TYPE Mini Review
PUBLISHED 08 January 2024
DOI 10.3389/fcomp.2023.1306064

Check for updates

OPEN ACCESS

EDITED BY
Hector Florez,
Universidad Distrital Francisco Jose de Caldas, Colombia

REVIEWED BY
Tomasz Górski,
University of Gdansk, Poland

*CORRESPONDENCE
Daniel Siahaan
✉ daniel@if.its.ac.id

RECEIVED 03 October 2023
ACCEPTED 13 December 2023
PUBLISHED 08 January 2024

CITATION
Siahaan D, Fauzan R, Widyadhana A, Firmawan DB, Putri RR, Desnelita Y, Gustientiedina and Putrian RN (2024) A scoping review of auto-generating transformation between software development artifacts. *Front. Comput. Sci.* 5:1306064. doi: 10.3389/fcomp.2023.1306064

A scoping review of auto-generating transformation between software development artifacts

Daniel Siahaan[1]*, Reza Fauzan[2], Arya Widyadhana[1], Dony Bahtera Firmawan[1], Rahmi Rizkiana Putri[1], Yenny Desnelita[3], Gustientiedina[4] and Ramalia Noratama Putrian[3]

[1]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia, [2]Department of Electical Engineering, Politeknik Negeri Banjarmasin, Banjarmasin, Indonesia, [3]Department of Information System, Institut Bisnis dan Teknologi Pelita Indonesia, Pekanbaru, Indonesia, [4]Department of Informatics, Institut Bisnis dan Teknologi Pelita, Pekanbaru, Indonesia

Every process within software development refers to a specific set of input and output artifacts. Each artifact models specific design information of a system, yet they complement each other and make an improved system description. The requirements phase is an early stage of software development that drives the rest of the development process. Throughout the software development life cycle, checking that every artifact produced in every development stage should comply with the given requirements is necessary. Moreover, there should be relatedness between elements within artifacts of different development stages. This study provides an overview of the conformity between artifacts and the possibility of artifact transformation. This study also describes the methods and tools used in previous studies for ensuring the conformity of artifacts with requirements in the transformation process between artifacts. It also provides their applications in the real world. The review identified three applications, seven methods and approaches, and five challenges in ensuring the conformity of artifacts with requirements. We identified the artifacts as class diagrams, aspect-oriented software architecture, architectural models, entity relationship diagrams, and sequence diagrams. The applications for ensuring the conformity of artifacts with requirements are maintaining traceability, software verification and validation, and software reuse. The methods include information retrieval, natural language processing, model transformations, text mining, graph-based, ontology-based, and optimization algorithms. The benefits of adopting methods and tools for ensuring the conformity of artifacts with requirements can motivate and assist practitioners in designing and creating artifacts.

KEYWORDS
design artifacts, resource use efficiency, requirements conformity, software artifact, systematic literature review

1 Introduction

A requirement is a specification or capability that a system must meet. A requirement can be derived directly from user needs and can be specified in the software requirements specification, a contract, a standard, or a term of reference. Regarding work material, schedule, and commitment, requirements serve as the foundation for the software development life cycle (SDLC) (Sagar and Abirami, 2014). Documenting requirements is vital in requirements engineering (RE), regardless of the RE mechanism. Since requirements are generally written to communicate with various stakeholders, they should be easy to understand. Requirements can be recorded in several ways, such as use cases, user stories, formal specifications, or personalized document models, but the most frequently used method is textual explanations (Bozyigit et al., 2021). Each requirement representation should have a certain degree of conformity with another or with other artifacts within the software development process. Requirement artifacts are the outputs of the software requirements phase, which can be a list of requirements, business processes, use case diagrams, etc.

There are two categories of requirements: (1) functional requirements (FRs) and (2) non-functional requirements (NFRs) (Sommerville, 2015). FRs describe a system regarding the services it must deliver, how it should respond to specific inputs, and how it should behave in specific circumstances. FRsFRs can also clearly state what a system must not do in certain situations. NFRs are limitations on the services or features of a system, including time limits, implementation phase constraints, and standards-based constraints. NFRs are sometimes applied to the whole system rather than specific system functions or utilities. Requirements are not self-contained, and one requirement always generates or constrains others.

Any textual or graphical record is an artifact. User stories use cases, error reports, and other illustrations are examples of artifacts, which may be at any scale or level of granularity (Ghazi and Glinz, 2017). Each software development activity requires the use or successful implementation of some artifacts. These artifacts are an SDLC (Tufail et al., 2017; Fauzan et al., 2018) pillar. Furthermore, artifacts are used by various people with varying tasks and demands depending on their role in the project. There is rarely one optimal type of artifact that can meet the needs of all participants. Ventures thus need to deploy a wide range of artifacts, which runs the risk of contradictions or inefficiencies arising from the interdependencies of several artifacts. In RE, efficient alignment of specifications artifacts is critical (Liskin, 2015). Various stakeholders with differing perspectives influence the implementation and design of complex systems. Due to the diverse capabilities of stakeholders, various artifacts and heterogeneous requirements are essential parts of the development process (Haidrar et al., 2017). It is essential to check that final system artifacts meet the requirements to deal with this complication.

In light of the explanation above, requirements documents are closely related to other artifacts (Irshad et al., 2018). All the documents must be integrated and have the same understanding of the user requirements (Ferrari et al., 2017; Muñoz-Fernández et al., 2017). However, in a project, requirements documents may be created by several different people (Noll et al., 2017). Moreover, requirements documents are generally written in natural language (NL), which can lead to misunderstandings between stakeholders on the details of requirements. Several other studies (Yue et al., 2011b; Souza et al., 2019; Bozyigit et al., 2021) have performed systematic literature reviews on the conformity of specific artifacts with requirements. However, an overview of the conformity of artifacts in general within the software development process is necessary as a basis for ensuring the conformity of artifacts with requirements. A requirement is closely related to the accompanying artifact because many artifacts can be generated from a requirement.

This review summarizes the current research related to requirements conformity with other artifacts. We limit our scope to the studies that have developed a method, an approach, a technique, a framework, or a tool that generates requirements artifacts (such as user story, textual requirements, use case diagram, etc.) from other artifacts (other requirements artifacts, design artifacts, implementation artifacts, testing artifacts, etc.); developed a method, an approach, a technique, a framework, or a tool that generates design/implementation/testing artifacts from requirements; developed method, an approach, a technique, a framework, or a tool that measure similarity or consistency between two requirements artifacts or between a requirement artifact and a design/implementation/testing artifact; developed a method based on method that generates requirements artifacts; compared various automated artifact generation methods (including requirements artifacts), or; reported real world experiences on various artifact automated artifact generation methods (including requirements artifacts). We summarize the methods, approaches, techniques, and tools that have been proposed and identify which challenges they aim to overcome. We also identify the applications of the methods and tools. Our study is intended to help practitioners better understand the methods and tools used to ensure the conformity of artifacts with requirements.

2 Related works

To the best of our knowledge, a limited number of review studies focus on the conformity of artifacts with requirements. The studies explore various automation methods for converting requirements into Unified Modelling Language (UML) diagrams (Yue et al., 2011b; Souza et al., 2019; Bozyigit et al., 2021), and automation methods for generating codes from Unified Modeling Language (UML) diagrams (Mehmood and Jawawi, 2013). The following is a detailed explanation of these related works.

Yue et al. (2011b) conducted a systematic literature review of transformation methods for converting consumer expectations into investigation models. They concluded that a perfect methodology for converting requirements into investigation models has the following attributes: (1) requirements should be simple to report using the arrangement necessitated by the methodology; (2) the examination models created should be complete; (3) the methodology should contain as few changes as possible (high efficiency); (4) the methodology ought to be automated; and (5) the methodology should be understandable. Seven of the 16 studies reviewed were able to change requirements directly into analysis models. In contrast, the others used midway models to bridge the gap between them. They discovered that UML diagrams were the most commonly used among the examined methods of representing generated research models. They suggested that further research on transformation approaches should also address their quality characteristics: efficiency, usability, interoperability, scalability, and extensibility.

Mehmood and Jawawi (2013) conducted a systematic mapping study of aspect-oriented model-driven code generation. Their research sought methods for applying aspect-oriented principles in a model-driven engineering (MDE) context. The reviewed studies often use an automated method to create templates. They found that models are the primary focus in an MDE framework and are used to visualize an executable concept of the system and automate the development of a working software system. Hence, modeling correction and completion need more attention, as does finding ways to verify models to incorporate aspects into the broader context. They also identified that aspect-oriented code creation is a difficult problem to solve, and only a few proposals have attempted to do so. Moreover, all of the proposed solutions could only produce portions of code. Furthermore, there are no documented code generation methods based on UML sequence and statechart diagrams, which present the behavioral viewpoint of the system. This should be examined more thoroughly in the future.

Souza et al. (2019) conducted a systematic mapping study on deriving architectural models based on requirements specifications. Their examination intended to outline the current methods for generating architectural models based on requirements specifications. The bulk of the reviewed studies reviewed focused on deriving system designs from specifications. Others were concerned with understanding the specifications, providing decision-making support, reusing architectural expertise, promoting modularization in software design, or providing requirements and architecture traceability support. Many experiments focused solely on the architect's experience, and only a few acknowledged their limitations. Those that did cite apparent uncertainty and a lack of consistent terminology, stating that portraying the design of a software system is a highly complex challenge that is impossible to solve in a single model. They also discovered several unsolved problems that can be further investigated. It included domain-specific methods that addressed a broader spectrum of NFRs. Mainly, external NFRs address methods' limitations, standards, or common understanding; explicit or tacit knowledge; supporting tools; evaluation methods; and requirements satisfaction.

Bozyigit et al. (2021) conducted a systematic literature review on linking software requirements and conceptual models. They discovered that the bulk of the studies were mostly focused on coping with requirements written in English. The growing number of groundbreaking works analyzing English-language documentation contributes significantly to the information engineering field. However, working in languages other than English helps these systems reach a larger audience. They discovered that most scientific experiments focused on constructing UML class diagrams despite the fact that 14 types of UML diagrams are used to depict the various aspects and characteristics of an expected software. Furthermore, the recognition of relationships among the components of the produced models was not completed correctly. The inability to establish relationships may limit traceability between the design and implementation phases. They suggested creating a well-designed algorithm that contains more complex transition rules to address these flaws. For future analysis, they suggested developing novel methods with the following features: (1) thoroughly determining all types of relationships, (2) producing diagram types other than class diagrams, (3) creating source code for more than one programming language, and (4) making a large-scale dataset of various programming issues.

There have been literature reviews on the conformity of artifacts with requirements in generating artifacts. However, all other reviews or mapping studies have focused only on specific artifacts (e.g., analysis models, architectural models, or code). Table 1 summarizes the selected literature on the conformity of artifacts with requirements in general. Yue et al. (2011b) explained the methods commonly used to generate artifacts at the analysis stage. Mehmood and Jawawi (2013) shows the relationship between requirements and artifacts in aspect-oriented modeling. Souza et al. (2019) and Bozyigit et al. (2021) explain all the methods used to generate artifacts. Our review aims to explain the methods applied to check conformity between requirements representations and between requirements artifacts and other artifacts or to generate conformance representations of requirements or different artifacts. It also aims to explain the challenges that are encountered in its generation. We identify the different applications of ensuring the conformity of artifacts with requirements. We summarize the methods, techniques, approaches, and tools used in studies and research within the past decade (2010–2020) and state the challenges in ensuring conformity of artifacts and requirements. We determine whether significant development has been in the investigated topic and identify the areas lacking research.

3 Overview of selected studies

The study follows The PRISMA 2020 guidelines for reporting systematic reviews (Page et al., 2021) as the reference. We designed a structured search strategy to delve into all acquirable scientific sources relevant to the objective of this review following distinguishing our research preferences and problems.

As shown in Table 2, the protocol included specifying the search space, which included electronic databases. The studies were initially retrieved from electronic sources. The search terms used were "Compliances," "Software requirements compliances," "Software requirements artifacts," "Requirement compliances artifacts," "Software requirements," and "Requirements with other artifacts." At this stage, we found 370 studies, including 61 studies from ACM Digital Library, 82 from IEEE Xplore, 105 from SpringerLink, and 122 from ScienceDirect. We conducted our search on March 9, 2023, and searched for papers published between January 2010 and our search date.

We used a list of inclusion and exclusion criteria to narrow down the 370 studies. We used the following inclusion criteria to ensure that all selected publications for review are relevant and cover state-of-the-art research.

I1 This study is a peer-reviewed publication.
I2 The study is written in English.
I3 The study is related to the search terms.
I4 The study is an empirical research paper, experience report, or workshop paper.
I5 The study was published between January 2010 and December 2020.

In order to reduce missed selection, bias, and muddling, we used a set of exclusion criteria. These exclusion criteria should improve the probability of finding an important and insightful association.

TABLE 1 Summary of selected literature on ensuring conformity with requirements in general.

References	Goal	Research questions/goals
Yue et al. (2011b)	Examine existing written works that change text-based requirements into investigation models, feature open issues, and offer ideas for future research directions.	RQ1: What are the various techniques for turning requirements into analysis models? RQ2: What are these methods' present limitations? RQ3: What are the open topics that need to be investigated further?
Mehmood and Jawawi (2013)	Give an overview of existing exploration on aspect-oriented modeling and code generation to find important work and recognize areas for future examination.	RQ1: What are the most frequently discussed aspects of aspect-oriented programming and model-driven code generation? How well have these issues been researched? Furthermore, what kinds of contributions have been made so far? RQ2: What are the most common places to publish research on aspect-oriented modeling and generation of aspect-oriented code? RQ3: What various forms of study are presented in the literature, and how much of it has been addressed?
Souza et al. (2019)	Give a thorough outline of the current methods to obtain architectural models from requirements specifications. Offer an exploration guide to shift the research focus to address the recognized limits and open issues that require further examination.	RQ1: What is the method's key goal? RQ2: What are the implementation domains of the method? RQ3: What is the starting point for the method? RQ4: What are the advantages of the system for users? RQ5: What are the limitations of the method? RQ6: What does the approach cover in terms of (1) knowing the problem, (2) identifying a solution, and (3) assessing the solution? RQ7: How does the system use architectural views? RQ8: Does the system specify a language or notation for describing the objects produced? RQ9: To what extent are the enabling methods automated? RQ10: What criteria are used to test the method? RQ11: How is the method's output quality validated?
Bozyigit et al. (2021)	Provide an overview of current methods for automatically converting software specifications into conceptual models, as well as an assessment of their approaches, functionalities, datasets used, validation methods, generated model forms, languages supported, and future research needs.	RQ1: Which languages are examined to convert software requirements into conceptual models? RQ2: What are the strategies for converting requirements into a computational model? RQ3: What types of mathematical models will the checked structures generate? RQ4: What are the specifics of the databases used in the experiments reviewed? RQ5: What techniques were used to determine consistency in the experiments reviewed?

E1 The study does not focus explicitly on requirements conformity with other artifacts.

E2 The study focuses on coverage outside of the conformity of artifacts with requirements (e.g., the conformity between different requirements representations or between requirements with non-requirements artifacts).

E3 The study is an opinion, point of view, keynote, discussion, editorial, comment, lecture, preface, narrative article, or presentation in slide format without accompanying text.

Table 3 displays the findings of the initial search, which yielded 370 publications. Our dataset included only published, peer-reviewed papers (I1). The researchers thoroughly examined the titles and abstracts of the studies based on the inclusion criteria (Round 1). The majority of the studies that were collected satisfied inclusion criteria I2, I3, and I5. Due to the inadequacies of web indexes in extending the search string to the entire body of the paper, a significant number of results were eliminated, resulting in the selection of 92 studies after the first round of classification. We also checked that the papers we found were empirical research papers, experience reports, or workshop papers (I4). The researchers then tested the selected papers in Round 2 against the exclusion criteria (E1, E2, and E3). A face-to-face consensus conference was held to review the agreements and differences in the evaluation of the researchers. When two researchers could not agree on whether a paper should be included, we read the entire paper. We then omitted studies based on the exclusion criteria. For instance, there are some publications that were omitted because although studies such as Abbas et al. (2021) discuss about conformity of artifacts, they do not explicitly focus on requirements. Other studies focus on modeling requirements (Dalpiaz et al., 2021), generating artifacts (Sunitha and Samuel, 2018), or measuring artifact integrated quality (Landhäußer et al., 2014). In total, 60 articles were omitted from the 92 pre-selected studies after applying the inclusion criteria because they did not address any subject specifically relevant to our investigation (E1–E4). The remaining thirty-two studies fulfilled the above-listed procedures and strategies for ensuring that the criteria were met. As a result, we narrowed down our final list to thirty-two studies (see the two farthest-right columns in Table 3).

TABLE 2 Search sources.

Type	Source names
Electronic databases	ACM digital library, IEEE Xplore, SpringerLink, ScienceDirect
Searched items	Journal articles, workshop and conference papers, and book chapters
Search applied on	Full text, skipping any articles important to the research objective but that did not contain our search keywords in titles or abstracts
Language	English
Period of publication	January 2010–December 2020

Finally, we performed backward snowballing according to the guidelines set by Wohlin (2014). Other papers are searched based

TABLE 3 The total studies found in each round of the systematic search.

Database	Retrieved	Round-1		Round-2	
		Included	Excluded	Included	Excluded
ACM digital library	61	13	48	3	10
IEEE Xplore	82	23	59	7	16
SpringerLink	105	24	81	10	14
ScienceDirect	122	32	90	12	20
Total	370	92	278	32	60

on the reference list of papers selected in the previous section. Of. From the backward snowballing on the thirty-two papers that had been selected, 42 new papers were identified. The 42 papers were filtered using the inclusion and exclusion criteria as previously described. Then the 42 papers were also assessed based on the same assessment criteria as previously described. Six papers (55%) were considered excellent, three papers (22%) were considered to be very good, and two papers (18%) were considered to be good. In the end, 11 additional papers were selected. There were 43 selected papers to be reviewed in total.

The distribution of research publishing origins is shown in Table 4. Of the forty-three studies, around 47% (n = 20) were published in journals, 49% (n = 21) in conferences, and 4% (n = 2) in book chapters. Our results in Table 4 show that the Journal of Systems and Software had the highest number of publications, followed by the India Software Engineering Conference's publication venue, showing that authors slightly preferred the Journal of Systems and Software. The years of publication, shown in Figure 1, indicate that the conformity of artifacts with requirements still attracts research interest. Figure 2 illustrates the distribution of peer-reviewed articles published between 2010 and 2020. Studies on this topic have been published yearly, with an average of three studies per year.

4 Applications of ensuring conformity of artifacts with requirements

Based on our findings, there are three applications for ensuring conformity of artifacts with requirements: maintaining traceability, software verification and validation, and software reuse. Of the collected studies, 51% were related to maintaining traceability. Of the collected studies, 46% (23 papers) were related to maintaining traceability. Another 42% (21 papers) proposed software verification and validation. Finally, 12% (6 papers) proposed software reuse.

Maintaining traceability in software engineering refers to monitoring job objects during the production process (Vale et al., 2017). There must be a trace, which is defined as a stated triplet of elements: a target artifact, a source artifact, and a connection linking the two artifacts to the specific requirements artifacts to verify the conformity of the artifacts with the requirements. The ability to create and use traces is known as traceability (Cleland-Huang et al., 2012). Traceability is essentially the ability to connect data in different objects and analyze the relationship between these data. As a result, providing navigable connections between data stored within objects is essential for achieving traceability (Cleland-Huang et al., 2012). Engineers can execute job tasks like effect analysis, identifying reusable objects, and specifications validation more reliably if they have access to trace information (Borg et al., 2014). By defining accurate associations between objects, trace ties can be created to ensure the conformity of artifacts with requirements. Traceability is generally achieved by defining rules connecting input and output artifacts.

Software verification and validation (V&V) determines whether the software meets its given requirements (Hsueh et al., 2008). The process includes checking programs, code, design, and documents. There are several verification methods, including reviews, walkthroughs, and inspections. Verification can be addressed by asking, "Are we building the product correctly?" Validation is the method of determining whether the software satisfies the specifications and desires of the customer. It entails testing and validating the commodity. Validation methods include non-functional testing, white-box testing, and black-box testing. Validation can be addressed by asking, "Are we building the right product?" (Sainani et al., 2020). Ensuring the conformity of artifacts with requirements can be applied to software V&V. The methods and techniques related to ensuring the conformity of artifacts with requirements provide a thorough analysis and understanding of the requirements. They can also provide guidelines and assistance in software V&V. The resulting artifacts of the methods and techniques that ensure the conformity of artifacts with requirements are closely constructed to the requirements of the software. The artifact is also designed to cater to individuals without technical expertise, providing them with the necessary tools to effortlessly generate, disseminate, and oversee fully operational applications across several client platforms (Pérez-Álvarez and Mos, 2020).

Software reuse is the method of explicitly specifying a series of structured procedures for stating, constructing, using, and adapting previously acquired software components to create new software (Mili et al., 1998; Irshad et al., 2018). It is a method of creating new applications from existing software and predefined components to increase software efficiency. Reusing apps has various advantages, including (1) increased dependability, (2) increased effectiveness, (3) accelerated development; (4) improved device interoperability; (5) the ability to create applications with fewer people; (6) lower costs of device servicing; (7) building structured applications; and (8) high-quality software and a significant competitive edge (Dabhade et al., 2016). The methods and techniques for ensuring conformity of artifacts with requirements can be applied to software reuse due to the thorough analysis and understanding of

TABLE 4 The distribution of studies by publication venues.

Publication source	Type	Number
ACM Transactions on Software Engineering and Methodology (TOSEM)	Journal	1
Arabian Journal for Science and Engineering	Journal	1
Computational Intelligence in Data Mining	Conference	1
Data & Knowledge Engineering	Journal	1
European Conference on Modelling Foundations and Applications	Conference	2
IEEE Transactions on Software Engineering	Journal	1
India Software Engineering Conference	Conference	3
Information and Software Technology	Journal	1
Intelligent Computing, Communication, and Devices	Book Chapter	1
International Conference on Advanced Informatics for Computing Research	Conference	1
International Conference on Advances in Computing	Conference	1
International Conference on Computer Research and Development	Conference	1
International Conference on Computer Science and Computational Intelligence	Conference	1
International Conference on Computer Science and Information Technology	Conference	1
International Conference on Fuzzy Systems and Knowledge Discovery	Conference	1
International Conference on Information Science and Applications	Conference	1
International Conference on Intelligent Systems: Theories and Applications (SITA)	Conference	1
International Conference on Modeling Simulation and Applied Optimization	Conference	1
International Conference on Swarm Intelligence	Conference	1
International Conference on System Analysis and Modeling	Conference	1
International Joint Conference on Computer Science and Software Engineering	Conference	1
International Journal of Computer Science and Informatics	Journal	1
International Journal of Database Theory and Application	Journal	1
International Symposium on Software Testing and Analysis	Conference	1
International Workshop on Artificial Intelligence for Requirements Engineering	Conference	1
Journal of King Saud University Computer and Information Sciences	Journal	1
Journal of Systems and Software	Journal	7
Procedia Computer Science	Journal	1
Relating Software Requirements and Architectures	Book Chapter	1
Requirements Engineering	Journal	1
Soft Computing	Journal	1
Software Quality Journal	Journal	1
Software: Practice and Experience	Journal	1
World Congress on Nature and Biologically Inspired Computing	Conference	1

the requirements needed to select existing software components and modify them to fit the requirements of the software under development. The artifacts generated through these methods and techniques can also be reused for future development.

5 Methods, techniques, approaches, and tools

We found seven types of approaches used in the forty-three studies, namely information retrieval (IR), natural language processing (NLP), model transformation (MT), text mining (TM), graph-based, ontology-based, and optimization algorithms. The following is a brief explanation of the approaches used. Table 5 shows the input artifact, the method used, and the generated artifacts for each study where AD is Activity diagram, CD is Class diagram, ERD is Entity Relationship Diagram, GA is Genetic Algorithm, IR is Information Retrieval, MT is Model Transformation, NLP is Natural language Processing, SD is Sequence diagram, ST is Statechart Diagram, TM is Text Mining, and UCD is Use case diagram.

5.1 Information retrieval

IR is the process of locating content (usually documents) of an unstructured nature (usually text) that meets a specific knowledge requirement from extensive collections (typically stored on PCs) (Harman et al., 2019). The phrase “unstructured data” refers to data that does not follow a simple, semantically obvious, and computer-friendly format. Supporting users in searching or filtering document sets and further processing a range of collected documents is included in the IR field (Schütze et al., 2008). Some common sub-tasks of IR are (1) tokenization, which separates a piece of text into smaller units called tokens; (2) stemming, which reduces derived words to their base forms; and (3) identifying and removing stop words, such as prepositions and pronouns. IR matched words to the artifact component based on pre-existing knowledge in the papers we found. Knowledge can be ontology (Robles et al., 2012) or data from previous projects. IR performance on papers is tested using precision, recall, and over-specification (Harmain and Gaizauskas, 2000). Precision indicates whether the output of the given solution is complete and shows the relevance of the human analyst with the system being built. Recall shows the accuracy of the system’s output by displaying the correct number of outputs. Over-specification assesses the amount of excess information within the output solution.

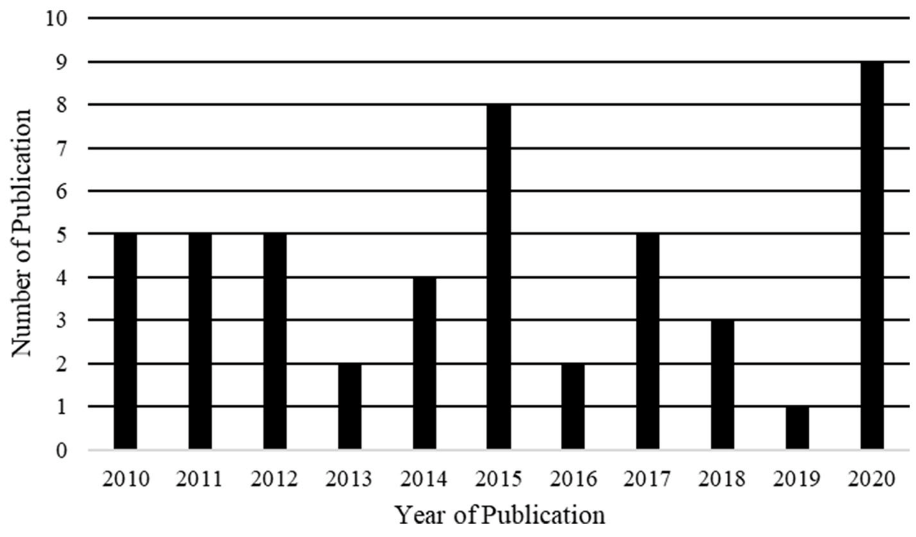

FIGURE 1
Distribution of selected studies by year.

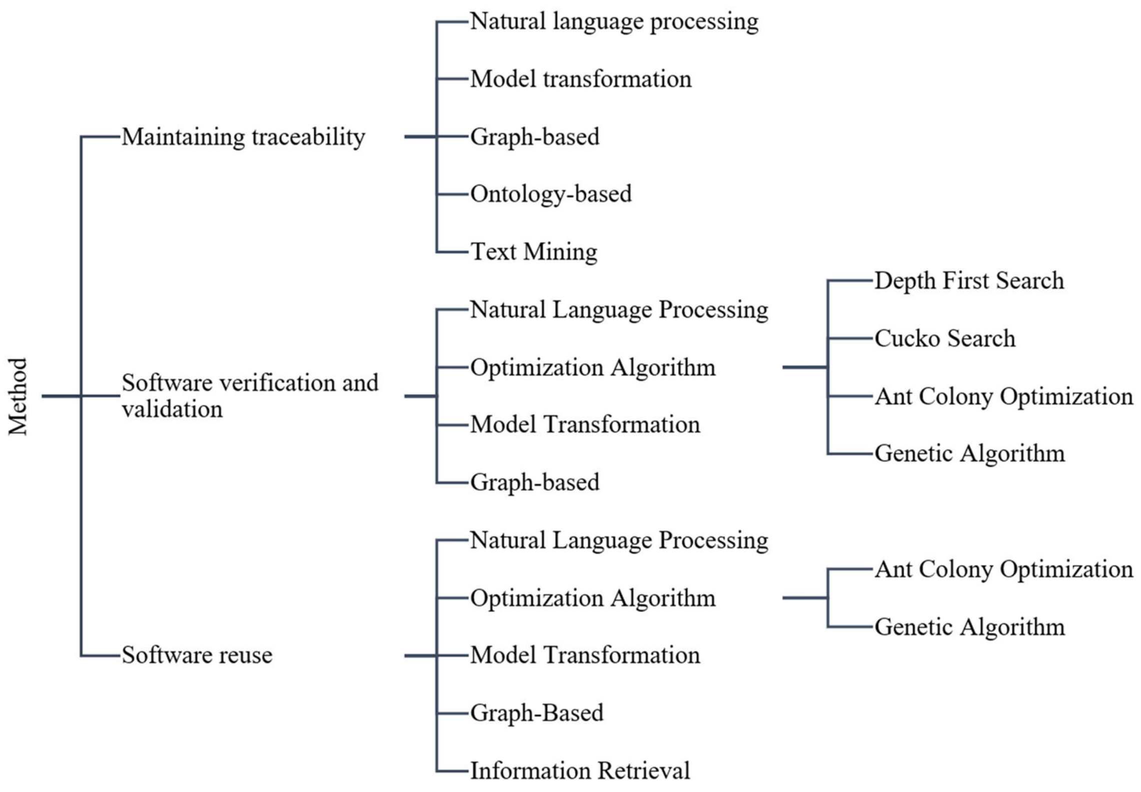

FIGURE 2
Methods based on applications.

5.2 Natural language processing

NLP applies computational techniques to analyze and synthesize NL and speech (Nadkarni et al., 2011; Pérez-Álvarez and Mos, 2020; Dinesh et al., 2021). It allows computers to comprehend both written and spoken human language. Systems interpret language on several levels. The first level is the morphological level, which treats morphemes as the constituent parts of a word. The second level is the linguistic level, which investigates how morphemes merge to form words and how minor variations can alter the meaning of the final word. The third level is the syntactic level, which considers how word order and dependency

TABLE 5 The input artifact, the method used, and the output artifacts of each study.

Study: Input → Output Artifact	Method
Bowman et al. (2010): Ill-suited CD → Optimal CD	GA
Ibrahim and Ahmad (2010): Textual requirements → CD	NLP, IR, and MT
Yue et al. (2010): UCD → AD	MT
Sanchez et al. (2010): Aspect-oriented requirements → Aspect-oriented soft. arch.	MT
Chen and Li (2010): UCD → Test case	MT
Deeptimahanti and Sanyal (2011): Textual requirements → UCD and CD	NLP and TM
Elbendak et al. (2011): UCD → Class diagram	NLP, IR, and MT
Yue et al. (2011a): UCD → ST	NLP and IR
Junior and Vijaykumar (2012): Textual requirements → ST	MT
Castro et al. (2012): Requirements in i* → Architectural model	MT
Colombo et al. (2012): Problem frames → Architectural model	MT
Lv and Xie (2012): Textual requirements → ERD	Ontology-based and MT
Sarkar et al. (2012): Use case diagram → CD	NLP, IR, and MT
Tripathy and Mitra (2013): AD and SD → Test case	Graph-based and depth-first search
Masoud and Jalili (2014): Textual requirements → CD	Graph-based and TM (clustering)
Sagar and Abirami (2014): Textual requirements → CD	NLP, IR, and MT
Thakur and Gupta (2014): UCD → SD	NLP, IR, and MT
Abirami et al. (2015): Textual requirements → CD	NLP, IR, TM (clustering), and MT
Jena et al. (2015): Sequence diagram → Test case	Graph-based and GA
Sharma et al. (2015): Textual requirements → CD	NLP, IR, and MT
Tawosi et al. (2015): Textual requirements → CD	NLP, IR, graph-based, and ant colony
Wang et al. (2015): Use case diagram → Test case	NLP and IR
Yue et al. (2015): UCD → AD, CD, and SQ	NLP and MT
Ben Abdessalem Karaa et al. (2016): Textual requirements → CD	NLP and MT
Khan and Mahmood (2016): Use case maps → Test case	MT
Ahmed et al. (2017): Textual requirements → CD	NLP, IR, and MT
Jaiwai and Sammapun (2017): Textual requirements → CD	NLP, IR, and MT
Kim et al. (2017): Design patterns → CD and SD	MT
Lucassen et al. (2017): User stories → ERD	NLP, IR, and MT
Meiliana et al. (2017): AD and SD → Test case	Graph-based and depth-first search
Azam et al. (2018): Feature diagrams → Test case	GA
Elallaoui et al. (2018): User stories → UCD	NLP, IR, and MT
Khari et al. (2018): Control flow graph → Test case	Graph-based and cuckoo search
Hamza and Hammad (2019): Textual requirements → UCD	NLP, IR, and MT
Omar and Baryannis (2020): Textual requirements → ERD	NLP and IR
Sahoo and Ray (2020): Control flow graph → Test case	Graph-based and particle swarm
Sankar and Chandra (2020): State transition diagram → Test case	Graph-based and ant colony
Pérez-Álvarez and Mos (2020): Textual requirements → Application	NLP and MT
Sainani et al. (2020): Contracts → Textual Requirement	NLP and IR
Wang et al. (2020): Textual requirements → Test Case	NLP and MT
Arrieta et al. (2020): Textual requirements → Test Case	NLP and MT
Yang et al. (2019): UCD, CD, SQD → Prototype	MT
Haris and Kurniawan (2020): Textual requirements → requirement sentence	NLP and MT

affect the interpretation of a sentence. The fourth level is the semantic level, which reflects how the sense of terms within a sentence influences the interpretation of the actual words. The fifth level is the discourse level, which examines how sentence order and composition influence the interpretation of sentences. The last level is the practical level, which explores the effect of the foundation of words or sentences on situational experience and world understanding.

The goal of NLP is to improve IR. The following are some examples of NLP sub-tasks: (1) lemmatization (combining inflected forms of a word so that they can be analyzed as a single object); (2) dependency parsing (determining whether any of the terms in a sentence are related); (3) part-of-speech (POS) tagging (identifying terms in a sentence based on their linguistic properties, such as noun, pronoun, and adjective); and (4) named entity recognition (determining the identity of a named object, such as an individual, movie, organization, or location). NLP is used to process artifacts printed in NL so that NL can be comprehended and defined, as NL can be ambiguous and have vague declarations (Sagar and Abirami, 2014). The combination of NLP and IR on artifacts at the design level, such as class diagrams and ERD, provides high precision and recall values, respectively 85% and 90%. The precision and recall values given for artifacts at the requirement level are 72% and 70%. The number of cases used is 5 to 6 case examples.

The following studies used NLP and IR. Ibrahim and Ahmad (2010) and Hamza and Hammad (2019) used NLP and IR to obtain nouns, proper nouns, noun phrases, verbs, and adjectives from textual requirements used for MT. Sharma et al. (2015) used NLP and IR to obtain grammatical knowledge patterns from textual requirements used for MT. Elbendak et al. (2011) and Sarkar et al. (2012) used NLP and IR to obtain verbs, nouns, adverbs, and adjectives using case descriptions used for MT. Thakur and Gupta (2014) used NLP and IR to obtain proper nouns, noun phrases, and verbs from the use case specifications used for MT. A few studies (Sagar and Abirami, 2014; Ahmed et al., 2017; Jaiwai and Sammapun, 2017) used NLP and IR to obtain nouns, verbs, adjectives, adverbs, and typed dependencies from textual requirements used for MT. Abirami et al. (2015) used NLP and IR to tokenize textual requirements into a list of sentences, which were then clustered into NFRs and FRs. NLP was then used to obtain verbs, nouns, adverbs, adjectives, and typed dependencies from the clustered requirements used for MT. Tawosi et al. (2015) used NLP and IR to identify subjects, objects, verbs, and behavioral relations in every sentence. They extracted responsibilities and their relationships from textual requirements to build a problem domain semantic network graph, which generated class diagrams using an optimization technique. Wang et al. (2015) used NLP and IR to extract behavioral information from use cases to generate test cases. Lucassen et al. (2017) used NLP and IR to tokenize user stories into individual terms and identified their POS tags and relationships with other tokens. Stop words were then removed from the tokens, and each term was given a weight based on its frequency and was used for MT in the next phase. Elallaoui et al. (2018) used NLP and IR to obtain nouns, proper nouns, determiners, and verbs from user stories used for MT. Omar and Baryannis (2020) used NLP and IR to obtain noun phrases for candidate entities and typed dependencies for identifying relationships. WordNet was used to determine whether the candidate entities were valid. Relationships among entities were then identified through human intervention to create entity relationship diagrams.

5.3 Model transformation

MTs are abstractions of a system or its environment that allow developers and other stakeholders to address concerns about a system effectively. MTs operate on models and are intended to have the following applications: (1) producing lower-level models based on higher-level models; (2) mapping and synchronizing among models at the same or different levels; (3) creating query-based views of software; (4) model evolution tasks (e.g., model refactoring); and (5) reverse-engineering higher-level models based on lower-level models (Czarnecki and Helsen, 2006). MT is a technique for producing a novel model from a current method by applying a series of transformation guidelines. The transformation guidelines between artifacts can often be applied to other cases, for example, in Baker et al. (2005) and Pastor and Molina (2007).

Some studies (Ibrahim and Ahmad, 2010; Sharma et al., 2015) used MT to identify classes, attributes, and relationships from concepts of a domain ontology to generate class diagrams. Sanchez et al. (2010) used MT to select an architecture that satisfied the FRs and NFRs of aspect-oriented scenario models. Chen and Li (2010) used MT to transform use cases into test cases. Some other studies (Elbendak et al., 2011; Sarkar et al., 2012; Sagar and Abirami, 2014; Abirami et al., 2015; Ahmed et al., 2017; Jaiwai and Sammapun, 2017) used MT to identify classes, attributes, and relationships from the preprocessed textual requirements to generate class diagrams. Lv and Xie (2012) used MT to identify entities and their relationships from domain ontology concepts to generate entity relationship diagrams. Colombo et al. (2012) altered the parts of the requirements model into elements of the design model. Castro et al. (2012) used MT to map i* elements (actors and dependencies) to detailed elements of an architecture description language. Thakur and Gupta (2014) used MT to identify actors, boundaries, controls, messages, and message parameters from the preprocessed use case specifications to generate sequence diagrams. Khan and Mahmood (2016) used MT to transform use case maps into sequences using the components and responsibilities obtained from the use case maps. Kim et al. (2017) used MT to transform design patterns into class and sequence diagrams based on the mapping between the solution of the pattern and problem domains. The final result was a solution model that incorporated pattern properties. Lucassen et al. (2017) used MT to identify all user story patterns and filtered out entities and relationships to generate entity relationship diagrams. Elallaoui et al. (2018) used MT to identify actors, use cases, and their associations to generate use case diagrams. Hamza and Hammad (2019) used MT to identify actors, use cases, and their associations from preprocessed user stories to generate use case diagrams. The completeness of the results of the papers was 85% to 100% from the 5 to 6 cases tested. Wang et al. (2020) use MT to generate test cases from formal requirement model. The conclusive examination outcomes indicate that the test cases produced by this approach can efficiently detect a range of potential faults in the implementation of requirements. Yang et al. (2019) combined several artifacts into a prototype. This can also be a reference that by combining several artifacts, we can build a simple system.

5.4 Text mining

TM converts unstructured text into a formal format to find new ideas and concrete trends. TM is a technique for categorizing texts based on sentiment, subject, and purpose. In software engineering, this enables developers and stakeholders to make data-driven decisions. Using the functionality from the previous process, Masoud and Jalili (2014) used clustering to organize related responsibilities into the same clusters and different responsibilities into separate clusters. After clustering, the associations between the clusters were set to generate a class diagram. Abirami et al. (2015) used clustering to classify tokenized textual requirements into NFRs and FRs using a predefined set of classifier rules. The evaluation in these papers was carried out by measuring the F-score on three to four cases. F-Score shows a more significant level of suitability to the experts (Goutte and Gaussier, 2005). The resulting F-score range is between 87% to 96%.

5.5 Graph-based

Graphs are mathematical structures used to model relationships between objects in a collection. In this context, graphs are composed of nodes or vertices and edges, which link pairs of vertices. Tripathy and Mitra (2013) integrated sequence and activity diagrams to produce a system testing graph, which was later used with the graph optimization technique to obtain test cases. Masoud and Jalili (2014) first collected the responsibilities of the system and their dependencies from the requirements. A responsibilities dependency graph (RDG) was constructed, where the nodes represent the responsibilities and the edges represent the dependencies between the responsibilities. This graph was then used to extract features. A feature is a property of a responsibility that determines how it is connected to other responsibilities; the RDG assists in extracting clustered features.

Jena et al. (2015) used sequence diagrams to produce a sequence flowchart. The sequence flow chart was then transformed into a message control flow graph (MCFG), used to produce test cases using an optimization technique. Meiliana et al. (2017) integrated the sequence diagram and activity diagram to form a system testing graph traversed using a graph optimization technique to obtain test cases. Khari et al. (2018) and Sahoo and Ray (2020) used control flow graphs, which depict data flow or calculations during the execution of programs or applications. Control flow graphs are used to obtain test cases using an optimization technique. Sankar and Chandra (2020) used a state transition diagram or state graph. The vertices in a state graph represent states, and the graph's edges denote transitions. The state graph is used to obtain test cases using an optimization technique.

5.6 Ontology-based

Ontology-based approaches describe the common words and principles (meaning) used to characterize and represent a field of knowledge. An ontology can range in expressivity from a taxonomy (information in negligible order or a parent-child structure) to a thesaurus (words and synonyms) to a calculated information model (with more detailed information) to a coherent hypothesis (with luxurious, complicated, predictable, and significant information). An ontology contains a grouping of comments, principles, and realities. The entire metaphysical content is conveyed in its theories and realities, which give data about classes, properties, and people in the cosmology (Lv and Xie, 2012).

5.7 Optimization algorithm

Optimization algorithms find the optimal solution when presented with possible solutions. The algorithm observes whether the current state has achieved the optimal solution in each step. Optimization algorithms are applied in software engineering. Bowman et al. (2010) used a genetic algorithm (GA) to reallocate methods and attributes to classes in a class diagram and use class coupling and cohesion measurement to define fitness functions. Räihä et al. (2011) used a GA to obtain the software architecture of the targeted system. The primary input of the GA is a basic architecture representing the functional decomposition of the system decomposition, obtained as an enhancement of use cases. The fitness function is adjusted for chosen efficiency, modifiability, and simplicity weights. The success rate in industrial practice is 50% to 90%. Tripathy and Mitra (2013) used depth-first search (DFS) to traverse the system testing graph formed by integrating sequence and activity diagrams to obtain test cases. DFS maximizes test coverage while also reducing time and expense. Tawosi et al. (2015) used ant colony optimization (ACO) to identify classes and detect relationships among classes from a problem domain semantic network graph constructed in the previous phase to generate class diagrams. As a result, an increase in F-score was exhibited. Jena et al. (2015) used a GA to obtain the optimal test cases from the message control flow graph constructed in the previous phase. They were able to reduce the expense and time of testing. Meiliana et al. (2017) used DFS to traverse the system testing graph obtained by integrating activity and sequence diagrams to obtain test cases. Khari et al. (2018) used cuckoo search to obtain test cases from the control flow graph of the system under testing. Azam et al. (2018) used a GA to obtain test cases from feature diagrams based on field IDs and their corresponding values. Sahoo and Ray (2020) obtained test cases from the control flow graph of the system under testing. In terms of path coverage, the optimization produced better results. Sankar and Chandra (2020) used ACO to obtain test cases from state transition diagrams. ACO ensures better input testing accuracy. Algorithm optimization can also be combined with rule-based to adjust applicable rules (Sainani et al., 2020). Arrieta et al. (2020) used Adaptive Random Testing (ART) to generate functional requirements in the test case. They combined with another tool to generate the functional requirement automatically. Recently, MT can also be used for blockchain (Górski, 2021; Tran et al., 2022). It guarantees the uninterrupted deployment of containerized blockchain-distributed applications. Górski (2021) divides components into two parts, namely delivery and deployment. The delivery component provides UML modeling assistance for the deployment architectural perspective.

We introduced a hierarchical subdivision form of methods, techniques, and approaches to represent the entire requirements domain to design artifacts conformance domain. Figure 2 depicts the details of the hierarchy.

5.8 Tools

We found ten tools that can be used for ensuring the conformity of artifacts with requirements, namely ATCGT, Class-Gen, DAT, AutoSDG, UMTG, aToucan, ABCD, AR2DT, Visual Narrator, and SACMES. Tools that are introduced for the last ten years are described as follows.

The Auto Sequence Diagram Generator (AutoSDG) was developed by Thakur and Gupta (2014). AutoSDG uses NLP and transformation rules. The input of this tool is a sequence diagram. The output is a use case specification. Thakur and Gupta (2014) compared AutoSDG to aToucan (Yue et al., 2015). The results of the comparison of sequence diagram correctness are 91.2% (AutoSDG) and 69.5% (aToucan). The results of the comparison of sequence diagram completeness are 100% (AutoSDG) and 97.8% (aToucan).

The Use Case Modelling for System Tests Generation (UMTG) was developed by Wang et al. (2015). UMTG uses NLP and IR. The input is use case specifications obtained from another tool called Restricted Use Case Modeling (RUCM), which was developed by Yue et al. (2013). RUCM diminishes imprecision and inadequacy in use cases to gather behavioral information. The output of UMTG is test cases. Wang et al. (2015) has yet to further evaluate their proposed method.

The aToucan was developed by Yue et al. (2015). This tool enhances previous research (Yue et al., 2010, 2011a). The purpose of a toucan is to convert a use case model into class, sequence, and activity diagrams. aToucan can generate class diagrams with more than 90% consistency with class diagrams generated by experts; generate sequence diagrams with more than 91% consistency with sequence diagrams generated by experts; 100% complete and correct control flow information and 85% complete data flow information in an activity diagram.

The Automatic builder of the class diagram (ABCD) was developed by Ben Abdessalem Karaa et al. (2016). ABCD uses NLP and transformation rules to generate the resulting artifact. The tool takes as input functional requirements and outputs class diagrams. Compared to similar tools, ABCD can generate the following class diagram concepts: aggregations, associations, attributes, classes, compositions, generalizations, methods, and multiplicity. The resulting precision and recall values are 92.3% and 89%, respectively.

The Automated Requirements 2 Design Transformation Tool (AR2DT) was developed by Ahmed et al. (2017). AR2DT uses NLP, IR, and transformation rules. Textual requirements are taken as the input, and the output artifact is a class diagram. AR2DT was evaluated in the study by comparing precision and recall values against Class-Gen (Elbendak et al., 2011). AR2DT exhibited better performance compared to Class-Gen, in which AR2DT achieved a precision and recall value of 94.9% and 85.56%, respectively, whereas Class-Gen obtained a precision and recall value of 82.6% and 83.3%, respectively.

The Visual Narrator was developed by Lucassen et al. (2017). A Visual Narrator is an automated tool that identifies entities and their relationships from user stories and generates entity-relationship diagrams. This tool is suitable for agile development methods. It uses NLP and transformation rules. The average precision and recall values achieved in the evaluation were 95% and 91%, respectively.

SACMES was developed by Omar and Baryannis (2020). SACMES is a semi-automated tool to identify entities and their relationships from textual requirements to assist in creating entity-relationship diagrams. The input is a textual requirement, and the output is an entity relationship diagram. The basis of this tool is NLP. The evaluations were based on precision and recall values in identifying the diagram components by users with the assistance of SACMES. The precision and recall value for identifying unrecognized entities by users was 100% and 35%, respectively. The precision and recall value for identifying unrecognized relationships was 89% and 22%, respectively. The precision and recall value for the identification of entities was 64% and 40%, respectively. The precision and recall value for the identification of relationships was 22% and 11%, respectively. Finally, the precision and recall value for the identification of cardinalities was 18% and 11%, respectively.

6 Challenges in ensuring conformity of artifacts with requirements

In the reviewed studies, we identified five challenges related to the conformity between requirements representations and between requirements and other artifacts, namely many requirements; conflicting and ambiguous requirements; changes in requirements; matching requirements specifications and architectural design; and lack of domain awareness and expertise of application designers.

6.1 Many requirements

Software is increasing in size and complexity. Given the growing and various needs of stakeholders and customers, software is designed and developed to fulfill various purposes. This also causes the requirements of a given software to increase. Requirements are additionally scattered across different documents, each with many pages. Reading through these requirements documents takes a great deal of time and effort. Moreover, the human examiner may commit errors while perusing countless pages. This may lead to an incorrect analysis model, resulting in artifacts not complying with the requirements (Kitchenham et al., 2010).

6.2 Conflicting and ambiguous requirements

Software requirements are written in NL. NL specifications are simple to comprehend since the requirements interact with others in the same language. On the other hand, NL also has various defects (Abirami et al., 2015). Noise, silence, over-specification, contradiction, forward comparison, wishful thinking, and uncertainty are the underlying issues (Hamza and Hammad, 2019). Furthermore, if many stakeholders with different backgrounds and ways of thinking are involved in the project, the presence of requirements that contradict each other is inevitable. Ambiguous and conflicting requirements make it hard to create and design artifacts that conform with requirements.

6.3 Changes in requirements

Analysts work with project partners in the requirements gathering and analysis stage to collect the project specifications. Unstated or implied requirements of which customers believe the analyst is aware of are significant. Issues like this in the requirements process lead to conflicts between architecture and business teams. The requirements analysis process is inherently arbitrary and reliant on personal views (Sagar and Abirami, 2014). The incompetence of stakeholders in decision-making and complete domain knowledge may cause a stakeholder to request a change in requirements during the development and implementation phase. Changes to requirements made in the latter stages of the development will be more costly than those made during the requirements development process. Failures and improvements cause changes in demands. They require additional effort that was not planned upfront and significantly affect the progress of a project. Changes in requirements cause difficulties in maintaining the conformity between requirements and other artifacts.

6.4 Matching requirements specifications and architectural design

The relationship between requirements specifications and design has been extensively researched to bridge the distance between the two (Colombo et al., 2012). Requirements are often articulated informally in NL, while architecture is represented semi-formally. NFRs are impossible to define in architecture models since they are software properties. Some requirements can only be recognized after the architectural plan has been modeled (Castro et al., 2012). Characteristics of the system are not listed in user specifications. Only the characteristics of the user observable problem domain that are influenced by the machine's behavior are expressed in user specifications. The system's actions must be described so that its relationship with the environment meets the requirements of users (Colombo et al., 2012).

Moreover, various architectural elements may address a simple requirement. A single architectural feature can be connected to many specifications in non-trivial ways. It is not easy to define and optimize the architecturally relevant details found in specifications, such as NFRs, for designing this type of structure (Castro et al., 2012). This makes it challenging to generate architectural artifacts that conform with requirements.

6.5 Lack of domain awareness and expertise among app designers

Converting NL specifications into domain models (DMs) requires thoroughly examining NL text. Programmers often make mistakes in this regard (Omar and Baryannis, 2020). Creating a correct and complete DM depends on the experience of the developers and may require additional knowledge in specific problem domains. For example, if the software is to be used in banks, financial knowledge may be required (Jaiwai and Sammapun, 2017). The manual, process-centric approach to designing high-level architecture from specifications is continually used, with a firm reliance on experts (Sarkar et al., 2012). The development of a DM can be challenging due to a lack of domain awareness, expertise, and formal preparation. The complex relationships between the concepts of DMs can be difficult for both novice and experienced designers to recognize in NL text, particularly for novice designers (Omar and Baryannis, 2020).

7 Discussion

We found that ensuring the conformity of artifacts with requirements has applications in traceability and software V&V. Both topics are important to software engineering. Traceability makes it easier to frequently check whether the software is consistent at every level, while software V&V ensures that the software conforms to the given specifications. Traceability and software V&V are at the heart of ensuring software quality. This indicates the importance of ensuring the conformity of artifacts with requirements. Another application of ensuring the conformity of artifacts with requirements is software reuse. Software reuse is gaining more attention among software developers to decrease the cost and time of software development, improve software quality, and control existing efforts by creating and applying multi-use assets such as patterns, architectures, frameworks, and components. However, software reuse is no easy feat. Further research on the conformity of artifacts with requirements can help achieve software reusability.

There are several approaches and methods for ensuring the conformity of artifacts with requirements. The most commonly used is MT. MDE is a software development approach that emphasizes rendering models as the primary development artifact and refining them by automated transformations before a working system is obtained. In doing so, MDE aspires to a higher degree of abstraction in system creation, resulting in a deeper understanding of complex systems. MT is a process within MDE that is performed according to a set of rules. Thus, our findings suggest that MDE is still one of the most preferred software engineering approaches.

All studies that used textual requirements as the input to their method used NLP and IR. Since textual requirements are written in NL, it is mandatory to use NLP and IR to process these requirements before using them as input to any technique. NLP and IR are closely related because NLP is needed to support IR. A graph is used to model relationships among the objects in a collection. Many artifacts within the software domain are used to present relationships between objects. Therefore, graphs can generate artifacts through either manual observation or an optimization algorithm to identify the ideal arrangement. Optimization algorithms have been used for various applications within the computer science realm due to their heuristic nature. Optimization algorithms can find the optimal solution from a set of possible solutions using a function determining the algorithm's goal. This area can be further researched to propose new tools that can automatically generate different artifacts at different software development levels.

We identified that the most commonly generated artifacts within the proposed methods are the class diagram, followed by test cases, software architecture, entity relationship diagrams (three studies), case diagrams, and sequence diagrams. This is consistent with the findings in Bozyigit et al. (2021), in which they identified class diagrams as the most generated artifact in a set of studies focused on transforming requirements into conceptual models. Their study also stated that identifying attributes, operations, and classes in the SDLC and their relationships is paramount. Unsurprisingly, the test case was secondary to the class diagram, as manually creating a test case can be a time-consuming task in which humans can be fallible. Software architecture, an integral component of a software or system, was only generated in four studies. While examining the selected papers, we found that research on software architecture was mainly focused on architecture reuse rather than generation. We did not include any studies that generated code because all possible solutions were restricted to producing only partial code, as described in the problem statement (Mehmood and Jawawi, 2013). Our scope was limited to the generation of completed artifacts.

We recognized five challenges in ensuring conformity of artifacts with requirements. The challenges include many requirements, conflicting and ambiguous requirements, changes in requirements, matching requirements specifications and architectural design, and lack of knowledge and experience among software designers. Automation can solve the issue of many requirements, which can be overwhelming for human analysts. The methods in the studies sought to process these requirements and automatically generate artifacts that comply with requirements. As for conflicting and ambiguous requirements, this challenge is solved using NLP and IR to process the textual requirements to create structure and eliminate ambiguity. Regarding the challenge of changes in requirements, the methods sought to process the requirements and present them more understandably in the early phases of the SDLC. As a result, stakeholders can fully understand the requirements, decreasing the chance of changes in requirements in the latter stages of the SDLC. The proposed methods also process the NFRs to specify the architectural properties of the software for the purpose of solving the challenge of matching requirements specifications and architectural design. With the use of NLP and IR in combination with TM, information on the problem domain can be extracted and used to assist designers who lack knowledge and experience.

Based on a thorough investigation of the primary studies, we present some of the research agenda highlights that we envisage of the software requirements conformity. We divided them into three periods.

2010–2013: Research emerges automatically from the artifacts requirement into other artifacts such as activity diagrams and class diagrams. Commonly used requirements artifacts as the starting point of the methods are use case diagrams and textual requirements. The proposed tools are aToucan, ATCGT (Automated Test Case Generation Tool), RACE (Requirements Analysis and Class Diagram Extraction), DAT (Design Assistant Tool), and so on. Graph-based approaches have also been started with simple algorithms such as DFS. Researchers are still focused on the life or generation from one artifact to another in this period.

2014–2020: The next period begins with the emergence of aToucan, which has been developed from the next period. Researchers widely use this tool to create metadata that has artifact requirements. Furthermore, the researchers optimized to increase the completeness and correctness of the resulting models. Optimization is carried out in combination with NLP. A semantic approach can further improve the conformity of artifacts with requirements. The artifact output in this period is a test case. Test cases can be used as a tool to measure the fulfillment of requirements.

2021–2030: Research continues optimizing existing approaches. Optimizations are carried out by combining algorithms, such as graph-based Ant Colony and graph-based Particle Swarm Optimization. Both algorithms are graph-based. In this period, NLP is still applied using similar tools in the previous period. The emergence of deep learning will also encourage using various deep learning architectures, such as BiLSTM, GRU, BERT, and RNN, to translate and transform software requirements into different forms or different artifacts. Research on requirements classification (Kici et al., 2021; Rahimi et al., 2021) has encountered this trend. We believe that the same trend will also occur in other issues related to requirements engineering, including the conformity of artifacts with requirements.

The findings of this study have several implications for both scholars and practitioners. More observational studies are required for analysis that focuses on a wider variety of artifacts other than class diagrams. With the variety of SDLC artifacts, it is difficult to ensure the conformity of these artifacts with requirements. The use of advanced methods and tools makes the process easier. Therefore, there is a need to broaden the focus and scope of the generated artifacts. For example, creating tools to automatically generate software architecture.

Furthermore, current approaches and tools for ensuring artifacts' conformity with requirements are applied and tested to indicate their usefulness and efficiency. With the growth of software dimensions and complexity, it remains difficult to determine whether the proposed methods and tools are applicable and efficient when integrated into the development and implementation of software in the real world. Therefore, there is a need to use these methods and tools in a large-scale project to examine its performance in terms of the accuracy of the resulting artifacts and the time it takes to generate them.

The bias in study selection and the possibility of imprecision in data retrieval from variable sources are the underlying shortcomings of any systematic analysis. When designing our analytical approach, we took the following steps to eradicate bias and ensure precision and sample collection consistency. First, we approached the method of creating search strings as a learning process involving creativity. Following our research concerns, we created search terms for a systematic search of electronic databases. Since search strings in software engineering are language-dependent, critical studies will likely be missed in each search. To avoid bias in study selection due to personal interests or missing information, we used a multi-stage method. Two scholars assessed the studies for validity based on the inclusion and exclusion principles (Keele, 2007). We discovered that the amount of specificity that characterized the analysis process differed significantly across experiments. Some research, for example, involved a more thorough assessment of threats to validity than others. Because of the inconsistencies between reports, the data extraction procedure likely involved inaccuracies.

We also searched for papers on only four electronic databases: ACM Digital Library, IEEE Xplore, SpringerLink, and ScienceDirect. We eliminated papers outside of these databases. This risks missing papers relevant to the topic that were published outside of the four electronic databases we searched. In our selection of studies, we also did not include research that generated partial artifacts. This could lead to ignoring research whose proposed methods or tools are still relevant to the topic. As current automatic code generation methods result only in partial code, this exclusion of generated partial artifacts leads to excluding studies that generated source code.

8 Conclusion

This paper examines the literature on the conformity of artifacts with requirements, including procedures and problems. This study was performed by searching and categorizing all current and accessible literature on the conformity of artifacts with requirements using available standards for conducting comprehensive literature reviews. The study selection strategy retrieved thirty-two related papers from 370 initial results from well-reputable electronic research databases.

The applications of ensuring the conformity of artifacts with requirements, namely traceability, software reuse, and software V&V, may help organizations in the industry to decrease costs, time, and effort in software development. We discovered that many studies focused on UML diagrams, specifically class diagrams. We addressed the benefits of adopting methods and tools for ensuring the conformity of artifacts with requirements, which can motivate practitioners to design and create artifacts. Using methods and tools for ensuring the conformity of artifacts with requirements can help mitigate the disastrous effects of improper software development caused by challenges that are frequently encountered in real-world settings, such as large numbers of requirements, conflicting and ambiguous requirements, changes in requirements, matching requirements specifications to architectural design, and lack of knowledge and experience among software designers. There are only a few advanced tools for ensuring the conformity of artifacts with requirements, and the purpose of these tools has only been identified as generating class diagrams and entity relationship diagrams. There is a need to focus more heavily on developing tools for generating various artifacts, testing these tools in real-world scenarios, and ultimately implementing them in more robust software development projects.

Author contributions

DS: Conceptualization, Data curation, Formal analysis, Funding acquisition, Methodology, Writing – review & editing. RF: Data curation, Investigation, Project administration, Resources, Software, Supervision, Writing – original draft. AW: Data curation, Investigation, Software, Validation, Writing – original draft. DF: Data curation, Investigation, Software, Validation, Writing – original draft. RP: Data curation, Investigation, Visualization, Writing – original draft. YD: Data curation, Validation, Visualization, Writing – original draft. G: Data curation, Validation, Visualization, Writing – original draft, Writing – review & editing. RP: Data curation, Validation, Visualization, Writing – original draft, Writing – review & editing.

Funding

The author(s) declare financial support was received for the research, authorship, and/or publication of this article. This research was funded by the Ministry of Education, Culture, Research, and Technology under the Doctoral Dissertation Research program grant with the number 084/E5/PG.02.00.PT/2022; April 28, 2022.

Acknowledgments

This research is a collaboration amongst Institut Teknologi Sepuluh Nopember, Politeknik Negeri Banjarmasin, and Institut Bisnis dan Teknologi Pelita Indonesia.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abbas, M., Rioboo, R., Ben-Yelles, C. B., and Snook, C. F. (2021). Formal modeling and verification of UML activity diagrams (UAD) with FoCaLiZe. *J. Syst. Architec* 114, 101911. doi: 10.1016/j.sysarc.2020.101911

Abirami, S., Shankari, G., Akshaya, S., and Sithika, M. (2015). "Conceptual modeling of non-functional requirements from natural language text," in *Computational Intelligence in Data Mining-Volume 3* (Springer India), 1–11. doi: 10.1007/978-81-322-2202-6_1

Ahmed, M. A., Butt, W. H., Ahsan, I., Anwar, M. W., Latif, M., and Azam, F. (2017). "A novel natural language processing (NLP) approach to automatically generate conceptual class model from initial software requirements," in *International Conference on Information Science and Applications* (Springer Singapore), 476–484. doi: 10.1007/978-981-10-4154-9_55

Arrieta, A., Agirre, J. A., and Sagardui, G. (2020). "A tool for the automatic generation of test cases and oracles for simulation models based on functional requirements," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* (IEEE), 1–5. doi: 10.1109/ICSTW50294.2020.00018

Azam, M., ur Rahman, A., Sultan, K., Dash, S., Khan, S. N., and Khan, M. A. A. (2018). "Automated testcase generation and prioritization using GA and FRBS," in *International Conference on Advanced Informatics for*

Computing Research (Springer Singapore), 571–584. doi: 10.1007/978-981-13-3140-4_52

Baker, P., Loh, S., and Weil, F. (2005). "Model-driven engineering in a large industrial context-motorola case study," in *International Conference on Model Driven Engineering Languages and Systems* (Springer), 476–491. doi: 10.1007/11557432_36

Ben Abdessalem Karaa, W., Ben Azzouz, Z., Singh, A., Dey, N., S., Ashour, A., et al. (2016). Automatic builder of class diagram (abcd): an application of uml generation from functional requirements. *Software* 46, 1443–1458. doi: 10.1002/spe.2384

Borg, M., Runeson, P., and Ardö, A. (2014). Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empir. Softw. Eng.* 19, 1565–1616. doi: 10.1007/s10664-013-9255-y

Bowman, M., Briand, L. C., and Labiche, Y. (2010). Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms. *IEEE Trans. Softw. Eng.* 36, 817–837. doi: 10.1109/TSE.2010.70

Bozyiğit, F., Akta s, Ö., and Kilinç, D. (2021). Linking software requirements and conceptual models: A systematic literature review. *Eng. Sci. Technol. Int. J.* 24, 71–82. doi: 10.1016/j.jestch.2020.11.006

Castro, J., Lucena, M., Silva, C., Alencar, F., Santos, E., and Pimentel, J. (2012). Changing attitudes towards the generation of architectural models. *J. Syst. Softw.* 85, 463–479. doi: 10.1016/j.jss.2011.05.047

Chen, L., and Li, Q. (2010). "Automated test case generation from use case : a model based approach," in *2010 3rd International Conference on Computer Science and Information Technology* (IEEE), 372–377.

Cleland-Huang, J., Gotel, O., and Zisman, A. (2012). *Software and Systems Traceability*. London: Springer. doi: 10.1007/978-1-4471-2239-5

Colombo, P., Khendek, F., and Lavazza, L. (2012). Bridging the gap between requirements and design: an approach based on Problem Frames and SysML. *J. Syst. Softw.* 85, 717–745. doi: 10.1016/j.jss.2011.09.046

Czarnecki, K., and Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Syst. J.* 45, 621–645. doi: 10.1147/sj.453.0621

Dabhade, M., Suryawanshi, S., and Manjula, R. (2016). "A systematic review of software reuse using domain engineering paradigms," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)* (IEEE), 1–6. doi: 10.1109/GET.2016.7916646

Dalpiaz, F., Gieske, P., and Sturm, A. (2021). On deriving conceptual models from user requirements: an empirical study. *Inf. Softw. Technol.* 131, 1–13. doi: 10.1016/j.infsof.2020.106484

Deeptimahanti, D. K., and Sanyal, R. (2011). "Semi-automatic generation of uml models from natural language requirements," in *Proceedings of the 4th India Software Engineering Conference* 165–174. doi: 10.1145/1953355.1953378

Dinesh, P., Sujitha, V., Salma, C., and Srijayapriya, B. (2021). "A review on natural language processing: Back to basics," in *Innovative Data Communication Technologies and Application* (Springer), 655–661. doi: 10.1007/978-981-15-9651-3_54

Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into UML use case diagrams using NLP techniques. *Proc. Comput. Sci.* 130, 42–49. doi: 10.1016/j.procs.2018.04.010

Elbendak, M., Vickers, P., and Rossiter, N. (2011). Parsed use case descriptions as a basis for object-oriented class model generation. *J. Syst. Softw.* 84, 1209–1223. doi: 10.1016/j.jss.2011.02.025

Fauzan, R., Siahaan, D., Rochimah, S., and Triandini, E. (2018). "Activity diagram similarity measurement: a different approach," in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)* (IEEE), 601–605. doi: 10.1109/ISRITI.2018.8864284

Ferrari, A., Spoletini, P., Donati, B., Zowghi, D., and Gnesi, S. (2017). "Interview review: detecting latent ambiguities to improve the requirements elicitation process," in *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference* (Lisbon, Portugal: IEEE), 400–405. doi: 10.1109/RE.2017.15

Ghazi, P., and Glinz, M. (2017). Challenges of working with artifacts in requirements engineering and software engineering. *Requir. Eng.* 22, 359–385. doi: 10.1007/s00766-017-0272-z

Górski, T. (2021). Towards continuous deployment for blockchain. *Appl. Sci.* 11, 11745. doi: 10.3390/app112411745

Goutte, C., and Gaussier, E. (2005). "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European Conference on Information Retrieval* (Springer), 345–359. doi: 10.1007/978-3-540-31865-1_25

Haidrar, S., Bencharqui, H., Anwar, A., Bruel, J. M., and Roudies, O. (2017). "REQDL: a requirements description language to support requirements traces generation," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)* (IEEE), 26–35. doi: 10.1109/REW.2017.72

Hamza, Z. A., and Hammad, M. (2019). "Generating UML use case models from software requirements using natural language processing," in *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)* (IEEE), 1–6. doi: 10.1109/ICMSAO.2019.8880431

Haris, M. S., and Kurniawan, T. A. (2020). "Automated requirement sentences extraction from software requirement specification document," in *Proceedings of the 5th International Conference on Sustainable Information Engineering and Technology* 142–147. doi: 10.1145/3427423.3427450

Harmain, H. M., and Gaizauskas, R. (2000). "Cm-builder: an automated nl-based case tool," in *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering* (IEEE), 45–53. doi: 10.1109/ASE.2000.873649

Harman, D. (2019). Information retrieval: the early years. *Found. Trends Inf. Retr.* 13, 425–577. doi: 10.1561/1500000065

Hsueh, N.-L., Shen, W.-H., Yang, Z.-W., and Yang, D.-L. (2008). Applying uml and software simulation for process definition, verification, and validation. *Inf. Softw. Technol.* 50, 897–911. doi: 10.1016/j.infsof.2007.10.015

Ibrahim, M., and Ahmad, R. (2010). "Class diagram extraction from textual requirements using natural language processing (NLP) techniques," in *2010 Second International Conference on Computer Research and Development* (IEEE), 200–204. doi: 10.1109/ICCRD.2010.71

Irshad, M., Petersen, K., and Poulding, S. (2018). A systematic literature review of software requirements reuse approaches. *Inf. Softw. Technol.* 93, 223–245. doi: 10.1016/j.infsof.2017.09.009

Jaiwai, M., and Sammapun, U. (2017). "Extracting UML class diagrams from software requirements in Thai using NLP," in *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)* 1–5. doi: 10.1109/JCSSE.2017.8025938

Jena, A. K., Swain, S. K., and Mohapatra, D. P. (2015). "Test case creation from UML sequence diagram: a soft computing approach," in *Intelligent Computing, Communication and Devices* (Springer India), 117–126. doi: 10.1007/978-81-322-2012-1_13

Junior, V. A., d,. S., and Vijaykumar, N. L. (2012). Generating model-based test cases from natural language requirements for space application software. *Softw. Qual. J.* 20, 77–143. doi: 10.1007/s11219-011-9155-6

Keele, S. (2007). "Guidelines for performing systematic literature reviews in software engineering," in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*, 20–39.

Khan, Y. A., and Mahmood, S. (2016). Generating UML sequence diagrams from use case maps: a model transformation approach. *Arabian J. Sci. Eng.* 41, 965–986. doi: 10.1007/s13369-015-1926-0

Khari, M., Kumar, P., Burgos, D., and Crespo, R. G. (2018). Optimized test suites for automated testing using different optimization techniques. *Soft Comput.* 22, 8341–8352. doi: 10.1007/s00500-017-2780-7

Kici, D., Malik, G., Cevik, M., Parikh, D., and Basar, A. (2021). "A BERT-based transfer learning approach to text classification on software requirements specifications," in *Proceedings of the Canadian Conference on Artificial Intelligence* (Vancouver. PubPub), 1–13. doi: 10.21428/594757db.a4880a62

Kim, D. K., Lu, L., and Lee, B. (2017). Design pattern-based model transformation supported by QVT. *J. Syst. Softw.* 125, 289–308. doi: 10.1016/j.jss.2016.12.019

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., et al. (2010). Systematic literature reviews in software engineering-A tertiary study. *Inf. Softw. Technol.* 52, 792–805. doi: 10.1016/j.infsof.2010.03.006

Landhäußer, M., Körner, S. J., and Tichy, W. F. (2014). From requirements to UML models and back: how automatic processing of text can support requirements engineering. *Softw. Qual. J.* 22, 121–149. doi: 10.1007/s11219-013-9210-6

Liskin, O. (2015). "How artifacts support and impede requirements communication," in *International Working Conference on Requirements Engineering: Foundation for Software Quality* (Cham: Springer), 132–147. doi: 10.1007/978-3-319-16101-3_9

Lucassen, G., Robeer, M., Dalpiaz, F., Werf, J. M. E. M., and Brinkkemper, S. (2017). Extracting conceptual models from user stories with Visual Narrator. *Requir. Eng.* 22, 339–358. doi: 10.1007/s00766-017-0270-1

Lv, Y., and Xie, C. (2012). "An ontology-based approach to build conceptual data model," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery* (IEEE), 807–810. doi: 10.1109/FSKD.2012.6234141

Masoud, H., and Jalili, S. (2014). A clustering-based model for class responsibility assignment problem in object-oriented analysis. *J. Syst. Softw.* 93, 110–131. doi: 10.1016/j.jss.2014.02.053

Mehmood, A., and Jawawi, D. N. (2013). Aspect-oriented model-driven code generation: A systematic mapping study. *Inf. Softw. Technol.* 55, 395–411. doi: 10.1016/j.infsof.2012.09.003

Meiliana, S. I., Alianto, R. S., Daniel, and Gaol, F. L. (2017). Automated test case generation from uml activity diagram and sequence diagram using depth first search algorithm. *Proc. Comput. Sci.* 116, 629–637. doi: 10.1016/j.procs.2017.10.029

Mili, A., Mili, R., and Mittermeir, R. T. (1998). A survey of software reuse libraries. *Ann. Softw. Eng.* 5, 349–414. doi: 10.1023/A:1018964121953

Mu noz-Fernández, J. C., Knauss, A., Casta neda, L., Derakhshanmanesh, M., Heinrich, R., Becker, M., et al. (2017). "Capturing ambiguity in artifacts to support requirements engineering for self-adaptive systems," in *CEUR Workshop Proceedings* 1–6.

Nadkarni, P. M., Ohno-Machado, L., and Chapman, W. W. (2011). Natural language processing: an introduction. *J. Am. Med. Inf. Assoc.* 18, 544–551. doi: 10.1136/amiajnl-2011-000464

Noll, J., Razzak, M. A., and Beecham, S. (2017). "Motivation and autonomy in global software development," in *International Workshop on Global Sourcing of Information Technology and Business Processes* (Cham: Springer), 19–38. doi: 10.1007/978-3-319-70305-3_2

Omar, M., and Baryannis, G. (2020). Semi-automated development of conceptual models from natural language text. *Data Knowl. Eng.* 127, 101796. doi: 10.1016/j.datak.2020.101796

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., et al. (2021). The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* 372, 1–9. doi: 10.1136/bmj.n71

Pastor, O., and Molina, J. C. (2007). *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling.* Berlin: Springer Science Business Media.

Pérez-Álvarez, J. M., and Mos, A. (2020). "From abstract specifications to application generation," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society* 11–20. doi: 10.1145/3377815.3381381

Rahimi, N., Eassa, F., and Elrefaei, L. (2021). One- and two-phase software requirement classification using ensemble deep learning. *Entropy* 23, 1–30. doi: 10.3390/e23101264

Räihä, O., Kundi, H., Koskimies, K., Mäkinen, E., and Outi, R. (2011). "Synthesizing architecture from requirements: a genetic approach," in *Relating Software Requirements and Architectures* (Berlin: Springer), 307–331. doi: 10.1007/978-3-642-21001-3_18

Robles, K., Fraga, A., Morato, J., and Llorens, J. (2012). Towards an ontology-based retrieval of uml class diagrams. *Inf. Softw. Technol.* 54, 72–86. doi: 10.1016/j.infsof.2011.07.003

Sagar, V. B. R. V., and Abirami, S. (2014). Conceptual modeling of natural language functional requirements. *J. Syst. Softw.* 88, 25–41. doi: 10.1016/j.jss.2013.08.036

Sahoo, R. R., and Ray, M. (2020). PSO based test case generation for critical path using improved combined fitness function. *Comput. Inf. Sci.* 32, 479–490. doi: 10.1016/j.jksuci.2019.09.010

Sainani, A., Anish, P. R., Joshi, V., and Ghaisas, S. (2020). "Extracting and classifying requirements from software engineering contracts," in *2020 IEEE 28th International Requirements Engineering Conference (RE)* (IEEE), 147–157. doi: 10.1109/RE48521.2020.00026

Sanchez, P., Moreira, A., Fuentes, L., Araujo, J., and Magno, J. (2010). Model-driven development for early aspects. *Inf. Softw. Technol.* 52, 249–273. doi: 10.1016/j.infsof.2009.09.001

Sankar, S., and Chandra, V. (2020). "An ant colony optimization algorithm based automated generation of software test cases," in *International Conference on Swarm Intelligence* (Cham: Springer), 231–239. doi: 10.1007/978-3-030-53956-6_21

Sarkar, S., Sharma, V. S., and Agarwal, R. (2012). "Creating design from requirements and use cases: Bridging the gap between requirement and detailed design," in *Proceedings of the 5th India Software Engineering Conference* 3–12. doi: 10.1145/2134254.2134256

Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to Information Retrieval, volume 39.* Cambridge: Cambridge University Press.

Sharma, R., Srivastava, P. K., and Biswas, K. K. (2015). "From natural language requirements to UML class diagrams," in *2015 IEEE Second International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)* (IEEE), 1–8. doi: 10.1109/AIRE.2015.7337625

Sommerville, I. (2015). *Software Engineering (10th edition).* Noida, India: Pearson.

Souza, E., Moreira, A., and Goulão, M. (2019). Deriving architectural models from requirements specifications: a systematic mapping study. *Inf. Softw. Technol.* 109, 26–39. doi: 10.1016/j.infsof.2019.01.004

Sunitha, E. V., and Samuel, P. (2018). Object constraint language for code generation from activity models. *Inf. Softw. Technol.* 103, 92–111. doi: 10.1016/j.infsof.2018.06.010

Tawosi, V., Jalili, S., and Hasheminejad, S. M. H. (2015). Automated software design using ant colony optimization with semantic network support. *J. Syst. Softw.* 109, 1–17. doi: 10.1016/j.jss.2015.06.067

Thakur, J. S., and Gupta, A. (2014). "Automatic generation of sequence diagram from use case specification," in *Proceedings of the 7th India Software Engineering Conference* 1–6. doi: 10.1145/2590748.2590768

Tran, N. K., Babar, M. A., and Walters, A. (2022). A framework for automating deployment and evaluation of blockchain networks. *J. Netw. Comput. Applic.* 206, 103460. doi: 10.1016/j.jnca.2022.103460

Tripathy, A., and Mitra, A. (2013). "Test case generation using activity diagram and sequence diagram," in *Proceedings of International Conference on Advances in Computing* (Springer India), 121–129. doi: 10.1007/978-81-322-0740-5_16

Tufail, H., Masood, M. F., Zeb, B., Azam, F., and Anwar, M. W. (2017). "A systematic review of requirement traceability techniques and tools," in *2017 2nd International Conference on System Reliability and Safety (ICSRS)* (IEEE), 450–454. doi: 10.1109/ICSRS.2017.8272863

Vale, T., de Almeida, E. S., Alves, V., Kulesza, U., Niu, N., and de Lima, R. (2017). Software product lines traceability: a systematic mapping study. *Inf. Softw. Technol.* 84, 1–18. doi: 10.1016/j.infsof.2016.12.004

Wang, C., Pastore, F., Goknil, A., Briand, L., and Iqbal, Z. (2015). "Automatic generation of system test cases from use case specifications," in *2015 International Symposium on Software Testing and Analysis, ISSTA 2015 - Proceedings* 385–396. doi: 10.1145/2771783.2771812

Wang, W., Hu, J., Hu, J., Kang, J., Wang, H., and Gao, Z. (2020). "Automatic test case generation from formal requirement model for avionics software," in *2020 6th International Symposium on System and Software Reliability (ISSSR)* (IEEE), 12–20. doi: 10.1109/ISSSR51244.2020.00011

Wohlin, C. (2014). "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* 1–10. doi: 10.1145/2601248.2601268

Yang, Y., Li, X., Ke, W., and Liu, Z. (2019). "Automated prototype generation from formal requirements model. *IEEE Trans. Reliabil.* 69, 632–656. doi: 10.1109/TR.2019.2934348

Yue, T., Ali, S., and Briand, L. (2011a). "Automated transition from use cases to uml state machines to support state-based testing," in *European Conference on Modelling Foundations and Applications* (Springer), 115–131. doi: 10.1007/978-3-642-21470-7_9

Yue, T., Briand, L. C., and Labiche, Y. (2010). "An automated approach to transform use cases into activity diagrams," in *European Conference on Modelling Foundations and Applications* (Springer), 337–353. doi: 10.1007/978-3-642-13595-8_26

Yue, T., Briand, L. C., and Labiche, Y. (2011b). A systematic review of transformation approaches between user requirements and analysis models. *Requir. Eng.* 16, 75–99. doi: 10.1007/s00766-010-0111-y

Yue, T., Briand, L. C., and Labiche, Y. (2013). Facilitating the transition from use case models to analysis models: approach and experiments. *ACM Trans. Softw. Eng. Methodol.* 22, 1–38. doi: 10.1145/2430536.2430539

Yue, T., Briand, L. C., and Labiche, Y. (2015). atoucan: an automated framework to derive uml analysis models from use case models. *ACM Trans. Softw. Eng. Methodol.* 24, 1–52. doi: 10.1145/2699697