark dataset

Assessing the performance of automatic text-to-model generation methods relies heavily on well-designed datasets. However, constructing a large-scale and meaningful requirements dataset for software model generation is challenging. The challenge lies mainly in the curation and application of careful quality control of the data.

We follow the methodology described in Sect. 3.1 to design an organized dataset for software requirements interpretation. We use a general definition of benchmark to ensure wide applicability. It applies equally to benchmarking of tools and techniques, as well as other technologies so we will use these terms interchangeably. We define a benchmark as a set of tests used to compare the performance of alternative tools or techniques.

3.1 Methodology

For creating our dataset, we follow the guidelines and standards in the software engineering literature for creating benchmarks [39, 40]. According to Sim et al. [39], a benchmark should have three elements: a motivating comparison to advance the research area it is intended for, a representative test sample for the task that the tools or techniques are expected to solve in practice, and performance measures.

Furthermore, Kistowski et al. [40] listed the following five key characteristics of benchmarks: (KC1) relevance, as discussed above by Sim et al. [39], (KC2) reproducibility, consistent use and generation of results, (KC3) fairness, allowing different tests and tools to compete, (KC4) verifiability, providing confidence on the accuracy of the results, and (KC5) usability, facilitating and not hindering the use of the benchmark.

We create our benchmark dataset to satisfy these characteristics as much as possible. We explain the process of constructing the dataset in Sect. 3.2, as well as provide a description and evaluation of the dataset in Sects. 3.3 and 3.4.

3.2 Process of constructing the dataset

During the process of constructing the dataset, we included three aspects defined by Sim et al. [39], namely motivating comparison, task sample, and performance measures.

Comparison: The first component, motivation comparison, encompasses two concepts: comparison and motivation. It captures both the technical comparison to be made as well as the research agenda that will be advanced by making this comparison. Our motivation in this study is to build a comprehensive and well-designed dataset to be used by researchers in the software engineering field. To achieve this, we analyzed all the similar studies in the literature with the search strategy we defined (see Sect. 4.1), highlighting several points in the current approaches to be improved. We observed that most datasets in the current studies (except a few, e.g., PURE [30]) are either inaccessible or are very limited in the number and diversity of the requirements documents they contain. Although there is a substantial amount of research on concept identification, there is a lack of a large-scale benchmark dataset to be used in generating models from requirements.

Task Sample: The tests in the benchmark should be a representative sample of the tasks that the tool or technique is expected to solve in actual practice. Since it is not possible to include the entire population of the problem domain, a selection of tasks acts as surrogates. To provide the criteria

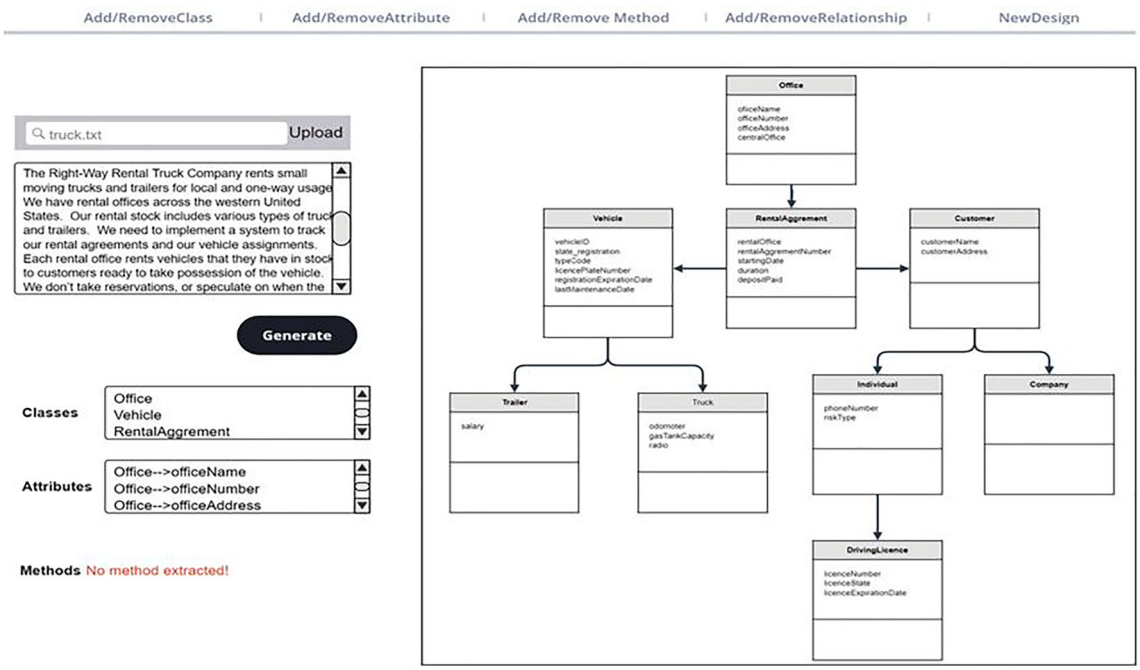

Fig. 2 txt2Model [37]—A screenshot of its user interface

from this point of view, we designed a comprehensive data search mechanism. This process will be detailed in Sect. 3.2.1

Performance Measures: These measurements can be made by a computer or by a human, and can be quantitative or qualitative. These are typically used to demonstrate the features and capabilities of a new tool or technique and are occasionally used to compare different technologies in an exploratory manner. The criteria used as performance measures are explained later in the experimental part of our paper in Sect. 4.

3.2.1 Data collection

In the process of constructing the benchmark dataset, we use different sources such as books, journals, publicly available datasets, and online course materials.

We first conduct a review of relevant research papers, articles, conference proceedings, and other scholarly sources to find publicly available requirements to be included in our benchmarking dataset. In this regard, a search strategy incorporating procedures for paper selection, including the identification of search terms and the creation of search queries, is devised. During the phase of study selection, prominent digital databases such as Science Direct, IEEE Explore, ACM, and Springer, as well as the Google Scholar academic search engine, are utilized to identify papers of high quality. The search encompasses journal papers and conference proceedings spanning from 2000 to 2023. We employ specific search strings in our data collection process: "software requirement," "software requirement dataset," "functional requirement," "software requirement document," and "software analysis document". Second, the search strings are constructed based on specified search terms, including population, intervention, and outcome. These search strings are adjusted to accommodate advanced source-searching methods within each selected digital library. For instance, while crafting a search string like (TITLE-ABS-KEY((“software requirements" or “software analysis document" or “software requirement document" or “software requirement," “software requirement dataset," “functional requirement," “software requirement document," and “software analysis document") AND LIMIT-TO (PUBYEAR, 2000))) for ScienceDirect, the equivalent query is entered as (“software requirement document," “software requirement," “software requirement dataset," “functional requirement," “software requirement document”) AND (year≥2000 AND year≤2023).

After completing the search on academic databases, we use search strings on Google to find software requirements in open educational resources (e.g., course materials), and Web-based knowledge-sharing platforms (i.e., GitHub, Medium, Kaggle). Also, we add 11 functional requirements of a software company which one of the authors in the paper works for (Fig. 3).

To decide whether these requirements are suitable to be included in the proposed dataset, all the requirements collected from different sources (203 cases) were analyzed by the two researchers of the paper (Fatma Bozyigit and Tol-

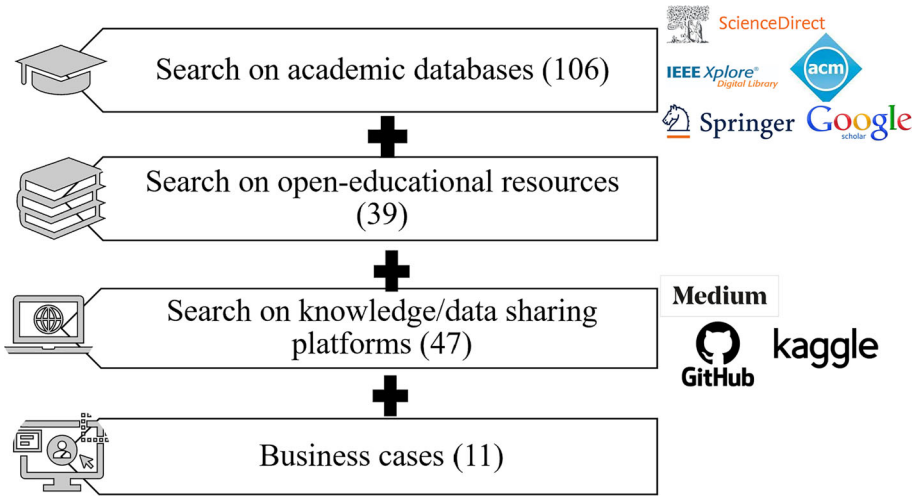

Fig. 3 Sources we used for creating our benchmark dataset

gahan Bardakci). In this direction, the researchers eliminate redundant (duplicate data stored in multiple places within the same system), trivial (including the simple and limited number of sentences), fully nonfunctional software requirements, and test cases. Consideringly, 83 of the selected cases were eliminated. Finally, we identify 120 cases that remained within scope. The whole process of selecting the requirements documents during the data collection is illustrated in Fig. 4.

3.2.2 Preprocessing

In the pursuit of developing a benchmark dataset comprising software functional requirements, a systematic approach is employed to ensure data quality. This process entailed the collection of requirements documents from publicly available sources, encompassing a diverse array of origins. As these requirements exhibited inherent variability, including grammatical errors, vague descriptions, extraneous elements such as images, and excessive details, a suite of text preprocessing techniques was thoughtfully deployed. This preprocessing phase aims to rectify the identified challenges, enhancing the overall quality and consistency of the dataset.

To specifically address these challenges, we used the assistance of a third-party proofreading service. This service was instrumental in correcting grammatical mistakes and misspelled words, thereby ensuring the linguistic correctness of the dataset. Additionally, it played a crucial role in refining the clarity of the requirements by rephrasing vague descriptions and rearranging sentences for better coherence. After preprocessing with this service, we manually deleted unrelated parts of these requirement datasets, like images. This preprocessing phase was vital in maintaining a high standard of data quality and ensuring that the dataset reflects accurate and clear software functional requirements.

The steps in the preprocessing can be listed as follows:

1. Correcting grammatical mistakes,
2. Correcting misspelled words,
3. Re-arranging some sentences for clarity,
4. Deleting unnecessary items, such as images.

3.2.3 Categorization of the functional requirements documents

Categorization in terms of complexity. To achieve our goals of having a representative task sample and fairness for benchmarking different tools (KC2), we aimed to have diversity in terms of the complexity of the requirement. This is particularly important, for example, considering the scenario where a tool designed for automatic model transformation might perform poorly on complex requirements while working well on simple requirements documents. Realizing this situation makes the system designer aware of fixing some critical issues to make the tool more useful.

For measuring the complexity, we analyzed the requirements documents using the number of sentences (NoS), the number of words (NoW), the number of words after removing stop words (NoSW), and the number of unique words (NoUW). We calculated these values using our Python scripts and documented them in an Excel Sheet. We created a formula for calculating requirement complexity values and giving the labels *simple, medium*, and *complex*. We used the following formula (Eq. 1) for the calculation:

$$Complexity = \frac{(0,4 * NoS) + (0,2 * NoW) + (0,2 * NoUW) + (0,2 * NoSW)}{10} \quad (1)$$

After these calculations, we categorized the requirements documents, a sample of which is depicted in Table 2. The final

Table 2 Requirement statistics

No	Requirement Name	Source	Domain	Complexity	number of sentences	number of words	number of unique words	number of words after eliminating stop words
R1	Restaurant Management	Course material (U Izmir Bakircay)	Business	Medium	20	255	125	176
R2	Employee Management	Course material (U Izmir Bakircay)	Business	Medium	22	277	122	179
R3	Library	Course material (U Izmir Bakircay)	Education	Medium	21	233	115	147
R4	Computer Game I (Galaxy Sleuth)	Course material (U Nevada)	Technology	Complex	39	714	275	432
R5	Computer Game II (Spy-Robot)	Course material (U Sichuan)	Technology	Medium	19	410	198	230
R6	Academic Program	Course material (U Izmir Bakircay)	Education	Simple	9	129	77	80
R7	Supermarket	Course material (U Izmir Bakircay)	Business	Complex	21	585	230	328
R8	Hotel Reservation	Course material (U Izmir Bakircay)	Business	Medium	20	301	119	184
R9	BeWell App	Course material	Health	Medium	30	433	145	253
R10	File Manager	Course material (U Celal Bayar)	Technology	Simple	7	83	46	55
R11	Football Team	Course material (U Celal Bayar)	Entertainment	Simple	12	136	67	84
R12	Rented-car Gallery Management System	Course material (U Celal Bayar)	Business	Simple	7	120	70	83
R13	Course Enrollment	Academic	Education	Medium	15	234	111	144
R14	ATM Rumbaugh	Academic	Finance	Simple	10	164	89	103
R15	Video Rental	Academic	Entertainment	Simple	17	221	98	137
…	…	…	…	…	…	…	…	…
R30	Wallet Application	Software Company	Finance	Medium	19	251	122	161
R31	Restaurant	CourseMaterial (U Izmir Bakircay)	Business	Complex	62	808	270	544
R32	Energy Management System	Business-Energy	Energy	Complex	30	519	156	357
R33	Travel System	Academic	Entertainment	Complex	50	551	210	410
R34	Truck	Academic	Business	Complex	32	494	205	314
R35	Unified University Inventory System	Academic	Education	Complex	16	524	189	352
R36	Video Rental	Academic	Business	Complex	48	553	160	371
R37	LawFirm2	Academic [31]	Business	Complex	81	1457	438	891
R38	Law Firm	Academic [31]	Business	Complex	38	858	344	570
R39	Insurance	Academic [31]	Finance	Complex	49	781	234	451
R40	Dental Clinic	Academic [31]	Health	Medium	22	330	187	204
…	…	…	…	…	…	…	…	…
R114	Clarus Low	Pure Dataset [30]	Business	Medium	18	268	98	205
R115	Clarus High	Pure Dataset [30]	Business	Complex	65	777	252	625
R116	Watcom	Pure Dataset [30]	Business	Complex	66	446	207	319

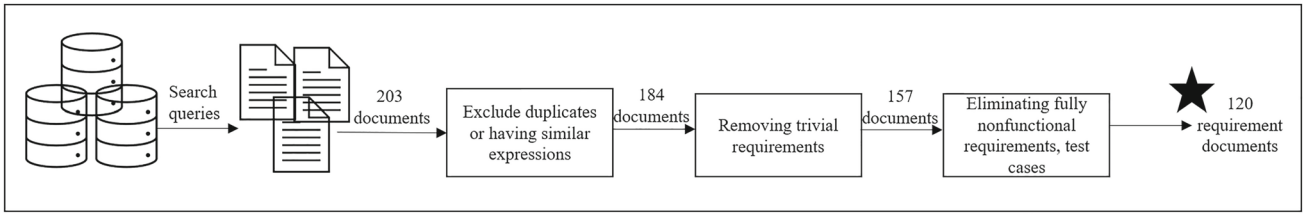

Fig. 4 Selection process during the data collection

Table 2 continued

No	Requirement Name	Source	Domain	Complexity	number of sentences	number of words	number of unique words	number of words after eliminating stop words
R117	Watcom GUI	Pure Dataset [30]	Business	Medium	13	323	176	214
R118	Sprat	Pure Dataset [30]	Business	Medium	17	397	150	268
R119	Rics	Pure Dataset [30]	Business	Complex	32	503	210	348
R120	Food Order App	Software Company	Business	Simple	11	142	84	89

dataset comprises 13 simple, 66 medium, and 41 complex requirements documents.

Categorization in terms of domain and source. A benchmark dataset should provide diversity and include various requirements documents concerning different domains (see KC1 relevance, and KC3 fairness in Sect. 3.1). In this study, we manually categorized the requirements documents with respect to their domains. Our final dataset covers the domains of business (systems that implement some type of shop), education, entertainment, finance, travel, and health (Table 2).

The other diversity point to consider is creating a dataset using a wide range of sources. Our aim is primarily to create a benchmark that is used by a technical research community. The community of interest may include participants from academia, industry, but they are all primarily interested in scientific research. Thus, we collect different course materials from various universities worldwide, the software requirements used in academic papers, and the requirements from a software company (Table 2).

3.3 Data description and availability

To partially validate the dataset presented in terms of domain diversity, the selected requirements have a suitable distribution and, according to Fig. 5, they are from various domains such as Business, Education, Technology, Health, Entertainment, Finance, Construction, and Transportation. The largest category (42 cases) is associated with Business, while the smallest category (4 cases) is tied to Transportation.

Moreover, the requirements documents in the benchmark dataset are classified based on their sources like academia and

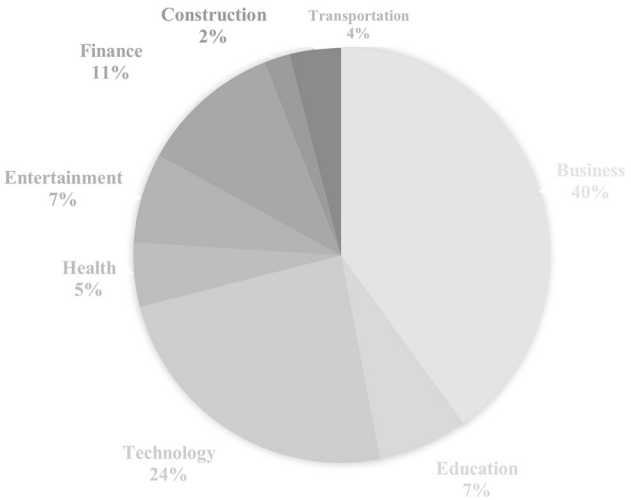

Fig. 5 Domains of the requirements documents

industry. Figure 6 shows that the data distribution between academic (58 cases) and industry (62 cases) categories is nearly equal. Hereby, it will be possible to achieve more realistic results during the validation of the tools that perform the model transformation from requirements to model.

When looking at the dataset from the criteria of complexity, the majority of the data has medium complexity (66 cases), then complex (44 cases), and 13 cases have simple complexity (Fig. 7). More specific statistical information is presented in Tables 3, 4, and 5 for simple, medium, and high complexity requirements, respectively. Such quantitative data was collected in five different categories: Complexity number (CompNo), number of sentences (NoS), number of words (NoW), number of unique words

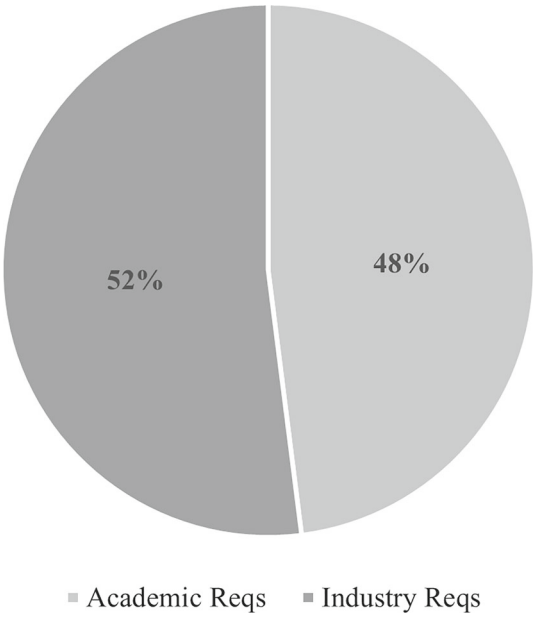

Fig. 6 Comparing academic and industrial requirements

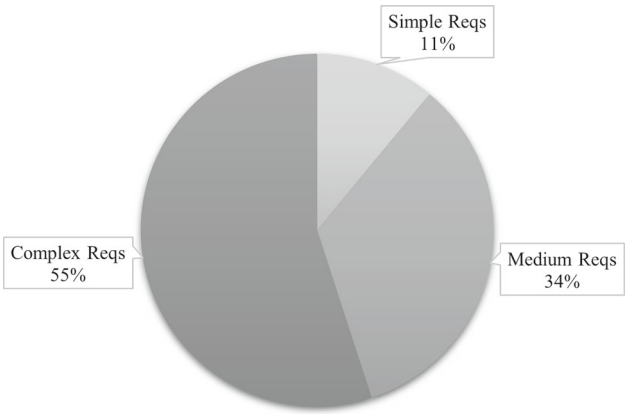

Fig. 7 Complexity of the requirements documents

(NoUW), and number of words after stop words were eliminated (NoWESW). According to this statistical data, the requirements collected had a mean of 26 sentences, 439 words, and 167 unique terms. After eliminating the stop words, the total number of words ranged between 55 and 891. In terms of the number of items mentioned, the resulting deviations confirm the quality of the collected data.

The latest version of the proposed dataset and the benchmark evaluation scripts are available in IEEEDataPort [41]. The available manual includes instructions on extracting and assessing data using the recommended benchmarks.

3.4 Benchmark dataset evaluation

We have sought to meet the key characteristics of benchmarks proposed by Kistowski et al. [40] and Sim et al. [39]. In this section, we elaborate on how our benchmark dataset meets those characteristics, in addition to the remarks on the dataset description and diversity in Sect. 3.3.

1. Relevance (KC1): Relevance involves ensuring that the benchmark addresses current and significant challenges in the field. In our benchmark, we focus on UML modeling. For instance, our dataset includes a variety of requirement data that reflects contemporary challenges in software engineering.
2. Reproducibility (KC2): This implies consistent use and generation of results. Our benchmark dataset has been developed with reproducibility in mind and is readily available on GitHub for anyone to access and use, as well as in a persistent database.
3. Fairness (KC3): Fairness entails allowing different tests and tools to compete on an equal footing. We ensure fairness in our benchmark by gathering documents from various resources, such as academia and industry, as well as establishing a high diversity in terms of complexity.
4. Verifiability (KC4): Providing confidence in the accuracy of the results is meant with verifiability. Our dataset supports verifiability through the experiments we present in this paper, which are publicly available in our replication package.
5. Usability (KC5): Facilitating the use of the benchmark without hindering it is an important characteristic. To enhance usability, our dataset is publicly available, documented, and ready to use. We create our benchmark dataset to satisfy these characteristics to the fullest extent possible.

4 Evaluation of text-to-model tools

In this section, we realize the evaluation of the benchmark dataset using two different tools, namely txt2Model [37] and ChatGPT. We feed the dataset into these tools and observe their model generation performance considering the five criteria we specified.

4.1 Selection of text-to-model tools

In this part of the study, we conduct a review of relevant research papers, articles, conference proceedings, and other scholarly sources to find publicly available tools to test their functionality on the benchmarking dataset we create.

In the initial phase of the search process, the first step is to identify keywords relevant to the research topic. Subsequently, these keywords are defined and organized based on research questions using the PICOC framework (Population, Intervention, Comparison, Outcome, Context), as introduced by [42]. Population criteria establish the scope of converting

Table 3 Descriptive statistics on the complexity metrics—for the requirements with simple complexity

	CompNo	NoS	NoW	NoUW	NoWESW
Mean	6.25	9.39	134.80	68.15	88.87
Median	6.74	9	136	68	88
Variance	2.11	14.53	1187.23	229.51	581.30
Skewness	0.36	0.69	0.63	0.11	0.28
Minimum	3.96	5	83	46	55
Maximum	9.80	17	221	98	137
Sum (for all req.)	83.48	131	1807	908	1197

Table 4 Descriptive statistics on the complexity metrics—for the requirements with medium complexity

	CompNo	NoS	NoW	NoUW	NoWESW
Mean	15.19	21.20	346.16	141.54	225.34
Median	15.2	22	341.5	141	223
Variance	7.17	36.54	4626.98	833.82	2135.45
Skewness	0.07	0.28	0.12	0.23	0.26
Minimum	10.38	9	233	96	144
Maximum	19.98	37	483	204	322
Sum (for all req.)	1018.14	1456	23284	9533	15178

Table 5 Descriptive statistics on the complexity metrics—for the requirements with high complexity

	CompNo	NoS	NoW	NoUW	NoWESW
Mean	27.74	36.85	647.48	227.94	429.69
Median	26.12	35	617	233	409
Variance	66.37	261.63	41646.48	3792.28	17787.12
Skewness	1.66	0.87	1.68	0.76	1.41
Minimum	20.46	16	446	111	284
Maximum	58.96	81	1457	438	891
Sum (for all req.)	1176.72	1634	27593	9669	18306

requirements into a conceptual model, covering areas like "requirements transformation," "requirements analysis," and "generating conceptual model." Intervention keywords refer to the methodologies used in the process, such as "Natural Language Processing" (NLP) and "rule-based model." Comparison keywords relate to studies analyzing requirements without employing automatic concept identification methods. Outcome keywords concentrate on the conceptual models generated from software requirements, including terms like "UML diagrams," "ontology model," and "source code." Context keywords revolve around contextual aspects within concept identification studies. Comparison criteria are not employed in the formulation of search strings. Instead, context criteria are utilized as exclusion criteria listed as patents, presentations, workshop papers, technical notes, informal papers, and tools not based on scientific study are eliminated. Duplicate papers of the same study are neglected. Paper that does not mention conceptual model in the title and abstract content is not included.

The search strings are constructed based on specified search terms, including population, intervention, and outcome. These search strings are adjusted to accommodate advanced source-searching methods within each selected digital library (Science Direct, IEEE Explore, ACM, and Springer).

After performing a literature search of available tools, we list three different text-to-model tools developed by [31, 34], and [37]. Since we could not execute the tools of [31] and [34], we have only the txt2Model tool to utilize the evaluation process. To complete the evaluation of benchmarking, we choose ChatGPT which has shown good results in a large variety of NLP tasks. Although the working mechanisms of these two tools are different, the performances of both tools on the dataset are quite consistent (see Sect. 4.3). This shows that the benchmarking dataset we created is suitable for testing and benchmarking the functionality of model transformation tools.

4.2 Evaluation method

The evaluation of a tool transforming requirements into UML models is a comprehensive process because there are various criteria affecting the performance of the produced model. Since the purpose of transformation tools is to determine: classes, relationships, attributes, methods, and relationship types, we defined five relative criteria as follows.

- Criterion 1 (C1): Finding the classes completely.
- Criterion 2 (C2): Finding the relationships between the classes completely.
- Criterion 3 (C3): Finding the attributes of the classes completely.
- Criterion 4 (C4): Finding the methods of the classes completely.
- Criterion 5 (C5): Specification of the relationship types correctly.

To observe the performance of the tools, we specified performance metrics considering the number of correct, incorrect, and missing elements for each criterion as follows.

- $C_i C$: the number of elements that are *correct* with respect to C_i.
- $C_i I$: the number of elements that are *incorrect* with respect to C_i.
- $C_i M$: the number of elements that are *missing* with respect to C_i.

The effectiveness of the system is assessed by comparing the output of the system with class diagrams created by experts. The set of design elements in the expert model is referred to as E, and the set of elements identified by the system is referred to as S. The comparison between S and E is used to determine the number of correct, incorrect, and missing elements. More such details are as follows:

1. The cardinality of the intersection of S and E is the number of elements that are correctly identified by the system. This is denoted as N_{correct}.
2. The cardinality of the difference between S and E is the number of elements that are incorrectly identified by the system in the generated class diagram. This is denoted as $N_{\text{incorrect}}$.
3. The cardinality of the difference between E and S is the number of elements that are missing from the output of the system and could not be extracted. This is denoted as N_{missing}.

Precision (Pr) is a measure of the accuracy of the proposed system and reflects the proportion of the output that is correct. It is calculated by dividing the number of correctly identified elements (N_{correct}) by the total number of elements extracted by the system ($N_{\text{correct}} + N_{\text{incorrect}}$). The formula for precision is (Eq. 2):

$$\text{Pr} = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{incorrect}}} \tag{2}$$

Recall (R) is a measure of the completeness of the output and gives information on how much of the total data was extracted by the system. It is obtained by finding the ratio of the correctly identified data to the total data in the expert model. Its formula is (Eq. 3):

$$Re = \frac{N_{\text{correct}}}{N_{\text{correct}} + N_{\text{missing}}} \tag{3}$$

F-measure (F) is a combination of precision and recall, which takes into account both the accuracy and completeness of the output. It is calculated using the harmonic mean of precision and recall (Eq. 4):

$$F_{\text{measure}} = \frac{2 \times Pr \times Re}{Pr + Re} \tag{4}$$

4.3 Evaluation results

In this section, we evaluate the performance of the tools txt2Model and ChatGPT on 30 requirements documents selected from our benchmark dataset. We analyze N_{correct}, $N_{\text{incorrect}}$, and N_{missing} values of each criterion for the tools txt2Model and ChatGPT, respectively. The detailed experimental results for each requirement are presented in Figs. 8 and 9. The x-axis represents each requirement with an *id* ranging from 1 to 30, while the y-axis represents the percentage of correctly, incorrectly predicted, and missing design elements in class diagrams generated by the tools. The bar chart depicts the overall performance scores across different requirements documents from various domains having different complexity.

Precision, recall, and F-measure values of both tools regarding each evaluation criterion are presented in Figs. 10, 11, and 12. Figure 10's five graphs illustrate the outcomes of the precision evaluation for both tools, txt2Model and ChatGPT, using the five criteria ranging from C1 to C5. The graphs indicate that normal distribution is typically lacking and that skewness is typically to the right or left. The txt2Model tool's median score is greater in C1 and C2, which indicates the improved performance of this tool. On the other hand, ChatGPT is more precise in both C3 and C4. Although the median is lower than that of ChatGPT, txt2Model's performance in C4 has a normal distribution. While both tools' precision in C5 is about the same, the ChatGPT skew is significantly to

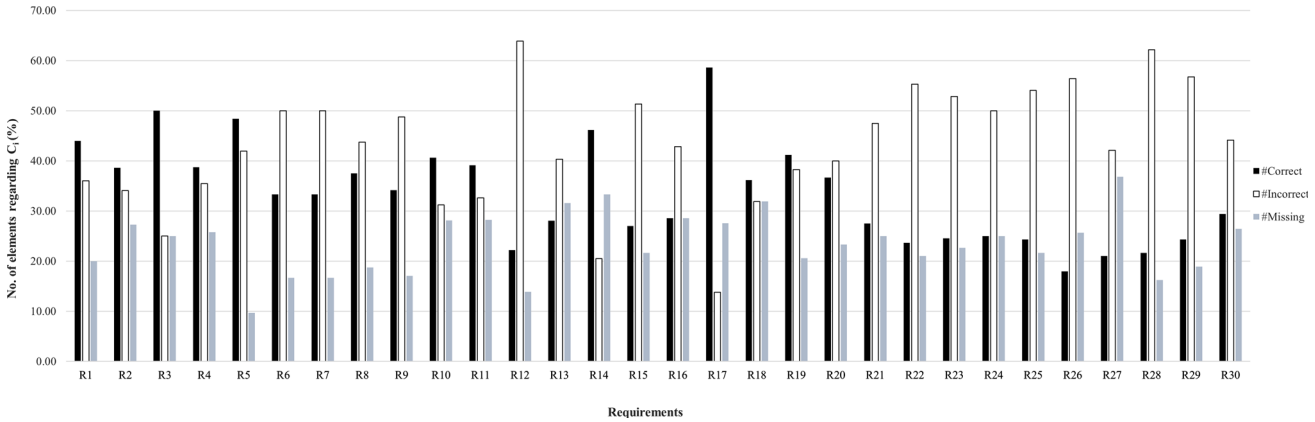

Fig. 8 Results for each criterion (C_iC: number of correct elements regarding C_i, C_iI: number of incorrect elements regarding C_iC, C_iM: number of missing elements regarding C_iC) - txt2Model

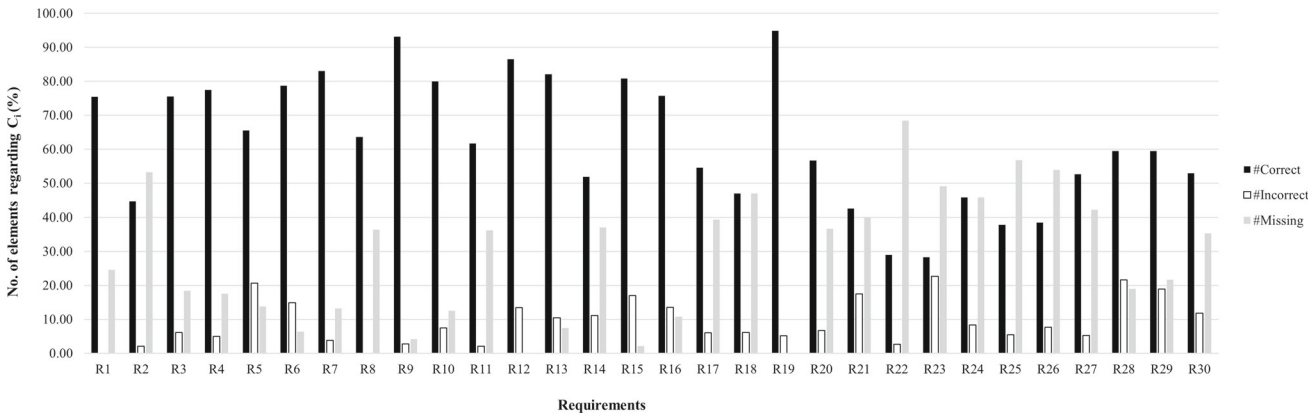

Fig. 9 Results for each criterion (C_iC: number of correct elements regarding C_i, C_iI: number of incorrect elements regarding C_iC, C_iM: number of missing elements regarding C_iC) - ChatGPT

the left. Thus, it can be concluded that the txt2Model tool's precision is better for C5 as well.

Figure 11 depicts the comparison of two tools across the five criteria (C1, C2, C3, C4, and C5) and the recall performance metric. Nearly always, txt2Model performs better than ChatGPT and has a higher median. ChatGPT has a lower minimum in each example, but a larger maximum in C4. Although ChatGPT's skew in C4 is to the right, txt2Model's minimum and first quartile are substantially higher. The performance of the txt2Model tool was superior in these circumstances as evidenced by the fact that in the C5 the first quartile of txt2Model, it is practically identical to the third quarter of ChatGPT. In other words, the txt2Model tool's skew is entirely to the right and the skew of ChatGPT is completely to the left.

In Fig. 12, the tools have been compared with the F-measure or F1 score according to the five conditions C1 to C5. In C2 and C4, txt2Model's tool has a normal distribution. In C2, the txt2Model tool has a higher mean, and in C4, the performance of ChatGPT is relatively better, although it is skewed to the left. In C1, txt2Model has both a higher mean and a skew to the right. In C3, the medians are almost the same, while in ChatGPT, the dispersion of data is more in the first quarter. In C5, both tools are skewed to the left, although the mean of txt2Model is higher and has a better performance.

5 Conclusion and future work

In this section, we reflect on conclusions and possible directions for future work.

5.1 Conclusions

A conceptual model is a simplified representation of a complex system or process. It is used to understand essential features and relationships of the system and ease communication between stakeholders. Conceptual models are used in fields such as science, engineering, and business and they are typically created using a visual diagram considering the textual requirements between the customers. Since generat-

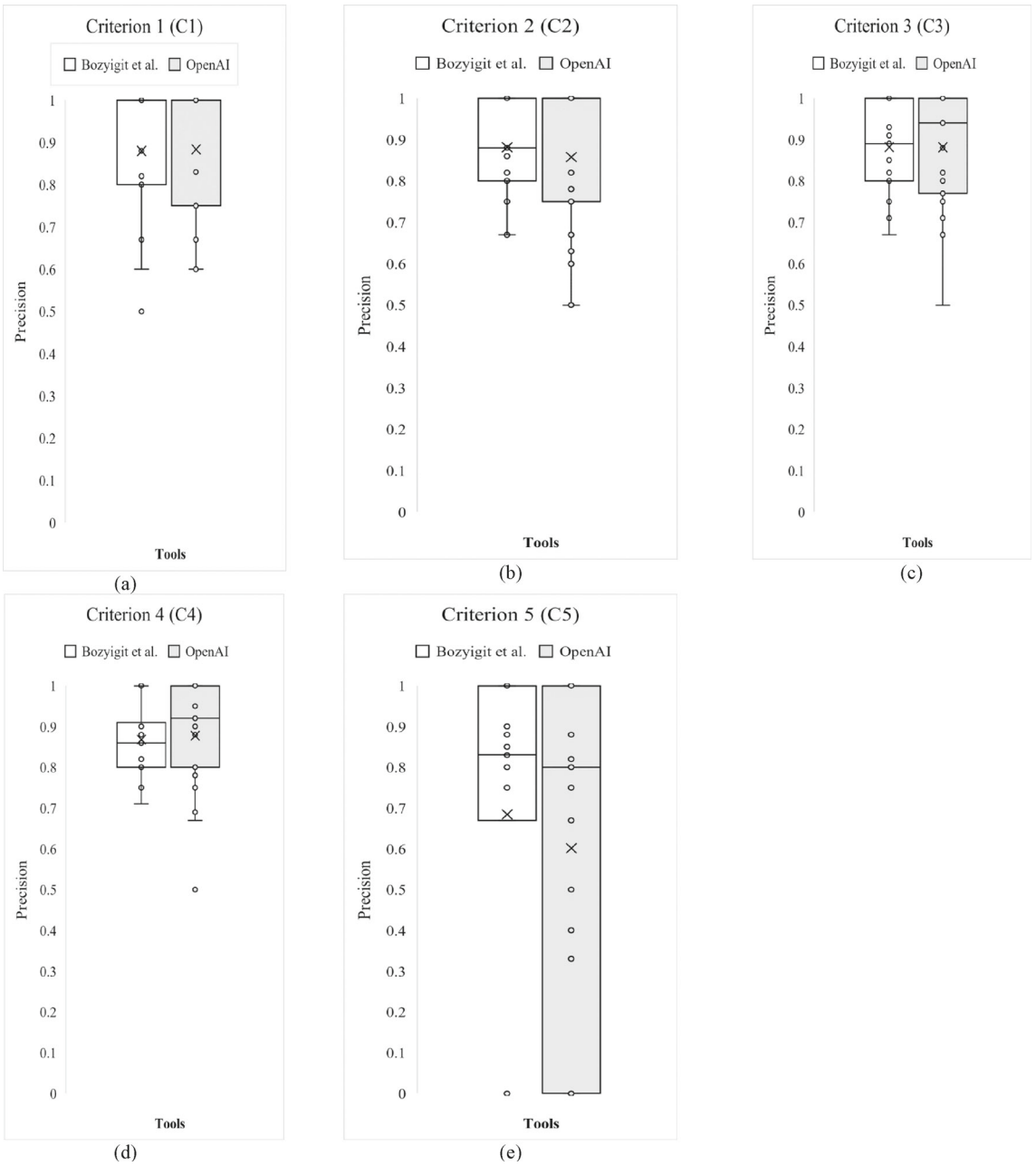

Fig. 10 Comparing the precision of txt2Model and ChatGPT tools

ing these models helps to understand and explain the complex system, transforming textual documents into visual diagrams automatically has attracted great attention from researchers in the recent decade.

We realized a need for better benchmarks for evaluating text-to-model approaches. Moreover, we realized a lack of realistic use cases in the existing datasets: their requirements are mostly trivial and include simple formatted sentences that could be easily analyzed by the transformation systems. In contrast, sophisticated statements may be required in real-life situations.

The main contributions of this paper are:
(1) Construction of a dataset using a systematic approach for use as a benchmark for text-to-model approaches. Concretely, we have created a well-defined benchmark dataset containing 120 software requirements in English and have made it publicly available on IEEEDataPort.[2] The dataset includes gold annotation and statistics files that describe various characteristics of the requirements, such as the number

[2] https://ieee-dataport.org/documents/dataset-text-requirements-models.

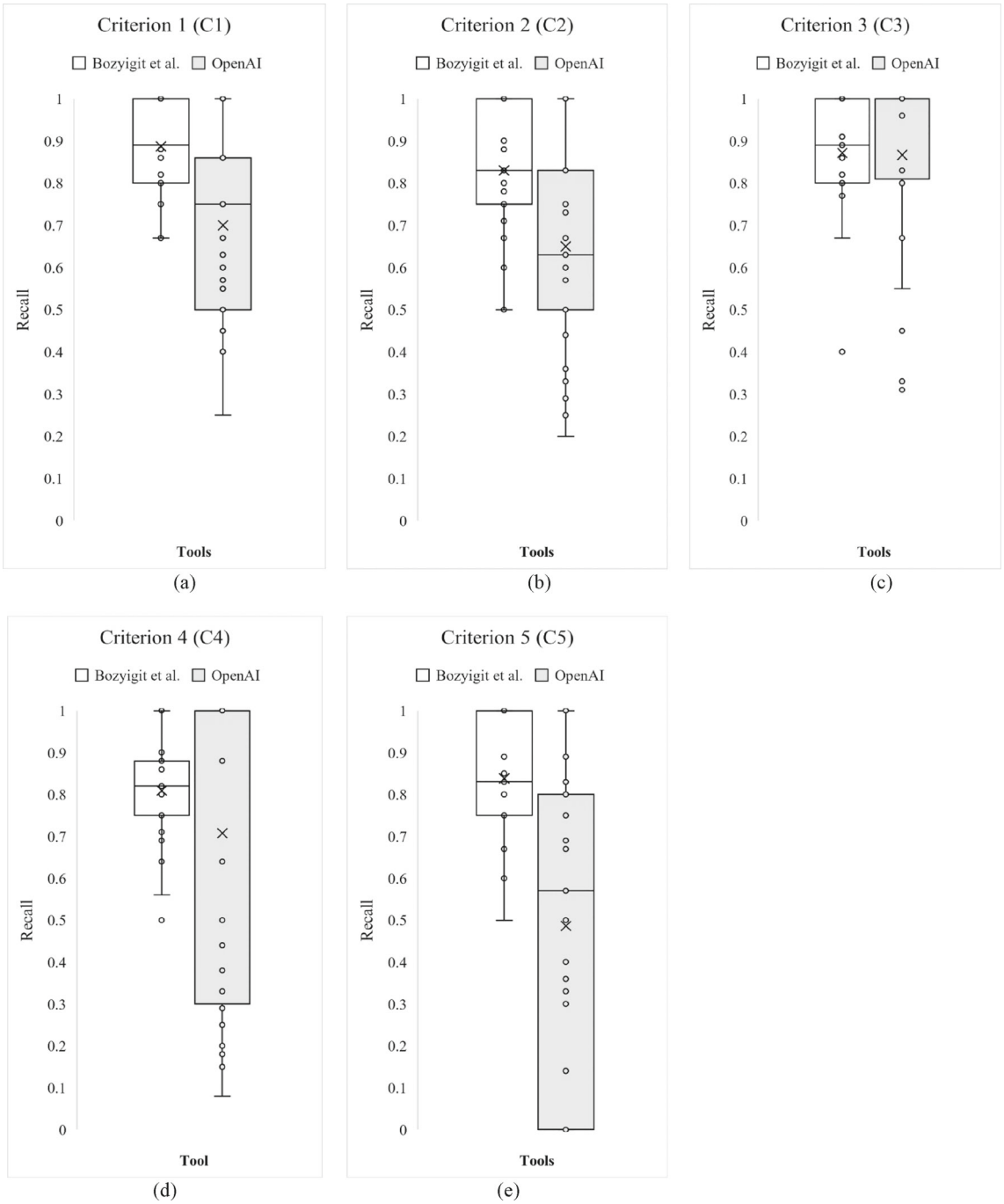

Fig. 11 Comparing the recall of txt2Model and ChatGPT tools

of sentences, words, classes, and also various measures of complexity of the requirements.

(2) Providing a benchmark for the proposed dataset has been one of the novelties of this article. By evaluating the dataset with the proposed benchmarks, it can be ensured that the proposed dataset can be used in a wide range of machine learning algorithms.

(3) We provide an empirical comparison of two tools based on the presented dataset and the proposed evaluation method.

5.2 Future work

In the future, for generating UML diagrams, we can take a more data-driven approach to understanding and determining relationship types. By using advanced data analysis tech-

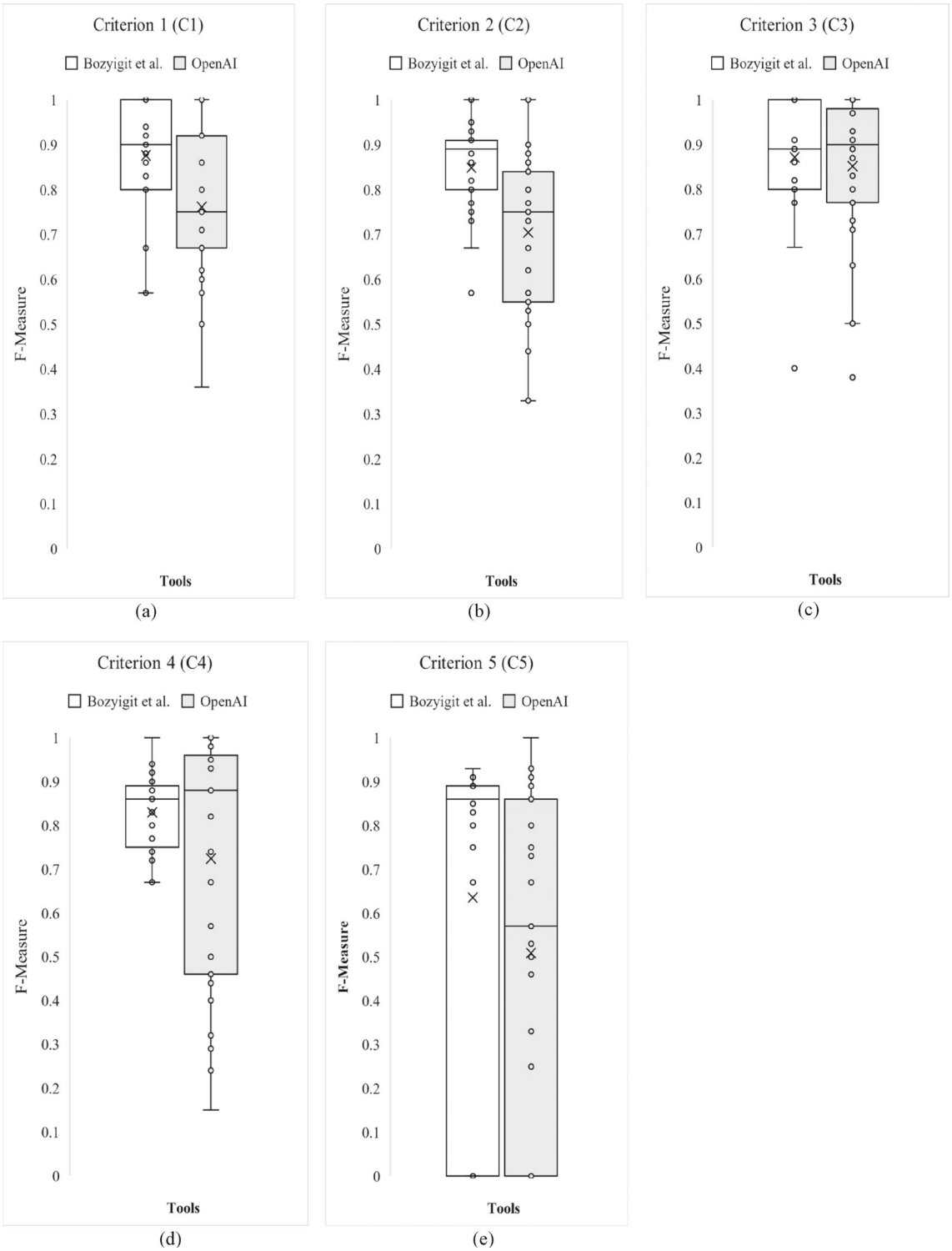

Fig. 12 Comparing the F-measure of txt2Model and ChatGPT tools

niques and machine learning algorithms, we can build more accurate models of relationship dynamics that can help us better predict and understand the various types of relationships that exist within a given system or software architecture. This serves to assist us in finding new design aspects and patterns that might be applicable to the system being modeled in addition to improving the accuracy of our relationship types. One potential application of this approach is to use

machine learning algorithms to identify similar requirements within a given dataset. By analyzing the requirements and their relationships to one another, the system can suggest similar UML diagrams that may be relevant to the requirements being analyzed. This can be a useful tool for system designers, as it can help them save time and effort by providing them with pre-designed UML diagrams that are tailored to their specific needs. Furthermore, one characteristic of the proposed dataset is that it can be used to develop models using machine learning algorithms that cluster data, categorize data according to particular attributes, or assist the user in creating models by creating recommender systems that recommend class, link, feature, cardinality, etc., to the user. Consequently, developing these kinds of data will open up novel opportunities for research.

In this paper, we focused on conceptual diagrams that capture the structural relations of the domain. Indeed, requirements also include information about the dynamic behavior of the system. Additional types of analyses can be developed that target the construction of behavior models, such as UML activity and sequence diagrams from requirements. This would require the addition of the benchmark data as well as the evaluation criteria.

Before conducting an analysis, like in our study, it is important to ensure that the dataset is clean and accurate. We have invested significant effort in cleaning and curating the dataset—by correcting grammar errors, clarifying ambiguous wording, and eliminating any incorrect or misleading information. In the future, we can also consider adding functionality to tools that can automatically preprocess the requirements and prepare them for analysis, making the tool more robust and potentially more practical for a wider range of datasets.

Finally, due to the restricted number of tools now available, we propose comparing more tools in the future. By doing this, we will enhance the evaluation's accuracy and overall quality.

Acknowledgements We acknowledge the sharing of cases by Jörg Kienzle and Günter Mussbacher. Ultimately the cases they provided were not included in our final dataset.

References

1. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.E., et al.: Objectoriented Modeling and Design, vol. 199. Prentice-hall Englewood Cliffs, NJ (1991)
2. Sagar, V.B.R.V., Abirami, S.: Conceptual modeling of natural language functional requirements. J. Syst. Softw. **88**, 25–41 (2014)
3. Özdağoğlu, A., Özdağoğlu, G.: Comparison of ahp and fuzzy ahp for the multi-criteria decision making processes with linguistic evaluations. İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi **6**(11), 65–85 (2007)
4. Landhäußer, M., Körner, S.J., Tichy, W.F.: From requirements to UML models and back: How automatic processing of text can support requirements engineering. Softw. Qual. J. **22**, 121–149 (2014)
5. Bozyigit, F., Aktaş, Ö., Kılınç, D.: Linking software requirements and conceptual models: a systematic literature review. Int. J. Eng. Sci. Technol. **24**(1), 71–82 (2021)
6. Satapathy, S.C., Jena, A.K., Singh, J., Bilgaiyan, S.: Automated Software Engineering: A Deep Learning-Based Approach. Springer (2020)
7. Jadhav, A., Kaur, M., Akter, F.: Evolution of software development effort and cost estimation techniques: five decades study using automated text mining approach. Math. Probl. Eng. **2022**, 1–17 (2022)
8. Mahmood, Y., Kama, N., Azmi, A., Khan, A.S., Ali, M.: Software effort estimation accuracy prediction of machine learning techniques: a systematic performance evaluation. Software Practice and Experience **52**(1), 39–65 (2022)
9. Giray, G., Bennin, K.E., Köksal, Ö., Babur, Ö., Tekinerdogan, B.: On the use of deep learning in software defect prediction. J. Syst. Softw. **195**, 111–537 (2023)
10. Khalilipour, A., Bozyigit, F., Utku, C., Challenger, M.: Machine learning based model categorization using textual and structural features. In: European Conference on Advances in Databases and Information Systems. Springer, pp. 425–436 (2022)
11. Khalilipour, A., Bozyigit, F., Utku, C., Challenger, M.: Categorization of the models based on structural information extraction and machine learning. In; International Conference on Intelligent and Fuzzy Systems. Springer, pp. 173–181 (2022)
12. Rahimi, S., Lano, K.C., Lin, C.: Requirement formalisation using natural language processing and machine learning: A systematic review. In: International conference on Model-Based Software and Systems Engineering, SCITEPRESS Digital Library, pp. 1–8 (2022)
13. Zhao, L., et al.: Natural language processing for requirements engineering: a systematic mapping study. ACM Comput. Surv. **54**(3), 1–41 (2021)
14. Falessi, D., Cantone, G.: The effort savings from using NLP to classify equivalent requirements. IEEE Softw. **36**(1), 48–55 (2018)
15. Li, G., Zheng, C., Li, M., Wang, H.: Automatic requirements classification based on graph attention network. IEEE Access **10**, 30080–30090 (2022)
16. Ahmed, S., Ahmed, A., Eisty, N.U.: Automatic transformation of natural to unified modeling language: a systematic review. In: 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), IEEE, pp. 112–119 (2022)
17. Habibullah, K.M., Gay, G., Horkoff, J.: Non-functional requirements for machine learning: Understanding current use and challenges among practitioners. Requirem. Eng. pp. 1–34 (2023)
18. Liu, Z., Lin, Y., Sun, M., Liu, Z., Lin, Y.: Representation learning and NLP. Representation Learning for Natural Language Processing, pp. 1–11 (2020)
19. Mich, L.: Nl-oops: From natural language to object oriented requirements using the natural language processing system lolita. Nat. Lang. Eng. **2**(2), 161–187 (1996)
20. Ibrahim, M., Ahmad, R.: Class diagram extraction from textual requirements using natural language processing (nlp) techniques. In: Second International Conference on Computer Research and Development, pp. 200–204 (2010). https://doi.org/10.1109/ICCRD.2010.71
21. Zhou, X., Zhou, N., Zhou, N.: Auto-generation of class diagram from free-text functional specifications and domain ontology (2004)
22. Bajwa, I.S.: Object oriented software modeling using NLP based knowledge extraction (2009)

23. Tripathy, A., Agrawal, A., Rath, S.K.: Requirement analysis using natural language processing. In: Fifth International Conference on Advances in Computer Engineering, vol. 26, p. 27 (2014)
24. Dori, D., Korda, N., Soffer, A., Cohen, S.: Smart: system model acquisition from requirements text. In: Proceedings of Business Process Management: Second International Conference, BPM: Potsdam, Germany, pp. 179–194. Springer (2004)
25. Abdelnabi, E.A., Maatuk, A.M., Abdelaziz, T.M., Elakeili, S.M.: Generating UML class diagram using nlp techniques and heuristic rules. In: 2020 20th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA), IEEE, pp. 277–282 (2020)
26. Deeptimahanti, D.K., Babar, M.A.: An automated tool for generating UML models from natural language requirements. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, IEEE, pp. 680–682 (2009)
27. Robeer, M., Lucassen, G., Van DerWerf, J.M.E., Dalpiaz, F., Brinkkemper, S.: Automated extraction of conceptual models from user stories via NLP. In: IEEE 24th International Requirements Engineering Conference (RE), pp. 196–205. IEEE (2016)
28. Lucassen, G., Robeer, M., Dalpiaz, F., Van Der Werf, J.M.E., Brinkkemper, S.: Extracting conceptual models from user stories with visual narrator. Requir. Eng. **22**, 339–358 (2017)
29. Bragilovski, M., Dalpiaz, F., Sturm, A.: From user stories to domain models: recommending relationships between entities (2023)
30. Ferrari, A., Spagnolo, G.O., Gnesi, S.: Pure: a dataset of public requirements documents. In: IEEE 25th International Requirements Engineering Conference (RE), pp. 502–505. IEEE (2017)
31. Ramackers, G.J., Griffioen, P.P., Schouten, M.B., Chaudron, M.R.V.: From prose to prototype: synthesising executable UML models from natural language. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 380–389. IEEE (2021)
32. Sedrakyan, G., Abdi, A., Van Den Berg, S.M., Veldkamp, B.P., Van Hillegersberg, J.: Text-to-model (tetomo) transformation framework to support requirements analysis and modeling. In: MODELSWARD, pp. 129–136 (2022)
33. Hamza, Z.A., Hammad, M.: Generating UML use case models from software requirements using natural language processing. In: 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), pp. 1–6, IEEE (2019)
34. Arora, c., Sabetzadeh, M., Briand, L., Zimmer, F.: Extracting domain models from natural-language requirements: approach and industrial evaluation. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 250–260 (2016)
35. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative AI in modeling tasks: an experience report with chatgpt and uml. Softw. Syst. Model, pp. 1–13 (2023)
36. Lano, K., Yassipour-Tehrani, S., Umar, M.: Automated requirements formalisation for agile MDE. In 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 173–180. IEEE (2021)
37. Bozyigit, F., Aktaş, Ö., Kılınç, D.: Automatic concept identification of software requirements in Turkish. Turkish Journal of Electrical Engineering and Computer Sciences (2019)
38. Wu, T., et al.: A brief overview of chatgpt: the history, status quo and potential future development. IEEE/CAA Journal of Automatica Sinica **10**(5), 1122–1136 (2023)
39. Sim, S.E., Easterbrook, S., Holt, R.C.: Using benchmarking to advance research: a challenge to software engineering. In: Proceedings of 25th International Conference on Software Engineering, pp. 74–83. IEEE (2003)
40. Kistowski, J.v., Arnold, J.A., Huppler, K., Lange, K.-D., Henning, J.L., Cao, P.: How to build a benchmark. In: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, pp. 333–336 (2015)
41. Bozyigit, F., Bardakci, T., Khalilipour, A., Challenger, M., Ramackers, G., Babur, O., Chaudron, M.R.V.: Dataset for: Text requirements to models. (2023). https://doi.org/10.21227/r9j6-nd62
42. Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. **80**(4), 571–583 (2007). https://doi.org/10.1016/j.jss.2006.07.009

Fatma Bozyigit received her B.Sc. degree in Computer Engineering from Eskişehir Osmangazi University in 2013 and went on to earn her M.Sc. and Ph.D. in Computer Engineering from Dokuz Eylül University in 2015 and 2019, respectively. Her scholarly pursuits are centered around natural language processing, text mining, and machine learning, including deep learning methodologies. Over her academic career, she has authored more than 30 peer-reviewed publications in these domains. Dr. Bozyigit's commitment to advancing the field of Computer Engineering extends beyond academia into industry, where she has applied her expertise as a research consultant for several high-profile software companies in Turkey. In 2023, she assumed the role of Valorization Manager in the Department of Computer Science at the University of Antwerp. Her current mission is to enhance the department's valorization efforts, actively promoting and increasing the engagement of researchers in valorization activities.

Tolgahan Bardakci is a Ph.D. holder and a member of the special academic staff at the University of Antwerp. His area of expertise is software engineering and testing. Before joining the university, he worked at Getir Technology as a Senior Software Test Engineer. With over 40,000 employees, Getir Technology is an international firm where he developed automation tests for back-end and mobile applications, showcasing his technical expertise in the field. As a Senior Test Engineer at Getir Technology, he was involved in various facets of testing beyond functional testing, including writing scripts for performance and security testing. In addition to his technical expertise, Tolgahan

has worked at culturally diverse and international companies from the beginning of his career. He has gained valuable experience in working with teams from various cultural backgrounds. Tolgahan has lived and worked in Turkey, Spain, and Belgium and has observed that continuous learning in a diverse and inclusive environment enhances productivity and engagement and promotes personal and professional well-being.

Alireza Khalilipour is a Ph.D. researcher in the Computer Science Department of the University of Antwerp. He has a B.Sc. and an M.Sc. degree in Software Engineering. His research interests include Code Refactoring, Soft Modeling, and Automated software Engineering. As an academic study, he could provide a Super Compiler to automatically transform sequential code to be distributed based on CORBA MICO-CCM middleware. He currently works on Intelligent Model Management in the MICSS-Laboratory and Flanders Make projects.

Moharram Challenger received his Ph.D. in IT from the International Computer Institute at Ege University in February 2016. From 2010 to 2013, he was a researcher and team leader of a bilateral project between Slovenia and Turkey (TUBITAK). From 2012 to 2016, he was the R&D director of UNIT IT Ltd., leading one national project funded by TUBITAK and two international software intensive projects in Europe called ITEA Model Writer and ITEA Assume. In 2017 and 2018, he has been a member of the faculty as an assistant professor at Ege University. From 2019 to 2020, he was a post-doc researcher at the University of Antwerp, working on Flanders Make projects. He is currently a tenure-track assistant professor in the Department of Computer Science at the University of Antwerp. His research interests include domain-specific modeling languages, multi-agent systems, cyber-physical systems, internet of things, and digital twins. Prof. Challenger is also a member of the IEEE and ACM.

Guus Ramackers is Assistant Professor at the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University. He is the research lead of the AI4MDE project which investigates the application of AI in model driven systems development. Previously, he has worked for 20 years at Oracle as Senior Principal Technical Product Manager on complex projects in R&D of software development tools, including UML, Components and Web Services. He has represented Oracle at standard bodies, most notable as Chair of the UML group and Lead Author of Activity modeling, and later Component modeling in the UML standard. He received a "Most Valuable Chair" award from OMG for his work on UML 2.0.

Önder Babur is an assistant-professor (tenure-track) in the Information Technology Group at Wageningen University and Research, The Netherlands. He holds a Ph.D. from Eindhoven University of Technology, M.Sc. from RWTH Aachen, Germany and B.Sc. from METU, Turkey. He was employed as a post-doctoral researcher in the Software Engineering and Technology group at Eindhoven University of Technology from 2019 to 2021, with which he is still affiliated as a guest researcher. He has further experience as a Software Engineer in Germany and as a researcher in Spain. His main research interests lie in the fields of model-driven engineering, systems modeling, software analytics, AI for software engineering and empirical software engineering. Over the years, he has participated in a number of research projects on automotive software engineering, digitalization and industrial automation, precision agriculture, and multiscale modeling. He has initiated and co-chaired the International Workshop on Analytics and Mining of Model Repositories, and the International Conference on Systems Modeling and Management. Around the main research lines, he has been collaborating with many international research groups and high-tech companies. With a career spanning over two decades.

Michel R. V. Chaudron With a career spanning over two decades. Michel R. V. Chaudron has made contributions to the field of Software Engineering, with a special focus on Software Architecture, (Component-based) Software Design and Modeling. His research is aimed at advancing the understanding of principles and practices of designing, modeling and using software architecture. His recent research looks at AI for Software Engineering and the architecture of digital twin systems. His academic career started during his M.Sc. where he spent one year as visitor in the Programming Research Laboratory of Oxford University. Subsequently, Dr. Chaudron earned his Ph.D. in Computer Science from the University of Leiden, The Netherlands by developing a methodology of formal program derivation using separation of coordination and computation for highly parallel computation. Since then, he has held various academic positions in Leiden, Gothenburg and Eindhoven. Currently he is a Professor of Software Engineering at the Department of Mathematics and Computer Science at Eindhoven University of Technology and has an adjoint professorships at ITB Bandung in Indonesia. Dr. Chaudron is a regular PC member and reviewer of respected software engineering conferences and journals. He is on the Steering Committee of Euromicro SEAA. In addition to his academic pursuits, Dr. Chaudron actively engages with the industry, collaborating with leading technology companies and participating in advisory roles for software development projects.