Article

Not peer-reviewed version

Requirements-Driven Automated Software Testing: A Systematic Review

Fanyu Wang , Chetan Arora [*] , Chakkrit Tantithamthavorn , Kaicheng Huang , Aldeida Aleti

Posted Date: 10 February 2025

doi: 10.20944/preprints202502.0628.v1

Keywords: Software Engineering; Requirements Engineering; Software Testing; Automated Test Generation; Systematic Literature Review

Article

Requirements-Driven Automated Software Testing: A Systematic Review

Fanyu Wang, Chetan Arora *, Chakkrit Tantithamthavorn, Kaicheng Huang and Aldeida Aleti

Faculty of Information Technology, Monash University, Clayton, Victoria, Australia
* Correspondence: chetan.arora@monash.edu

Abstract: Automated software testing has the potential to enhance efficiency and reliability in software development, yet its adoption remains hindered by challenges in aligning test generation with software requirements. **RE**quirements-**D**riven **A**utomated **S**oftware **T**esting (REDAST) aims to bridge this gap by leveraging requirements as the foundation for automated test artifact generation. This systematic literature review (SLR) explores the landscape of REDAST by analyzing requirements input, transformation techniques, test outcomes, evaluation methods, and existing limitations. We conducted a comprehensive review of 156 papers selected from six major research databases. Our findings reveal the predominant types, formats, and notations used for requirements in REDAST, the automation techniques employed for generating test artifacts from requirements, and the abstraction levels of resulting test cases. Furthermore, we evaluate the effectiveness of various testing frameworks and identify key challenges such as scalability, automation gaps, and dependency on input quality. This study synthesizes the current state of REDAST research, highlights trends, and proposes future directions, serving as a reference for researchers and practitioners aiming to advance automated software testing.

Keywords: software engineering; requirements engineering; software testing; automated test generation; systematic literature review

1. Introduction

Software testing plays a critical role in assuring the quality, reliability, and performance of software systems [1,2]. At its core, software testing seeks to identify and address defects, ensure that a system functions as intended, meets stakeholder requirements, and mitigate risks before deployment [3]. Central to achieving these objectives is the alignment of testing activities with the software's requirements, as requirements define the system's intended behaviour and scope. Effective testing demands comprehensive test artifacts that trace back to these requirements, including test cases, test plans, and test scenarios. As software systems grow more complex, there is an increasing need for automated approaches to bridge the gap between requirements engineering (RE) and software testing [4]. This increasing complexity of software systems has amplified the importance of *requirements-driven testing*—a paradigm that directly links testing activities to the software requirements.

In the requirements-driven automated software testing paradigm, the test generation process is mainly motivated by requirements, where the requirements are further interpreted, transformed, and implemented to test artifacts with the help of some additional supporting documents or tools [5,6]. Traditionally, this process has been predominantly manual, relying on human effort to interpret requirements and design corresponding tests [7,8]. However, manual approaches are time-consuming, prone to human error, and struggle to scale with the increasing size and complexity of modern software systems. To address these limitations, several research papers have explored automated methods for generating test artifacts directly from requirements. These methods leverage model representations, natural language processing (NLP), and artificial intelligence (AI) to translate requirements into actionable testing artifacts. However, a systematic understanding of the approaches, their challenges, and potential opportunities in the area of requirements-driven testing remains limited.

In this paper, we study the **RE**quirements-**D**riven **A**utomated **S**oftware **T**esting (REDAST) landscape. Figure 1 shows an overview of the REDAST process. The generation process consists of several components, including requirements as the necessary input, additional documents as the potential input, intermediate expressions as the potential step, transformation techniques as the main applied methodologies, and generated test artifacts as the final outcome. We cover the REDAST research landscape by (1) summarizing and reporting the statistics of valuable research works of automated software test generation, (2) comparing the evaluation methods for the current studies, and (3) identifying and analyzing current limitations and future opportunities of automated test generation technology in the context of the current era. We not only focus on the previous research under the traditional software engineering perspective but also introduce a new view from advance technologies to discuss the prospects and possibilities for REDAST studies. We followed the empirical SLR guidelines of Kitchenham et al. [9] in performing our systematic review to answer the following research questions (RQs) on selected 156 papers P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P15, P16, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P45, P46, P47, P48, P49, P50, P51, P52, P53, P54, P55, P56, P57, P58, P59, P60, P61, P62, P63, P64, P65, P66, P67, P68, P69, P70, P71, P72, P73, P74, P75, P76, P77, P78, P79, P80, P82, P83, P84, P86, P88, P89, P90, P91, P92, P93, P94, P95, P96, P98, P99, P100, P101, P102, P103, P104, P105, P106, P107, P108, P109, P110, P111, P112, P113, P114, P115, P116, P117, P118, P119, P121, P122, P123, P124, P125, P126, P127, P128, P129, P130, P131, P132, P133, P134, P135, P136, P137, P138, P139, P140, P141, P142, P143, P144, P145, P146, P147, P148, P149, P150, P151, P152, P153, P154, P155, P156, P157, P158, P159, P160, P161:

- RQ1. What are the input configurations, formats, and notations used in the requirements in requirements-driven automated software testing?
- RQ2. What are the frameworks, tools, processing methods, and transformation techniques used in requirements-driven automated software testing studies?
- RQ3. What are the test formats and coverage criteria used in the requirements-driven automated software testing process?
- RQ4. How do existing studies evaluate the generated test artifacts in the requirements-driven automated software testing process?
- RQ5. What are the limitations and challenges of existing requirements-driven automated software testing methods in the current era?

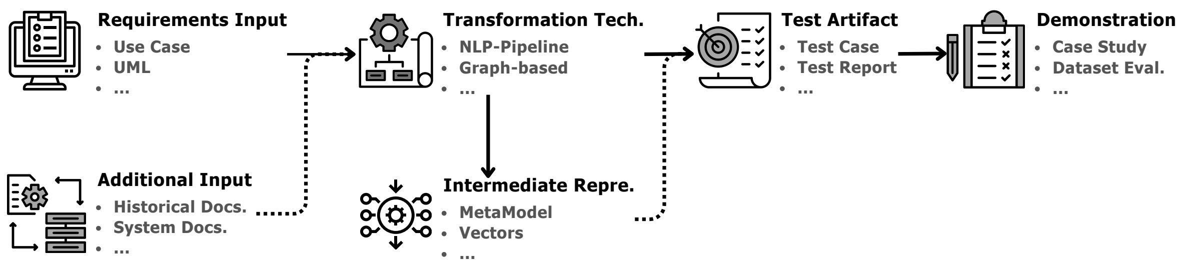


Figure 1. The General Framework of REDAST Methodology (The dotted route is not technically necessary.)

Structure. Section 2 discusses the background concepts and related work for REDAST. Section 3 presents our methodology and process of conducting our systematic review. Section 4 describes the taxonomy behind REDAST process. Section 5 discusses results from our five RQs. Section 6 examines threats to validity of our study. Section 7 discusses the insights from our results and the REDAST research roadmap. Section 8 concludes the paper.

2. Background and Related Work

2.1. Requirements Engineering

Requirement engineering (RE) is the initial phase in software development, guiding all subsequent stages [10,11]. The RE process requires gathering user needs and implementing the non-structured requirements into modeling language or other formed statements [12,13]. It encompasses various activities tailored to the specific demands of software systems, with requirements elicitation, analysis, specification, and validation being the most necessary stages [14,15]. Requirements can be broadly categorized into functional and non-functional requirements [16]. Requirements can be specified in different formats, e.g., using natural language (NL), modeling languages, such as UML and SysML, templates, such as use cases, or using formal notations. Thus, rather than using a single categorization for requirements in REDAST, we adopted multiple-level analysis in RQ1.

2.2. Automated Software Testing and Requirements Engineering

Software testing aims to provide objective, independent information about the quality of software and the risk of its failure to users or sponsors [17,18]. Automated software testing is using automation techniques to use specialized tools and scripts to execute test cases on a software application without manual intervention, which can improve time efficiency and human resource efficiency [19,20]. While the satisfaction of stakeholders is one of the priorities in software testing, the relationship between software requirements and testing becomes a critical focus in SDLC [21,22]. The alignment between different stages of verification and validation, e.g., system analysis and system testing is key for effective software quality assurance. Here, we primarily focus on the requirements specification and testing, while testing verifies that the software meets its specified requirements. This relationship is fundamental to ensuring the final product aligns with stakeholder expectations and functions correctly.

2.3. REDAST Secondary Studies

REDAST studies have been long investigated in past research. However, only limited studies systematically discussed RE-driven automated software testing. Atoum et al. [23] conducted a systematic study that examines the requirements of quality assurance and validation, where they reported a test-oriented approach. Unterkalmsteiner et al. [24] built a taxonomy for aligning requirements engineering and software testing to enhance coordination between these activities. They pointed out the importance of integrating requirements into the testing process, which contains some REDAST studies. Mustafa et al. [5]'s literature review is the most related paper. They investigated 30 selected papers by 2018 and limitedly analyzed the requirements-driven testing process from requirements input, techniques, and output perspectives. Their review provides a basic view of these parts but did not comprehensively discuss the details from various dimensions and levels due to the depth of understanding.

3. Research Methodology

In this section, we discuss the process of conducting our systematic review, e.g., our search strategy for data extraction of relevant studies, based on the guidelines of Kitchenham et al. [9] to conduct SLRs and Petersen et al. [25] to conduct systematic mapping studies (SMSs) in Software Engineering. In this systematic review, we divide our work into a four-stage procedure, including planning, conducting, building a taxonomy, and reporting the review, illustrated in Figure 2. The four stages are as follows: (1) the *planning* stage involved identifying research questions (RQs) and specifying the detailed research plan for the study; (2) the *conducting* stage involved analyzing and synthesizing the existing primary studies to answer the research questions; (3) the *taxonomy* stage was introduced to optimize the data extraction results and consolidate a taxonomy schema for REDAST methodology; (4) the *reporting* stage involved the reviewing, concluding and reporting the final result of our study.

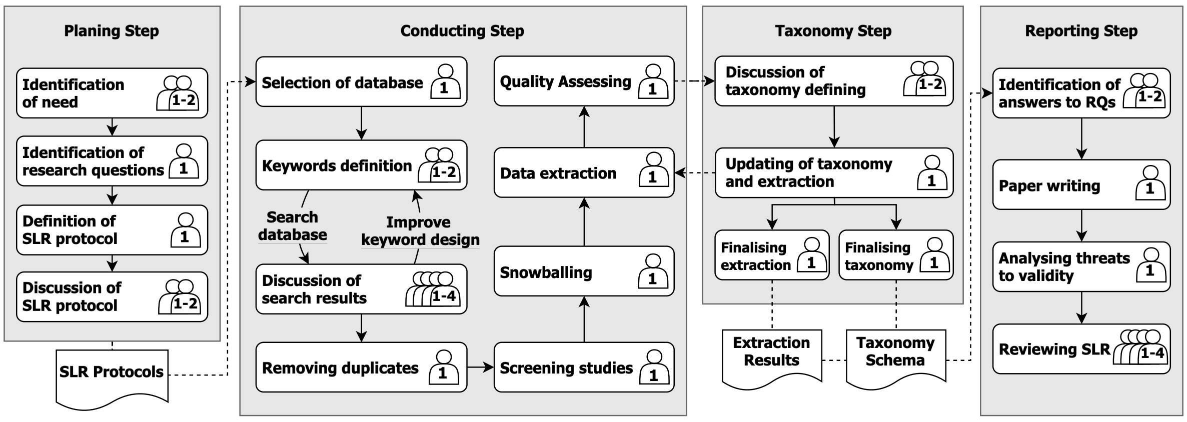


Figure 2. Systematic Literature Review Process

3.1. Research Questions

In this study, we developed five research questions (RQs) to identify the input and output, analyze technologies, evaluate metrics, identify challenges, and identify potential opportunities.

RQ1. What are the input configurations, formats, and notations used in the requirements in requirements-driven automated software testing? In requirements-driven testing, the input is some form of requirements specification – which can vary significantly. RQ1 maps the input for REDAST and reports on the comparison among different formats for requirements specification.

RQ2. What are the frameworks, tools, processing methods, and transformation techniques used in requirements-driven automated software testing studies? RQ2 explores the technical solutions from requirements to generated artifacts, e.g., rule-based transformation applying natural language processing (NLP) pipelines and deep learning (DL) techniques, where we additionally discuss the potential intermediate representation and additional input for the transformation process.

RQ3. What are the test formats and coverage criteria used in the requirements-driven automated software testing process? RQ3 focuses on identifying the formulation of generated artifacts (i.e., the final output). We map the adopted test formats and analyze their characteristics in the REDAST process.

RQ4. How do existing studies evaluate the generated test artifacts in the requirements-driven automated software testing process? RQ4 identifies the evaluation datasets, metrics, and case study methodologies in the selected papers. This aims to understand how researchers assess the effectiveness, accuracy, and practical applicability of the generated test artifacts.

RQ5. What are the limitations and challenges of existing requirements-driven automated software testing methods in the current era? RQ5 addresses the limitations and challenges of existing studies while exploring future directions in the current era of technology development.

3.2. Searching Strategy

The overview of the search process is exhibited in Fig. 3, which includes all the details of our search steps.

Table 1. List of Search Terms

Terms Group	Terms
Test Group	test*
Requirement Group	requirement* OR use case* OR user stor* OR specification*
Software Group	software* OR system*
Method Group	generat* OR deriv* OR map* OR creat* OR extract* OR design* OR priorit* OR construct* OR transform*

Table 2. Selection Criteria

Criterion ID	Criterion Description
S01	Papers written in English.
S02-1	Papers in the subjects of "Computer Science" or "Software Engineering".
S02-2	Papers published on software testing-related issues.
S03	Papers published from 1991 to the present.
S04	Papers with accessible full text.

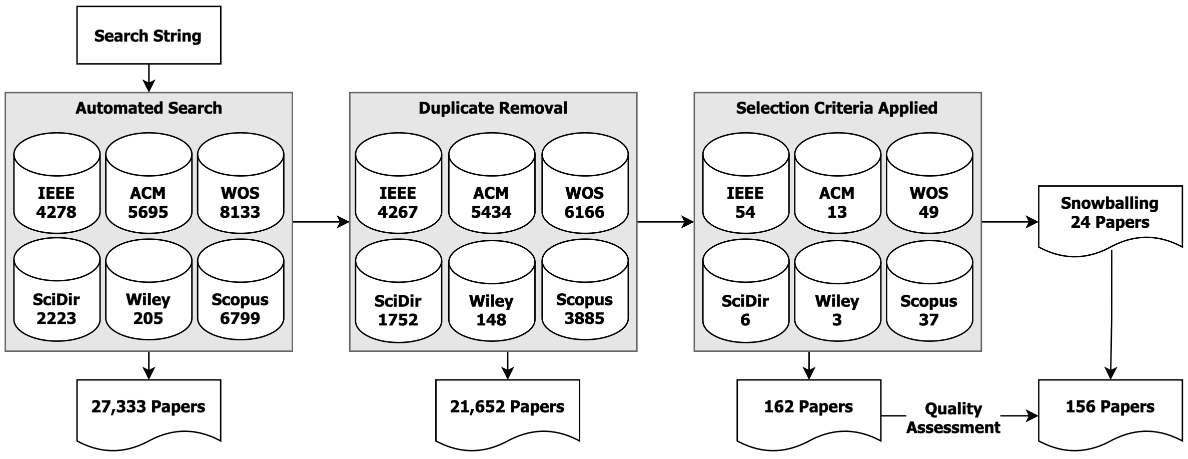


Figure 3. Study Search Process

3.2.1. Search String Formulation

Our research questions (RQs) guided the identification of the main search terms. We designed our search string with generic keywords to avoid missing out on any related papers, where four groups of search terms are included, namely “test group”, “requirement group”, “software group”, and “method group”. In order to capture all the expressions of the search terms, we use wildcards to match the appendix of the word, e.g., “test*” can capture “testing”, “tests” and so on. The search terms are listed in Table 1, decided after iterative discussion and refinement among all the authors. As a result, we finally formed the search string as follows:

ON ABSTRACT ((“test*”) **AND** (“requirement*” **OR** “use case*” **OR** “user stor*” **OR** “specifications”) **AND** (“software*” **OR** “system*”) **AND** (“generat*” **OR** “deriv*” **OR** “map*” **OR** “creat*” **OR** “extract*” **OR** “design*” **OR** “priorit*” **OR** “construct*” **OR** “transform*”))

The search process was conducted in September 2024, and therefore, the search results reflect studies available up to that date. We conducted the search process on six online databases: IEEE Xplore, ACM Digital Library, Wiley, Scopus, Web of Science, and Science Direct. However, some databases were incompatible with our default search string in the following situations: (1) unsupported for searching within abstract, such as Scopus, and (2) limited search terms, such as ScienceDirect. Here, for (1) situation, we searched within the title, keyword, and abstract, and for (2) situation, we separately executed the search and removed the duplicate papers in the merging process.

3.2.2. Automated Searching and Duplicate Removal

We used advanced search to execute our search string within our selected databases, following our designed selection criteria in Table 2. The first search returned 27, 333 papers. Specifically for the duplicate removal, we used a Python script to remove (1) overlapped search results among multiple databases and (2) conference or workshop papers, also found with the same title and authors in the other journals. After duplicate removal, we obtained 21, 652 papers for further filtering.

3.2.3. Filtering Process

In this step, we filtered a total of 21, 652 papers using the inclusion and exclusion criteria outlined in Table 3. This process was primarily carried out by the first and second authors. Our criteria are

Table 3. Inclusion and Exclusion Criteria

ID	Description
Inclusion Criteria	
I01	Papers about requirements-driven automated system testing or acceptance testing generation, or studies that generate system-testing-related artifacts.
I02	Peer-reviewed studies that have been used in academia with references from literature.
Exclusion Criteria	
E01	Studies that only support automated code generation, but not test-artifact generation.
E02	Studies that do not use requirements-related information as an input.
E03	Papers with fewer than 5 pages (1-4 pages).
E04	Non-primary studies (secondary or tertiary studies).
E05	Vision papers and grey literature (unpublished work), books (chapters), posters, discussions, opinions, keynotes, magazine articles, experience, and comparison papers.

structured at different levels, facilitating a multi-step filtering process. This approach involves applying various criteria in three distinct phases. We employed a cross-verification method involving (1) the first and second authors and (2) the other authors. Initially, the filtering was conducted separately by the first and second authors. After cross-verifying their results, the results were then reviewed and discussed further by the other authors for final decision-making. We widely adopted this verification strategy within the filtering stages. During the filtering process, we managed our paper list using a BibTeX file and categorized the papers with color-coding through BibTeX management software[1], i.e., "red" for irrelevant papers, "yellow" for potentially relevant papers, and "blue" for relevant papers. This color-coding system facilitated the organization and review of papers according to their relevance.

The screening process is shown below,

- **1st-round Filtering** was based on the title and abstract, using the criteria I01 and E01. At this stage, the number of papers was reduced from 21, 652 to 9, 071.
- **2nd-round Filtering**. We attempted to include requirements-related papers based on E02 on the title and abstract level, which resulted from 9, 071 to 4, 071 papers. We excluded all the papers that did not focus on requirements-related information as an input or only mentioned the term "requirements" but did not refer to the requirements specification.
- **3rd-round Filtering**. We selectively reviewed the content of papers identified as potentially relevant to requirements-driven automated test generation. This process resulted in 162 papers for further analysis.

Note that, especially for third-round filtering, we aimed to include as many relevant papers as possible, even borderline cases, according to our criteria. The results were then discussed iteratively among all the authors to reach a consensus.

3.2.4. Snowballing

Snowballing is necessary for identifying papers that may have been missed during the automated search. Following the guidelines by Wohlin [26], we conducted both forward and backward snowballing. As a result, we identified 24 additional papers through this process.

3.2.5. Data Extraction

Based on the formulated research questions (RQs), we designed 38 data extraction questions[2] and created a Google Form to collect the required information from the relevant papers. The questions included 30 short-answer questions, six checkbox questions, and two selection questions. The data extraction was organized into five sections: (1) basic information: fundamental details such as title, author, venue, etc.; (2) open information: insights on motivation, limitations, challenges, etc.; (3) requirements: requirements format, notation, and related aspects; (4) methodology: details, including

1 https://bibdesk.sourceforge.io/
2 https://drive.google.com/file/d/1yjy-59Juu9L3WHaOPu-XQo-j-HHGTbx_/view?usp=sharing

immediate representation and technique support; (5) test-related information: test format(s), coverage, and related elements. Similar to the filtering process, the first and second authors conducted the data extraction and then forwarded the results to the other authors to initiate the review meeting.

3.2.6. Quality Assessment

During the data extraction process, we encountered papers with insufficient information. To address this, we conducted a quality assessment in parallel to ensure the relevance of the papers to our objectives. This approach, also adopted in previous secondary studies [27,28], involved designing a set of assessment questions based on guidelines by Kitchenham et al. [9]. The quality assessment questions in our study are shown below:

- **QA1**. Does this study clearly state *how* requirements drive automated test generation?
- **QA2**. Does this study clearly state the *aim* of REDAST?
- **QA3**. Does this study enable *automation* in test generation?
- **QA4**. Does this study demonstrate the usability of the method from the perspective of methodology explanation, discussion, case examples, and experiments?

QA4 originates from an open perspective in the review process, where we focused on evaluation, discussion, and explanation. Our review also examined the study's overall structure, including the methodology description, case studies, experiments, and analyses. The detailed results of the quality assessment are provided in the Appendix. Following this assessment, the final data extraction was based on 156 papers.

4. Taxonomy

In literature review studies, the taxonomy schema plays a crucial role in shaping the quality of statistical analysis and addressing research questions. Aware that the complexity of the test generation process will lead to confusion in our results, we define a four-stage schema for our REDAST process based on our literature analysis and informed from Figure 4. In each of the schemas, recognizing that the entire SE life cycle is a practical process, we incorporated multiple categorizations to enhance the structure and clarity of the schemas.

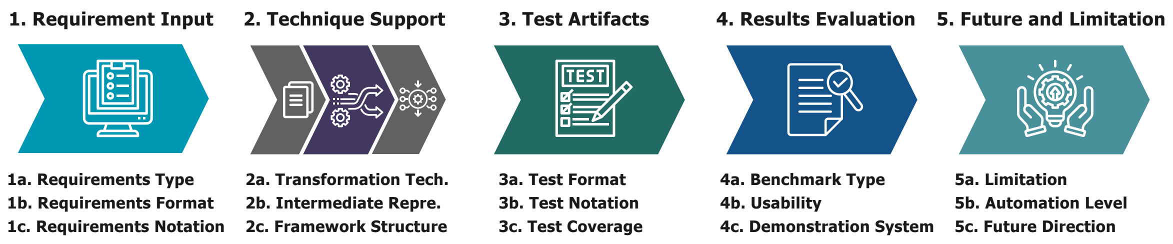


Figure 4. Overview of Taxonomy Schema in REDAST Studies

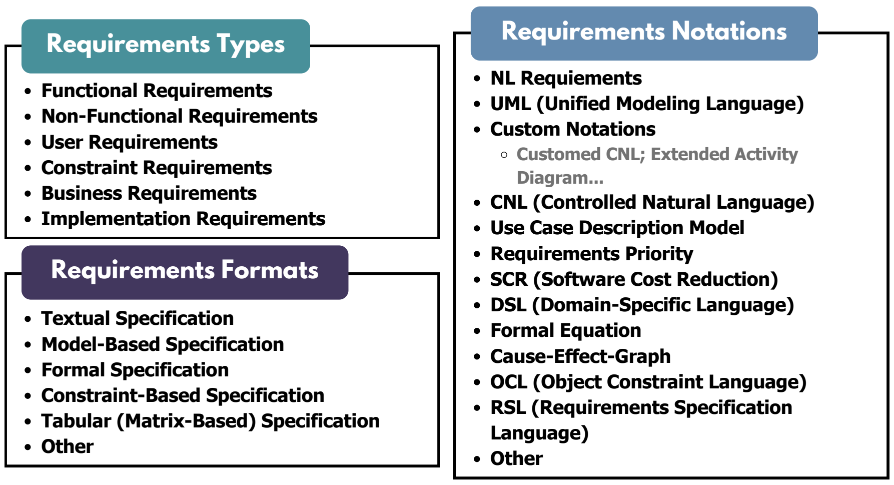

Requirements Types
• Functional Requirements
• Non-Functional Requirements
• User Requirements
• Constraint Requirements
• Business Requirements
• Implementation Requirements
Requirements Formats
• Textual Specification
• Model-Based Specification
• Formal Specification
• Constraint-Based Specification
• Tabular (Matrix-Based) Specification
• Other
Requirements Notations
• NL Requiements
• UML (Unified Modeling Language)
• Custom Notations
◦ Customed CNL; Extended Activity Diagram...
• CNL (Controlled Natural Language)
• Use Case Description Model
• Requirements Priority
• SCR (Software Cost Reduction)
• DSL (Domain-Specific Language)
• Formal Equation
• Cause-Effect-Graph
• OCL (Object Constraint Language)
• RSL (Requirements Specification Language)
• Other

Figure 5. RQ1 Taxonomy - Requirements Input in REDAST Studies

4.1. Designing of Taxonomy Schema

Several studies have discussed the taxonomy categories in the RE and software testing domains in previous surveys. However, these existing schemas do not provide a comprehensive view of the REDAST process, especially for the detailed transforming process from requirements to test artifacts. In order to address the gap, we propose a hierarchical structure that encompasses (1) requirements input, (2) technical methodology, (3) test outcomes, and (4) results evaluation by referring to some related studies [6,29]. The overview of the taxonomy schema in our literature review is illustrated in Fig. 4, where we not only exhibit the taxonomy schema in this figure but also indicate the REDAST procedure.

4.2. Requirements Input Category

Requirements are the necessary input of the REDAST process. The formulation of requirements input will decisively affect the choices of the following steps, including processing technology, framework designing, and so on. In the Requirements input section (taxonomy in Figure 5), we focus on the formulation of requirements based on the schema outlined in [30,31], which corresponds to RQ1 about requirements. However, as we mentioned, there is no universal requirements categorization. Thus, we adopted multiple categorizations in RQ1. The requirements type consists of (1) functional requirements, (2) non-functional requirements, (3) business requirements, (4) user requirements, (5) constraint requirements, and (6) implementation requirements, which describe the covering scale of requirements [12,32]. Requirements format is the second categorization in requirements, which includes constraint-based, executable, formal (mathematical), goal-oriented, model-based, scenario-based, tabular (matrix-based), and textual specifications [33–37]. The format categorization of requirements specifications represents how requirements can be structured, documented, and communicated based on their intended purpose and context. The last categorization in RQ1 concerns the specific notation of requirements specifications, such as use case, NL specification, and SysML, which aims to illustrate the adoption trend in requirements notation. These schemas are expected to cover a broad range of requirements and provide a comprehensive view of the requirements input. Note that our study focuses specifically on requirements-related aspects, deliberately excluding papers focused on design-level information. For instance, Yang et al. [38] proposed an automated test scenario generation method using SysML for modeling system behavior in the system design phase. Although SysML is a commonly used specification format for requirements and system architecture, this paper was excluded from our literature review due to its emphasis on system design rather than system requirements.

4.3. Transformation Techniques Category

The transformation techniques support the generation process from requirements to test artifacts, wherein we attempt to analyze the details of the methodologies employed. Figure 6 shows the taxonomy schema of RQ2. We identified the following aspects,

- *Transformation Techniques*. From requirements to generated test artifacts, the transformation techniques are expected to transfer requirements to readable, understandable, and generation-friendly artifacts for test generation. However, varying from different usage scenarios, various types of techniques are adopted in the transformation framework. Even though there are only limited studies explicitly discussing the transformation techniques for REDAST, we referred to the survey in related fields [39–42], such as automated software testing and software generation, to finalize our schema for transformation techniques. The categorization for transformation techniques can be formulated as five categories: (1) **rule-based** techniques rely on predefined templates or rules to formulate requirements to test artefacts; (2) **meta-model-based** techniques employ the meta-models to define the behavior, structure, relationships, or constraints to enable enhanced expression ability; (3) **graph-based** techniques mainly use the graph as representation (e.g., state-transition graphs, dependency graphs) with traversing or analyzing on paths, nodes, or conditions; (4) **natural language processing pipeline-based** techniques focus on leveraging open-source NLP tools for REDAST, like text segmentation and syntax analysis; (5) **machine learning-based** techniques leverage ML (including deep learning) in the REDAST process, which always involves the patterns or feature learning process using training data.
- *Intermediate Representations* are related to the optional steps in the generation framework. Some papers employ a stepwise transformation approach instead of directly transforming requirements into test artifacts. This approach generates intermediate artifacts that enhance the traceability and explainability of the methodology. For example, the unstructured NL requirements could be transformed into an intermediate more structured representation that facilitates the generation of test artifacts. While intermediate representations are derived from requirements, we employed a categorization method for representation types similar to that used in requirements schemas.
- *Additional Inputs*. In addition to simply using requirements as input, some frameworks accommodate additional input types, such as supporting documents, user preferences, and more. To analyze these frameworks from the perspective of input composition, we introduce additional inputs that categorize and examine the variety of inputs utilized.
- *Framework Structure* refers to the underlying architectural approach used to transform requirements into test artifacts. It determines how different stages of transformation interact. Within the REDAST framework, transformation methodologies are categorized into four distinct structures ([43,44]): (1) **Sequential** – Follows a strict, ordered sequence of transformation steps, maintaining logical continuity without deviations. Each step builds upon the previous one; (2) **Conditional** – Introduces decision points that enable alternative transformation paths based on specific conditions, increasing adaptability to varying requirements; (3) **Parallel** – Allows simultaneous processing of different representations across multiple transformation units, significantly improving efficiency; (4) **Loop** – Incorporates iterative cycles for continuous refinement, ensuring enhanced quality through repeated validation and adjustment.

Thus, based on the above aspects, we plan to introduce two aspects in RQ2: transformation techniques and framework design, where the input portion, framework structure, and intermediate representation are included in the framework design. In our review, we also explored the advantages and disadvantages of various techniques, with a particular emphasis on recent advancements in LLMs. This category is related to RQ2.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 10 February 2025 doi:10.20944/preprints202502.0628.v1

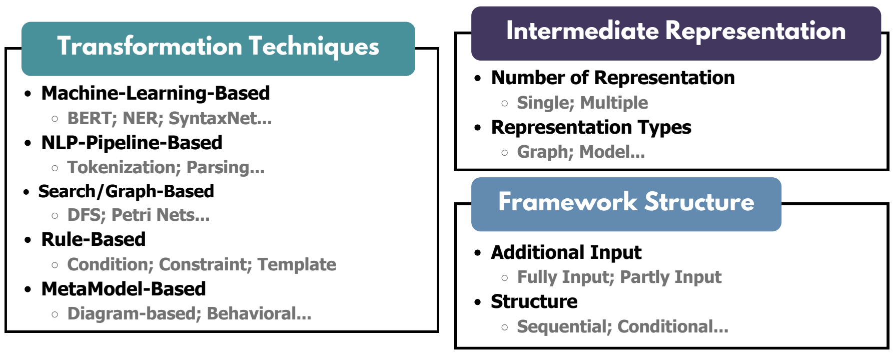


Figure 6. RQ2 Taxonomy - Transformation Techniques in REDAST Studies

4.4. Test Artifacts Category

In the *Test Artifacts* section, we aim to focus on the formulation of generated testing artifacts. The illustration of RQ3 is shown in Figure 7. The branches within the test artifacts category, such as test format, notation, and coverage, have been explored in previous surveys [45–47]. Additionally, we introduce a new categorization based on the abstraction level of test artifacts, specifically designed for the REDAST process. While the generated test artifacts are typically applied to system testing and acceptance testing [48], they commonly include code, test descriptions, or test reports. The abstraction level categorization includes three categories: abstract, concrete, and report, which refers to the test artifacts in the system and acceptance testing. (1) Abstract test artifacts cannot be directly executed but provide enhanced traceability and coverage for requirements. (2) Executable test artifacts are executable, with multiple concrete artifacts often corresponding to a single requirement. (3) Report-level artifacts represent the outcomes of executing the test artifacts. The results of the test artifact-related categories are presented in response to RQ3.

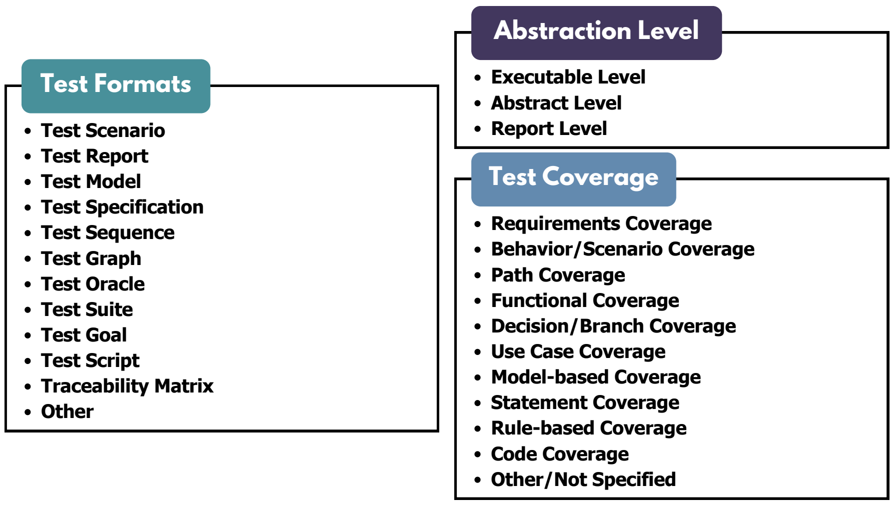


Figure 7. RQ3 Taxonomy - Test Artifacts in REDAST Studies

4.5. Results Demonstration Category

REDAST always introduces case demonstration or dataset evaluation to assess the quality of the generated test artifacts. The taxonomy of RQ4 is exhibited in Figure 8 Here, we include evaluation as a separate category in the taxonomy schema to obtain some results about the quality assessment criteria of test artifacts. Specifically, we planned to (1) conclude the evaluation methods used in relevant studies, (2) categorize the software platforms in the demonstration, and (2) report typical examples and analyze their efficacy based on their usage scenarios, pros, and cons, where [49] was opted as the guideline for designing the schema of this category. We introduce demonstration types and software platforms to illustrate the details of the demonstration method. As for the software platform, this categorization is introduced to identify the software platform adopted in the case demonstrations.

Besides, we introduced a categorization for usability in evaluation, where we will manually evaluate the selected studies and illustrate their results for different parts, including methodology explanation, discussion, case example, and experiment.

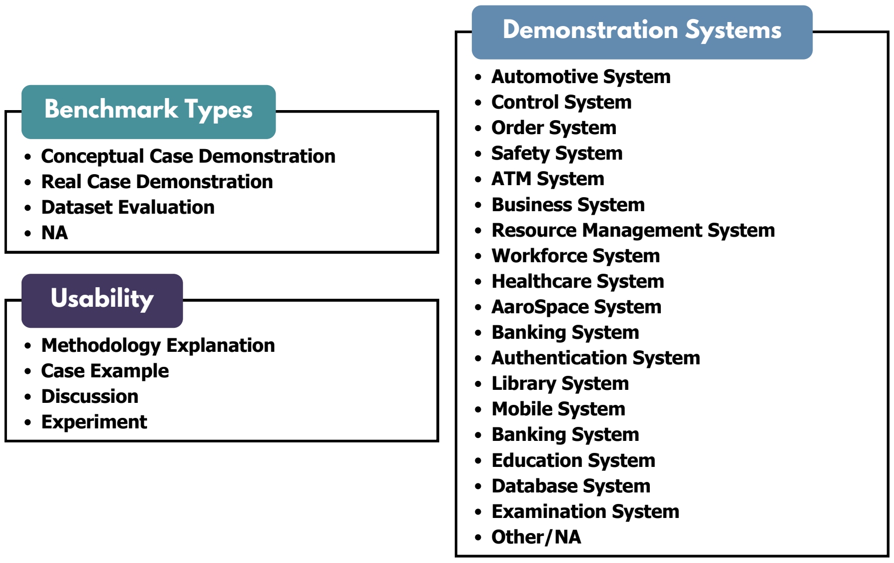


Figure 8. RQ4 Taxonomy - Demonstration Methods in REDAST Studies

4.6. Future and Limitation Category

RQ5 primarily examines the limitations and future directions of REDAST studies, which are illustrated in Figure 9. The categorizations for these aspects were determined post hoc, based on our analysis of the results; therefore, detailed categorization methods will not be presented. Additionally, considering the importance of automation in REDAST studies, we provide an analysis of the automation levels observed in the selected studies. Four levels of automation are defined as follows:

- *Fully Automated (End-to-End Automation)*: Studies in this category require no human intervention or operation.
- *Highly Automated (Automation-Dominant)*: These studies demonstrate a high degree of automation, with human intervention incorporated into the methodology but not essential.
- *Semi-Automated (Automation-Supported)*: This level involves significant manual operations at specific stages of the process.
- *Low Automated (Minimal Automation)*: Studies at this level exhibit only basic automation, relying primarily on manual operations across all steps.

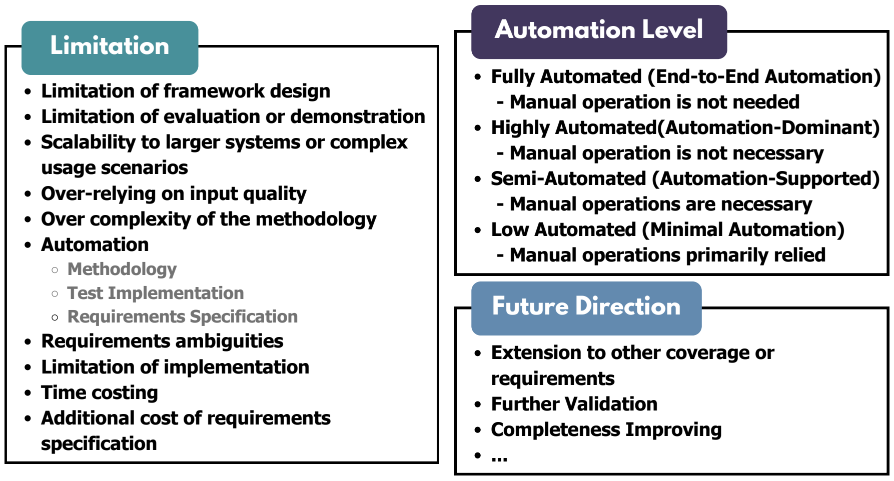

Limitation
• Limitation of framework design
• Limitation of evaluation or demonstration
• Scalability to larger systems or complex usage scenarios
• Over-relying on input quality
• Over complexity of the methodology
• Automation
◦ Methodology
◦ Test Implementation
◦ Requirements Specification
• Requirements ambiguities
• Limitation of implementation
• Time costing
• Additional cost of requirements specification
Automation Level
• Fully Automated (End-to-End Automation)
- Manual operation is not needed
• Highly Automated(Automation-Dominant)
- Manual operation is not necessary
• Semi-Automated (Automation-Supported)
- Manual operations are necessary
• Low Automated (Minimal Automation)
- Manual operations primarily relied
Future Direction
• Extension to other coverage or requirements
• Further Validation
• Completeness Improving
• ...

Figure 9. RQ5 Taxonomy - Future and Limitations in REDAST Studies

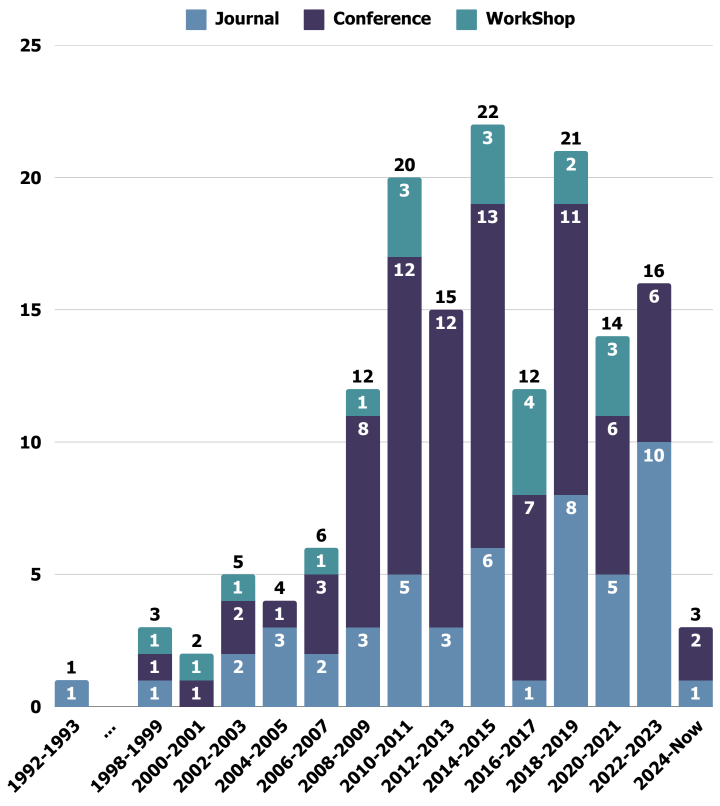
5. Results

In this section, we present the results of the SLR on requirements-driven automated software testing, where the results are structured based on our RQs.

5.1. Trend: General Results of the Publications

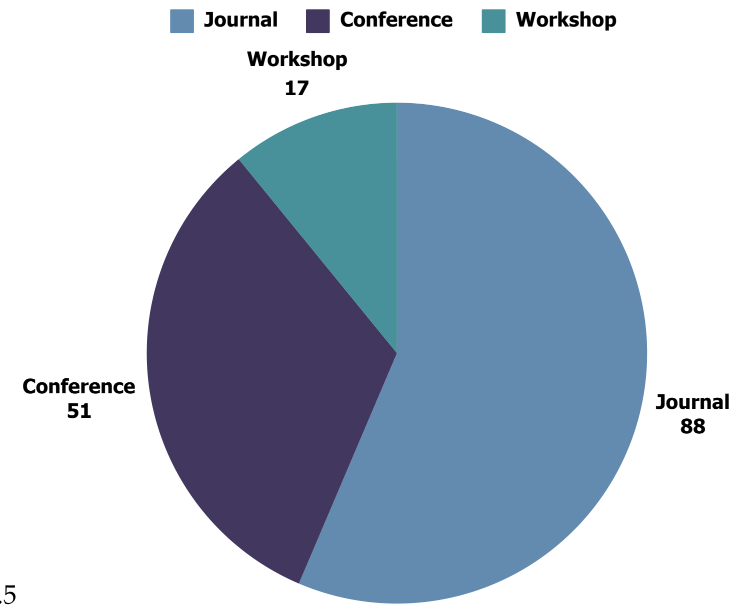
Before addressing the research questions, we present the publication trends of the 156 primary studies on REDAST, including study distribution, venue names, and features of the selected studies.

The study distribution is analyzed in two parts: (a) study distribution by the publication year and (b) study distribution by publication type, which are illustrated in Figures 10 and 11, respectively.



.5

Figure 10. Study Distribution by Publication Year



.5

Figure 11. Study Distribution by Publication Type
Figure 12. Study Distribution (Trend)

In analyzing the distribution of REDAST studies by year, we observed that the first study was published in 1993, followed by a steady annual increase. A notable surge occurred around 2008, prompting further investigation into the technical differences between studies conducted before and after this period. Two key conclusions emerged: (1) An increased adoption of NLP-pipeline-based methods from 2008 onward. Prior to 2008, only 20% of studies employed these methods, whereas from 2008 to 2013, the proportion rose to 31.9%. (2) A decline in the use of graph-based methods after 2008. The prevalence of graph-based approaches decreased from 35% before 2008 to 19.14% in the subsequent period from 2008 to 2013. By comparing with the landmark studies published between 2006 and 2008, including works by Hinton et al., [50] and Van der Maaten and Hinton[51], we believe that deep learning technologies, including LLMs and Convolutional Neural Networks (CNNs), can promote the adoption of NLP-pipeline-based methods in SE domain.

Table 4. Publication Venues with Two or More Studies in Selected Papers (Trend)

Venue Names	**Type**	**Num.**
IEEE International Requirements Engineering Conference (RE)	Conference	6
IEEE International Conference on Software Quality, Reliability, and Security (QRS)	Conference	6
IEEE International Conference on Software Testing, Verification, and Validation (ICST) Workshops	Workshop	5
IEEE Transactions on Software Engineering (TSE)	Journal	4
Software Quality Journal (SQJ)	Journal	3
Science of Computer Programming	Journal	3
IEEE International Conference on Software Testing, Verification and Validation (ICST)	Conference	3
International Conference on Quality Software (QSIC)	Conference	3
International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)	Conference	3
Innovations in Systems and Software Engineering	Journal	3
IEEE International Workshop on Requirements Engineering and Testing (RET)	Workshop	3
ACM SIGSOFT Software Engineering Notes	Journal	3
Journal of Systems and Software (JSS)	Journal	2
IEEE International Symposium on Software Reliability Engineering (ISSRE)	Conference	2
IEEE International Requirements Engineering Conference Workshops (REW)	Workshop	2
International Journal of System Assurance Engineering and Management	Journal	2
International Conference on Enterprise Information Systems (ICEIS)	Conference	2
International Conference on Emerging Trends in Engineering and Technology (ICETET)	Conference	2
Electronic Notes in Theoretical Computer Science	Journal	2
Australian Software Engineering Conference (ASWEC)	Conference	2
International Journal of Advanced Computer Science and Applications	Journal	2
Electronics	Journal	2

Regarding the publication type distribution, most studies were published in conferences (56.41%) and journals (32.69%). Our selected studies were published across 115 different venues. We present the venues with two or more published studies in Table 4. The IEEE International Requirements Engineering Conference, the IEEE International Conference on Software Quality, Reliability, and Security, and the IEEE Transactions on Software Engineering have the highest percentages in their respective categories, which is unsurprising given their top-level reputation in the software engineering field.

Table 5. Overview of Target Software in Selected Papers (Trend)

Target Software Systems	Paper ID	Num.
General Software	P2, P3, P5, P6, P7, P8, P9, P10, P13, P14, P15, P16, P17, P18, P19, P23, P24, P25, P30, P31, P34, P35, P36, P37, P39, P40, P41, P42, P43, P46, P51, P52, P53, P54, P55, P57, P62, P63, P65, P68, P69, P70, P71, P73, P74, P75, P76, P78, P79, P82, P83, P84, P86, P88, P89, P92, P93, P96, P99, P100, P103, P104, P105, P106, P109, P114, P116, P117, P118, P119, P121, P122, P124, P125, P126, P127, P128, P129, P130, P132, P135, P136, P137, P138, P139, P141, P142, P143, P144, P145, P146, P148, P150, P151, P152, P153, P157, P158, P159, P160	100
Embedded System	P29, P33, P38, P60, P61, P94, P98, P111, P113, P140, P147, P149, P155, P161	14
Web Services System	P1, P4, P11, P21, P56, P67, P112	7
Safety-Critical System	P27, P50, P72, P101, P123, P131	6
Timed Data-flow Reactive System	P80, P108, P133, P134	4
Reactive System	P45, P107, P156	3
Real-time Embedded System	P26, P64, P115	3
Product Line System	P32, P48	2
Object-Oriented System	P28, P66	2
Telecommunication Application	P22, P95	2
Automotive System	P59, P102	2
Space Application	P44	1
SOA-based System	P90	1
Labeled Transition System	P49	1
Healthcare Application	P110	1
Event-driven System	P154	1
Cyber-Physical System	P58	1
Core Business System	P20	1
Concurrent System	P12	1
Complex Dynamic System	P91	1
Aspect-Oriented Software	P47	1
Agricultural Software	P77	1

Besides the trending information, we also investigated the target software in the selected studies. The results of the target software are exhibited in Table 5. Most of the selected studies are designed for general software (100/156), followed by embedded systems (14/156) web services systems (7/156), safety-critical systems (6/156), and so on.

5.2. *RQ1: Requirements Specification Formulation*

Based on our taxonomy schema in Section 4, we explore the techniques for requirements type, format, and notation in the selected studies. Moreover, by comparing the characteristics of the studies, we analyze the features in different adoptions of specification techniques.

5.2.1. Requirements Types

In this section, we mainly discuss the results of the requirements type of the selected studies, where the requirements could be classified into six categories, functional, non-functional, user, constraint, business, and implementation requirements The total number of the adopted requirements type is not exactly the same as the number of publications, where some studies cover multiple requirements types as the input of the methodologies.

Based on the results in Table 6, almost all of the REDAST studies cover functional requirements (152/156), followed by non-functional requirements (27/156), user requirements (10/156), constraint requirements (7/156), business requirements (7/156), and implementation requirements (1/156). This trend can be attributed to functional requirements being inherently *testable*, as they explicitly define the system's expected behavior. Unlike non-functional requirements, which often involve abstract or qualitative criteria, functional requirements provide concrete, measurable specifications that align well with the design of test artifacts. Additionally, we noticed that some domain-specific requirements have been adopted in several studies, such as the security requirements in safety-critical systems (e.g., P118 P123), timing requirements in reactive systems (e.g., P45 P45), and so on (e.g., P45 P45, P28 P28, P119 P124). These requirements could be considered in the other requirements categories.

Table 6. Requirements Types in Selected Studies (RQ1)

Requirements Types	**Paper IDs**	**Num.**
Functional Requirements	Almost all papers support functional requirements, except P18, P74, P79, P155, P156.	152
Non-functional Requirements	P2, P33, P41, P48, P50, P51, P56, P57, P61, P68, P69, P71, P73, P76, P79, P83, P88, P90, P96, P116, P123, P124, P146, P155, P156, P157, P161	27
User Requirements	P2, P43, P74, P84, P88, P92, P119, P121, P139, P142	10
Constraint Requirements	P86, P91, P94, P100, P101, P114, P161	7
Business Requirements	P18, P51, P74, P89, P94, P95, P137	7
Implementation Requirements	P95	1

5.2.2. Requirements Specification Format

Requirements specification is categorized into seven different types: textual, model-based, constraint-based, formal (mathematical), tabular (matrix-based), and other specifications. The specification formats adopted in the selected studies are presented in Table 7. Note that we found some studies that use the transformation method further to convert raw requirement input into the other requirement formats. Here, we only consider the first raw requirement input in this section. For example, P76 P76 adopts model-based specification as the raw input for requirements. However, the requirements are further converted to model-based requirements. To clarify our objective, we only consider scenario-based requirements in RQ1. The intermediate representations are discussed in RQ2.

Table 7. Requirements Format in Selected Studies (RQ1)

Requirements Formats	**Paper IDs**	**Num.**	
Textual	P2,P5,P6,P7,P8,P9,P10,P11,P12,		
P13,P14,P15,P16,P17,P18,P19,P22,P23,P24,			
P25,P26,P28,P31,P32,P34,P36,P37,P38,P39,P41,P42,P43,P44,P45,P46,			
P47,P48,P51,P52,P54,P55,P57,P58,P60,P61,P62,P63,P64,P65,P67,P68,P70,			
P71,P75,P76,P77,P80,P82,P83,P84,P86,P88,P92,P93,P96,P98,P99,P102,P103,			
P106,P108,P109,P110,P112,P114,P116,P117,P119,P121,P122,P123,P125,P126,			
P127,P128,P129,P130,P132,P133,P134,P135,P136,P137,P138,P139,P140,P145,			
P147,P149,P151,P152,P153,P156,P157,P160	105		
Model-Based	P1,P4,P7,P8,P10,P11,P14,P15,		
P16,P17,P20,P21,P25,P27,P28,P30,P33,P35,P46,P49,			
P50,P53,P56,P57,P65,P66,P69,P73,P74,P75,P78,P86,			
P89,P90,P95,P96,P99,P104,P109,P110,P115,P119,P123,			
P131,P137,P143,P144,P145,P148,P150,P152,P153,P154,P161	54		
Formal	P3,P21,P45,P58,P59,P72,		
P79,P105,P110,P111,P113,P124,P155	13		
Constraint-Based	P3,P27,P33,P59,P72,		
P78,P91,P100,P101,P105,P154	11		
Tabular (Matrix-Based)	P29,P33,P78,P92,P107,P1	7	
Other	P40,P94,P118,P142,P141,	7	8

Table 8. Requirements Format Results of Selected Studies with Unique Formulations (RQ1)

Requirements Formats	Requirements Formulations	Num.
Model Specification	Behavior Tree, Graph-based, Finite State Machine, Specification and Description Language (SDL), Use Case Map, Activity Diagram, Communication Diagram, Misuse Case, Conditioned Requirements Specification, Use Case, Scenario conceptual model, UML, Models, Sequence Diagram, State Machine Diagram, Scenario, NL requirements, pseudo-natural language, Behavior Model, Extended Use Case Pattern, Linear temporal logic, Communication Event Diagram (CED), SysML, Formal Use Case, Textual Normal Form, Object Diagram	30
Textual Specification	Graph-based, Scenario specification, User Story, NL Requirements, Use Case Map, Misuse Case, Use Case, Scenario Model, Textual, Scenario conceptual model, DSL, Use Case/Scenario, NL requirements (Behavior), Business Process Modeling Language, Signal Temporal Logic (STL), Restricted Signals First-Order Logic (RFOL), Formal Requirements Specification, Formal Use Case	26
Constraint Specification	Graph-based, OCL, DSL, Finite State Machine, Formal Requirements Specification, SysReq-CNL, Scenario, UML	7
Formal Specification	Signal Temporal Logic (STL), Restricted Signals First-Order Logic (RFOL), Formal Requirements Specification, Linear Temporal Logic, NL requirements, Use Case	7
Other Specification	Requirements Dependency, Requirements Priorities, Safety Requirements Specification, Test requirements	4
Tabular Specification	Scenario, Tabular Requirements Specification, Finite State Machine	3

- *Textual Specification.* 105 studies adopted textual specification methods. Our analysis shows that textual specification is the most commonly used format in REDAST studies. Textual specification, written in natural language (NL), is the predominant choice. Besides the textual specification, NL is widely integrated into other specification formats, including formal (mathematical) and tabular specifications. NL-based requirements are generally favored in RADAST studies due to their accessibility and ease of understanding, which supports both requirements description and parsing NL is commonly used in these studies due to its advanced explainability and flexibility. We separately discuss this category by distinguishing textual specifications from others and whether the requirements follow natural language logic [52,53]. Other formats especially involve specific specification rules or templates compared to textual specifications.
- *Model-based Specification.* 54 studies specify requirements using model-based specifications. Model-based approaches construct semi-formal or formal meta-models to represent and analyze requirements. Compared to textual specifications, meta-models have better abstraction capabilities for illustrating the behaviors (e.g., P1 P1, P27 P27, P35 P35, etc.), activities (e.g., P25 P25, P139 P144, P146 P151, etc.), etc., of a software system [54].
- *Constraint-based Specification.* We identified 11 studies in the selected papers that utilize constraint-based requirements specification. Constraint-based specification is also a welcomed requirements format, where Domain-Specific Language are adopted in the selected studies, e.g., P3 P3, P89 P92, P96 P100, etc. Constraint-based specification involves defining system properties as limits, conditions, or relationships that must hold within the system. Constraint-based specification can provide a concise and precise description of system behavior, especially in the context of complex software systems [55,56]. DSL is a typical constraint-based specification, e.g., P88 P91, P95 P99, and P96 P100.
- *Formal (Mathematical) Specification.* In the selected studies, we identified 13 papers that used formal (mathematical) specifications in requirements elicitation. The formal (mathematical) specification can translate natural language requirements into a precise and unambiguous specification that can be used to guide the development of software systems [57], where the typical formal requirements specification are assertions (e.g., P109 P113), controlled language (e.g., P79 P79, P108 P112), and so on. Unlike textual specification, the logical expression can describe software requirements unambiguously [58].

- *Tabular (Matrix-based) Specification.* We identified 7 studies that adopted tabular specifications. Tabular specification methods can formalize requirements in a structured and organized manner, where each row represents a requirement, and each column represents a specific attribute or aspect of the requirement [57]. Tabular specification can greatly improve traceability and make it more friendly for verification and validation.

5.2.3. Requirements Specification Notation

The requirements notation is an extended detail of the requirements format. We identified over 60 requirement notations across the selected studies. Specifically, for some variations of standard requirement notations, we categorized the similar notations into their original forms or grouped uncommon notations under the "other" category. The summary of these notations is presented in Table 9.

Table 9. Requirements Notation Results of Selected Studies (RQ1)

Requirements Notations	Paper IDs	Num.
NL requirements	P23 (PURE dataset), P26,P34 (Textual User Story), P37 (Usage Scenario), P38 (NL Requirements), P39 (Scenario Specification), P40 (Test requirements), P41 (Textual Use Case), P42,P44, P52 (Textual Use Case), P54,P55, P60 (Textual Use Case), P63,P67,P71 (Template-based), P77 (Scenario), P82,P84,P87, P88 (Textual Use Case), P93,P98,P102,P104,P106,P113,P116,P117, P121 (Claret Format), P122,P126,P127,P128,P129,P130,P135,P136,P138,P139,P140, P147 (Technical Requirements Specification), P149,P151, P157 (Positive and negative pair), P160	47
UML	P4,P7,P8,P10,P11,P15,P16,P17, P25 (Activity Diagram), P28 (Sequence Diagram), P35,P46,P49,P53,P56, P57,P66,P73,P74,P75,P89,P90, P96 (UML MAP), P99, P109 (Sequence Diagram), P110,P115,P119,P123, P145 (Activity Diagram), P152 (Activity Diagram), P153 (Sequence Diagram), P161 (Modeling and Analysis of Real Time and Embedded Systems)	33
Other	P18 (Semi-Structured NL), P21 (OWL-S Model), P29 (State-Transition Table), P58 (Formal NL Specification), P64 (Structured Requirements Specification), P65 (Class Diagram, Restricted-form of NL), P68 (Semi-Formal Requirements Description), P69 (Requirements Specification Modeling Language), P94 (Safety Requirements Specification), P107 (Expressive Decision Table), P125 (Textual Use Case), P131 (Functional Diagram), P137 (Requirement Description Modeling Language), P142 (Requirements Dependency Mapping), P143 (Specification and Description Language), P144 (Behavior Tree), P148 (Domain-Specific Modeling Language), P150 (Statechart Diagram), P158 (Risk Factor), P159 (Requirement Traceability Matrix)	20
Custom	P1 (Custom Metamodel), P3 (Constraint-based Requirements Specification), P9 (Custom CNL), P12 (Semi-Structured NL Extended Lexicon), P14 (Interaction Overview Diagram), P27 (SCADE Specification), P30 (Extended SysML), P32 (RUCM with PL extension), P47 (Aspect-Oriented PetriNet), P50 (Safety SysML State Machine), P59 (OCL-Combined AD), P62 (State-based Use Case), P86 (Contract Language for Functional PF Requirements (UML)), P92 (Textual Scenario based on tabular expression), P103 (NL requirements (Language Extended Lexicon)), P111 (Specification language for Embedded Network Systems), P114 (Requirements Specification Modeling Language)	17
CNL	P2, P5 (RUCM), P6 (Use Case Specification Language (USL)), P22 (RUCM), P24 (RUCM), P45, P61 (RUCM), P80, P83 (Restricted Misuse Case Modeling), P108, P112,P132,P133,P134,P156	15
Use Case Description Model	P13 (Use Case Description Model), P19 (Use Case Description Model), P36 (Use Case Description Model), P43 (Use Case Description Model), P48 (Use Case Description Model), P70 (Use Case Description Model), P76 (Use Case Description Model)	7
Requirements Priorities	P118 (Customer-assigned priorities, Developer-assigned priorities), P141 (Customer Assigned Priority), P146 (Stakeholder Priority)	3
SCR	P33,P78,P154	3
DSL	P91,P100,P101	3
Formal Equation	P79,P124,P155	3
Cause-Effect-Graph	P20,P95	2
OCL	P72,P105	2
RSL	P31,P51	2

Based on the results, we identified that Natural language (NL) requirements specification is the most frequently adopted notation in the selected studies (47 studies), followed by UML notation (33 studies), other (20 studies), and custom requirements (17 studies). Overall, this result aligns with the trend observed in the requirements format results, where natural language is widely adopted in REDAST methods.

- *NL Requirements Specification.* We found that 47 studies introduced natural language (NL) requirements specifications in their methods. NL requirements specifications are used not only in

REDAST studies but also in requirements elicitation and specification domains. For example, “shall” requirements (formally known as IEEE-830 style “shall” requirements [59]) are widely used for requirements specification, enabling less ambiguity and more flexibility. NL requirements specifications are applicable for various processing methods, such as condition detection (e.g., P23 P23) and semantic analysis (e.g., P63 P63).

- *Unified Modeling Language* Unified Modeling Language (UML) is a commonly used notation in model-based specification. We identified 33 studies that utilized UML in the selected papers. UML is versatile and can be combined with other notations to describe scenarios, behaviors, or events, effectively capturing functional requirements [60]. For instance, in the selected studies, P56 P56 introduced a tabular-based UML for requirements traceability, while P17 P17 employed UML use case diagrams specifically to depict requirement scenarios.
- *Controlled Natural Language.* In the selected studies, 15 papers opted for controlled natural language (CNL) as a requirement notation. CNL is partly based on natural language but is structured using the Rimay pattern [61], deviating from conventional expression syntax.
- *Use Case, User Story, and Their Variations.* Use cases, user stories, and their variations are distinct requirement notations in scenario-based specifications, sharing similar characteristics. These notations generally consist of a cohesive set of possible dialogues that describe how an individual actor interacts with a system or use textual descriptions to depict the operational processes of the system. In this way, the system behavior is vividly explained.
- *Other Specifications.* Other specification notations are not frequently adopted methods, where the “Other” category contains the notations that appear one time. Most of them are variations of common notations.

5.2.4. Findings: Cross-Analysis of Requirements Input and Target Software

As the first step in the REDAST process, the selection of requirements formulations predominately decides the usage scenario of the framework. More specifically, the end goal of the framework forces the researchers to select appropriate requirements formats and notations to describe the different system behaviors, events, or activities. Here, besides the results in requirements format, we cross-discuss the requirements format and target software (in Section 5.1) to illustrate the requirements preference in REDAST in the context of usage scenarios.

Textual requirements dominate across all categories, where general software (60%), Embedded Systems (50%), Real-Time Systems (67%), and other domains primarily select textual requirements as default. This trend suggests that, due to the flexibility and simplicity of textual requirements, textual requirements can handle most usage scenarios in REDAST.

Model-based requirements are preferred for structured systems. Detailly, the selection of model-based requirements, Web Serviced (P1 P1, P4 P4, P11 P11, P21 P21, P56 P56), Safety-Critical Systems (P27 P27, P50 P50), Object-Oriented Systems (P28 P28, P66 P66), Product Line Systems (P32 P32), and SOA-based Systems (P87 P90), indicates the preference of model-based requirements for service-oriented architectures, correctness and traceability assurance.

Formal and constraint-based requirements are crucial for high-reliability domains, wherein formal requirements and constraint-based requirements can additionally satisfy the needs of strict verification and validations, e.g., (1) for formal requirements, Safety-Critical Systems (P72 P72), Automotive Systems (P59 P59), Cyber-Physical Systems (P58 P58), Embedded Systems (P107 P111, P109 P113), (2) for constraint-based requirements, Safety-Critical Systems (P27 P27, P72 P72, P97 P101), Automotive Systems (P59 P59), Event-driven Systems (P149 P154), and Complex Dynamic Systems (P88 P91).

Table 10. Cross Distribution of Requirements Format and Target Software

Target Software	Model-based	Textual	Constraint-based	Formal	Other	Tabular (Matrix-based)
General Software	75	34	4	4	3	6
Embedded System	7	2	3	1	2	1
Web Services	3	5	1	0	0	0
Safety-Critical	1	4	1	3	0	0
Timed Data-flow	4	0	0	0	0	0
Reactive	2	0	1	0	1	0
Real-time Embedded	2	1	0	0	0	0
Product Line	2	0	0	0	0	0
Object-Oriented	1	2	0	0	0	0
Telecom	1	1	0	0	0	0
Automotive	1	0	1	1	0	0

5.2.5. Findings: Trend of Requirements Input Over the Years

With the advancement of requirements engineering research, an increasing number of requirements specification methods have emerged over the past decade [62,63]. In this section, we analyze the trend of requirements format preferences over time, as illustrated in Figure 13. Specifically, before 2008, although textual requirements were already widely employed in the REDAST methodology, their proportion did not significantly dominate among the six requirements formats. This observation aligns with the study distribution discussed in Section 5.1. After 2008, textual requirements gradually became the preferred choice for requirements specification. Furthermore, as indicated by the increasing trend in the "other" requirements format, we observed a growing adoption of diverse requirements formats in recent years. This trend suggests an increasing diversification in requirements selection over time.

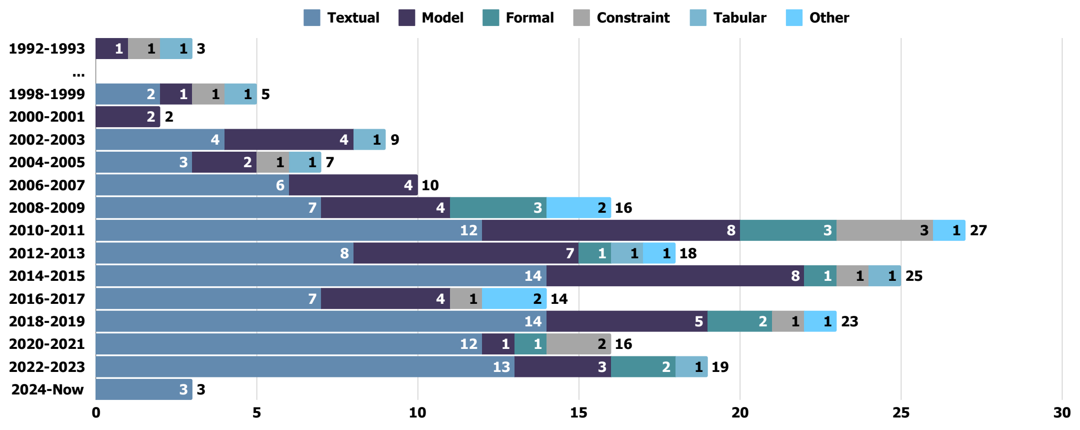


Figure 13. Trend of Requirements Input by Years

[mybox, title=RQ1 Key Takeaways] • Textual specifications are the most prevalent format for REDAST studies and dominate the test artifact generation process in general-purpose software systems. Structured NL (e.g., CNL, RUCM) are preferred over unstructured NL, as they offer a balance between readability and automation in test artifact generation.

• Model-based, formal or constraint specifications are more structured and preferred in embedded, safety-critical, and real-time domains.

• Most studies focus on functional requirements, with only a few addressing non-functional requirements (e.g., performance, security, safety).

5.3. RQ2: Transformation Technology in REDAST

Transformation technology, as discussed in Section 4, is a key component of the REDAST process. After reviewing the selected papers, we present the results of transformation technology in four categories: technique type, framework structure, intermediate representation, and additional input. Finally, to better explain our design of these categories, we provide several examples to interpret our analysis process of RQ2.

5.3.1. Transformation Techniques

This section maps the selected studies into high-level categories based on their adopted transformation techniques. However, we found that the transformation techniques vary widely, and it is hard to initialize one feasible category schema to cover all the selected studies due to the large number of papers. Thus, we classified the 156 papers into five categories, including rule-based, meta-model based, NLP pipeline based, graph/search-based, and ML-based techniques, as we explain later in this subsection. The results are illustrated in Table 11. We adopted high-level categories, however, several studies still overlap in terms of their transformation techniques. For example, P1 P1, P3 P3, and P4 P4 both cover the model-based and rule-based techniques. From the results, we found rule-based techniques are the most commonly adopted in REDAST studies (122 studies), followed by metamodel-based techniques (102 studies), NLP-pipeline-based techniques (51 studies), search/graph-based techniques (38 studies), and ML-based techniques (11 studies). Below we explain these categories with examples.

Table 11. Transformation Techniques of Selected Studies (RQ2)

Techniques	**Paper IDs**	**Num.**
Rule-based	P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P14, P16, P17, P18, P19, P21, P22, P26, P28, P31, P32, P33, P34, P35, P36, P38, P40, P41, P42, P43, P44, P45, P46, P47, P48, P49, P51, P52, P55, P56, P57, P59, P60, P61, P64, P65, P66, P67, P68, P69, P70, P71, P72, P73, P74, P75, P76, P77, P78, P80, P83, P84, P86, P88, P90, P91, P92, P95, P96, P99, P100, P101, P102, P103, P104, P106, P107, P108, P109, P111, P112, P113, P114, P115, P116, P117, P118, P119, P121, P123, P124, P125, P126, P127, P129, P131, P132, P133, P134, P135, P136, P137, P138, P140, P141, P142, P143, P144, P145, P147, P148, P149, P150, P151, P154, P156, P157, P158, P161	122
MetaModel-based	P1, P3, P4, P5, P6, P7, P8, P9, P11, P13, P14, P15, P16, P20, P21, P22, P24, P25, P27, P28, P29, P30, P31, P32, P33, P35, P36, P37, P39, P43, P44, P45, P46, P47, P48, P49, P50, P51, P53, P54, P56, P57, P58, P59, P60, P61, P62, P64, P65, P66, P69, P70, P71, P72, P74, P75, P76, P77, P78, P79, P80, P88, P90, P91, P92, P94, P95, P96, P99, P100, P101, P103, P104, P105, P108, P109, P110, P114, P115, P116, P121, P123, P129, P131, P132, P133, P134, P135, P136, P137, P140, P143, P144, P146, P147, P148, P149, P150, P152, P154, P155, P161	102
NLP-Pipeline-based	P2, P9, P18, P19, P22, P23, P24, P28, P32, P36, P41, P42, P43, P44, P45, P48, P52, P55, P57, P60, P62, P63, P65, P68, P70, P71, P76, P77, P80, P82, P83, P98, P106, P108, P115, P116, P117, P122, P126, P127, P130, P132, P133, P134, P135, P137, P139, P156, P157, P159, P160	51
Graph-based	P6, P8, P12, P13, P17, P19, P25, P26, P28, P33, P42, P47, P50, P53, P57, P59, P61, P65, P66, P70, P71, P74, P77, P78, P86, P88, P89, P92, P98, P103, P116, P121, P127, P136, P139, P145, P153, P161	38
Machine Learning-based	P18, P23, P52, P93, P98, P122, P126, P127, P128, P159, P160	11

Table 12. Rule-based Methods in Selected Studies (RQ2)

Rule Methods	Paper IDs	Num.
Constraint	P1, P2, P3, P4, P9, P10, P12, P16, P17, P18, P21, P22, P26, P28, P31, P32, P33, P35, P36, P38, P41, P42, P43, P44, P45, P46, P47, P48, P49, P51, P55, P60, P65, P66, P67, P69, P71, P73, P75, P77, P83, P84, P92, P96, P99, P108, P109, P111, P112, P113, P115, P116, P117, P123, P129, P135, P136, P137, P141, P142, P143, P150, P151, P154, P156, P158	68
Condition	P3, P4, P6, P7, P8, P11, P14, P16, P28, P32, P34, P40, P45, P47, P51, P52, P56, P57, P59, P61, P64, P68, P69, P70, P72, P73, P76, P78, P80, P86, P88, P90, P91, P95, P101, P102, P104, P106, P107, P111, P114, P118, P119, P124, P125, P126, P127, P131, P132, P133, P134, P137, P138, P140, P141, P142, P144, P145, P147, P148, P149, P156, P161	61
Template	P5, P13, P19, P43, P48, P74, P99, P100, P103, P106, P108, P109, P112, P116, P119, P121, P123, P145, P157	19

Rule-based techniques can be broadly classified into three categories: condition, rule, and template, based on our findings. Specifically,

- *Template-based techniques* formulate the generated artifacts into predefined templates for further processing.
- *Constraint-based techniques* indicate the transforming process for new artifact generation, where the rule here refers to the transformation rules.
- *Condition-based techniques* are defined for the static regulating within existing generated artifacts based on predefined conditions or forms.

For example, P95 P99 is a template-based study that introduces a behavior test pattern (template) from requirements transformation. Thus, we categorize this study into template-based techniques. In another example in P114 P118, this study generally applies static assertion analysis on requirements properties. We categorize this study as condition-based because it doesn't involve a transformation process. We present the results of the rule-based technique in Table 12.

Table 13. Model-based Methods in Selected Studies (RQ2)

MetaModel Methods	Paper IDs	Num.
Other	P1, P3, P11, P15, P25, P28, P29, P32, P37, P39, P43, P46, P48, P50, P54, P57, P58, P60, P61, P64, P70, P71, P74, P75, P76, P77, P80, P90, P91, P92, P94, P95, P100, P101, P103, P104, P116, P137, P140, P146, P148, P155	44
Diagram-based Models	P8, P13, P14, P16, P20, P22, P33, P47, P49, P53, P56, P65, P66, P72, P99, P105, P108, P109, P110, P115, P121, P123, P132, P135, P150, P152	25
Formal and Logic-based Models	P45, P62, P69, P78, P79, P96, P129, P131, P133, P134, P147	11
Domain-Specific Modeling Languages	P27, P30, P31, P51, P59, P114, P161	7
State-based and Transition Models	P36, P44, P88, P143, P149, P154	6
Use Case Models	P4, P5, P6, P7, P24	5
Behavioral Models	P35, P136, P144	3
Ontology and Knowledge-based Models	P9, P21	2
Graph and Flow-based Models	P140	1

Meta-Model-based technologies describe system attributes, user behavior, or event situations in RE. REDAST studies employ metamodels in the transformation process to describe the system information inclusively, which provides a more comprehensive view of test generation. In the selected studies, we introduce nine categories, and the "other" category classifies metamodel-based techniques. Here, we illustrate the distribution of the metamodel-based technique in Table 13.

Table 14. NLP-Pipeline-based Methods in Selected Studies (RQ2)

NLP-Pipeline Method	Paper IDs	Num.
Dependency Parsing	P22, P23, P36, P41, P42, P43, P44, P55, P57, P70, P71, P77, P98, P106, P108, P116, P117, P127, P130, P135, P159	21
POS Tagging	P22, P24, P32, P36, P41, P43, P48, P52, P60, P68, P70, P77, P106, P116, P122, P133, P134, P135, P139, P157, P159	21
NL Parsing	P19, P44, P48, P55, P60, P70, P71, P76, P115, P126, P127, P130, P139, P157	14
Tokenization	P18, P23, P24, P36, P43, P48, P83, P98, P116, P122, P127, P135, P157, P160	14
CNL Parsing	P2, P9, P28, P42, P45, P62, P65, P80, P132, P134, P137, P156	12
Semantic Analysing	P24, P60, P63, P77, P82, P83, P108, P117, P126	9
Sentence Splitting	P43, P71, P115, P126	4
Condition Detector	P23	1
Lemmatization	P135	1
Word Embedding	P127	1
Word Frequency Analysing	P52	1

NLP-Pipeline-based techniques generally employ open-source NLP toolkits, such as NLTK and Stanford coreNLP toolkits, and so on. We observed that POS Tagging (e.g., P22 P22, P70 P70, P128 P133, etc.), Dependency Parsing (e.g., P41 P41, P112 P116, P125 P130, etc.), and Tokenization (e.g., P18 P18, P82 P83, P155 P160, etc.) are frequently adopted in REDAST studies. We illustrate the details of the NLP-Pipeline-based techniques in Table 14.

Table 15. Graph-based Method Adoption in Selected Studies (RQ2)

Graph-based Methods	Paper IDs	Num.
Depth First Traversal (DFT)	P6, P8, P12, P33, P50, P53, P59, P66, P78, P103, P121, P127, P136	13
Breadth-First Traversal (BFT)	P13, P28, P74, P77, P78, P92, P103, P127, P145	9
Graph Traversal	P19, P70, P86, P88, P116, P153, P161	7
Knowledge Graph (KG)	P98, P127, P139	3
Petri Nets	P47, P89, P65	3
Graph Splitting	P71	1
Graph-Theoretical Clustering	P71	1
Greedy Search Strategy	P26	1
Meta-Heuristic Search Algorithm	P61	1
Path Sensitization Algorithm	P57	1
Round-Strip Strategy	P25	1
Shortest Path Finding Strategy	P42	1
Graph Simplifying	P17	1

Graph-based techniques use graphs or diagrams, focused on describing the system behavior. However, the transformation among the other specifications and diagrams is challenging. The existing REDAST studies introduce the search within the graph-based technique to bridge the gap between sequential description and the specification diagram by finding a route or path. For example, in P50 P50, the depth-first traversal algorithm is employed to find the test path. Our findings show that even some traditional graph traversal techniques are still effective in test generation, such as Breadth-First Traversal (BFT), Depth-First Traversal (DFT), etc.

Table 16. Machine-Learning-based Methods in Selected Studies (RQ2)

Machine Learning Methods	Paper IDs	Num.
BERT & Classifier	P23, P98, P128 (Seq2Seq)	3
SyntaxNet	P93, P126	2
Pretrained NER Model	P52, P127	2
MLP Classifier	P18	1
k-Means Clustering	P159	1
LLM & RAG	P160	1

Machine Learning based techniques were not as prevalent as others in REDAST studies. 11 studies opted for ML techniques, including Pretained LMs (e.g., BERT in P23 P23, P94 P98, P123 P128), traditional machine learning algorithms (e.g., P154 P159), SyntaxNet (e.g., P90 P93, P121 P126), and so on, which didn't reflect any trend in the technique adoption.

5.3.2. Framework Details

In this subsection, we cover the remaining parts of the transformation technology, i.e., additional output, intermediate representation and the framework structure.

Table 17. Additional Inputs of Selected Studies (RQ2)

Additional Input Types	Paper IDs	Num.
Source Code	P82, P159	2
System Implementation	P69, P112	2
System Implementation and Supporting Documents	P113, P129	2
Historical Documents	P32	1
Requirements Documents and Historical Documents	P127	1
Requirements Documents and Scenario	P96	1

Additional Input. While our SLR studies requirements-driven automated test generation, the inputs are not constrained to the requirements specification only. We found that some studies introduce additional docs as input to improve the functional coverage of their methodologies. This section illustrates the details of additional input in REDAST studies, including:

- *Historical Documents* are always referenced in the generation process. For example, P2 P2 opted for historical test logs as the additional input in the test generation step. The test logs can serve as a reference for evaluating the generated test artifacts.
- *System Implementation* is the next stage after requirements engineering based on the SDLC, where the requirements are believed to dominate the implementation process and vice versa significantly. In P109 P113, the implementation documents are used in the analysis to provide evidence from the actual scenario.
- *Source Code* can also be used as an additional input alongside requirements in the transformation process. P154 P159 introduces code updating information in the test prioritization process, which is used to find the error-prone modules.
- *Additional Documents*, such as ground knowledge documents, are used to support test generation with a more substantial knowledge base. P122 P127 is a KnowledgeGraph-based method, where the ground knowledge is largely integrated for constructing the knowledge graph.

Table 18. Framework Structure Results of Selected Studies (RQ2)

Structure	Paper IDs	Num.
Sequential	P1, P2, P5, P6, P8, P11, P12, P13, P14, P16, P17, P20, P22, P24, P26, P27, P29, P30, P31, P32, P33, P35, P36, P37, P40, P43, P44, P45, P46, P50, P53, P54, P55, P58, P59, P61, P62, P64, P65, P70, P75, P76, P77, P78, P100, P101, P102, P107, P108, P109, P110, P111, P114, P115, P116, P118, P121, P124, P125, P127, P128, P129, P132, P133, P134, P135, P136, P137, P139, P142, P143, P145, P146, P147, P148, P149, P151, P154, P156, P159	80
Parallel	P3, P7, P9, P10, P19, P21, P25, P34, P39, P51, P52, P57, P66, P68, P72, P74, P80, P104, P140, P150, P152, P153	22
Conditional	P15, P18, P28, P47, P49, P71, P73, P144, P155	9
Loop	P4, P56, P113	3

Framework Structure. From requirements specification to the generation of test artifacts, various designs can be applied to transformation methodologies. Within the selected REDAST framework, we categorize these methodologies into four distinct structure types:

- *Sequential Structure.* The sequential framework conducts transformations in a strict, ordered sequence without any bypasses or shortcuts. This approach enhances logical continuity and maintains a clear connection between each step. P107 P111 is a typical sequential framework where the formal specifications are step-by-step transformed into conjunctive normal form, assignment, and test cases.

- *Conditional Structure.* This framework introduces conditional steps, allowing for alternative paths at key stages. This flexibility improves adaptability and generalization, enabling the framework to manage diverse scenarios effectively. P73 P73 is a good example in this category, where this study constructs several conditions in transformation, e.g., "Need more details of requirements", "There are improvements of transformation", etc.
- *Parallel Structure.* In a parallel framework, different representations can be processed simultaneously across multiple transformation processors. This structure significantly boosts time efficiency. The typical parallel structure can be found in P10 P10. This study constructs a two-way structure and converts requirements input to use case diagrams and executable contracts for generating contractual use case scenarios.
- *Loop structure.* The loop structure incorporates assertion-controlled loops, enabling iterative refinement of generated artifacts. Cycling through iterations ensures higher quality in the final outputs. For example, P109 P113 introduces the loop structure by designing a validation and tuning process to refine the guarded assertions in this method iteratively.

5.3.3. Intermediate Representation

The intermediate representation functions as a detailed explanation of requirements or system structure, reflecting the framework structure's complexity. Specifically, intermediate representation is the step-generated artifacts during the transformation process. When reviewing the selected studies, the intermediate representation necessarily exists in a complex REDAST framework to enable a stepwise transformation. Thus, we illustrate the details of the adopted intermediate representations to understand the framework's composition better.

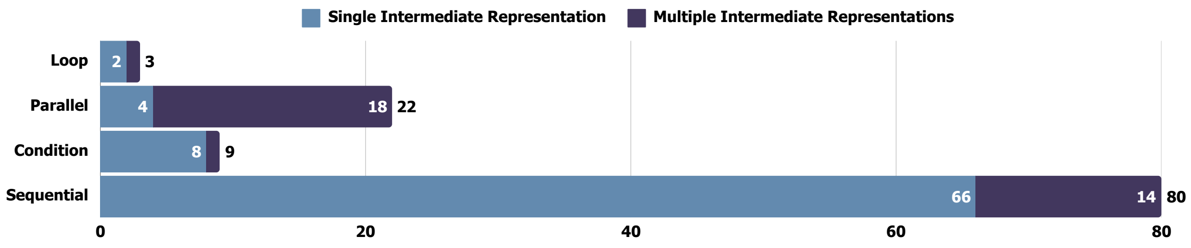


Figure 14. Illustration of Joint Distribution of Intermediate Representation and Framework Structure (RQ2)

Number of Intermediate Representations. The number of intermediate representations is a basic feature of REDAST's framework. In order to improve its explainability, we jointly illustrate the number of intermediate representations with framework structure in Figure 14. The results show different trends in "single" and "multiple" categories, where (1) sequential structure (66 studies) is the dominantly common framework in the "single" category, followed by conditional structure (8 studies), parallel structure (4 studies), and loop structure (2 studies), (2) parallel structure is the most common framework in "multiple" category (18 studies), where sequential is also popular (14 studies).

Table 19. Type of Intermediate Representation in Selected Studies (RQ2)

Repre. Types	Paper IDs	Num.
Rule	P2, P3, P7, P9, P10, P19, P22, P24, P26, P28, P34, P36, P39, P43, P49, P50, P51, P55, P55, P57, P58, P64, P66, P68, P68, P71, P71, P76, P78, P80, P101, P102, P104, P107, P109, P110, P111, P113, P113, P114, P118, P121, P125, P129, P133, P134, P136, P140, P142, P143, P146, P156, P159	51
Model	P1, P4, P7, P8, P11, P15, P18, P20, P21, P25, P25, P27, P30, P31, P32, P33, P40, P43, P45, P47, P50, P51, P52, P58, P59, P61, P62, P65, P70, P72, P73, P74, P74, P75, P76, P80, P100, P104, P114, P124, P132, P133, P137, P137, P145, P147, P148, P150, P152, P154, P155	48
Graph	P3, P9, P10, P12, P13, P14, P16, P17, P19, P30, P33, P35, P37, P44, P46, P53, P54, P56, P57, P66, P77, P108, P109, P110, P115, P116, P127, P128, P135, P136, P139, P140, P143, P144, P149, P150, P152, P153, P153	38
Test Case	P5, P6, P21, P29, P151	5

Type of Intermediate Representations. Considering the intermediate representation generally extended from requirements, we also introduce a similar category used in the requirements specifications, which is illustrated in Table 19. The type category includes:

- *Rule-based Representation* (51 studies): The rule-based representation here refers to general controlled NL, assertion, or equation, where these notations generally consist of descriptions with predefined conditions or forms.
- *MetaModel-based Representation* (48 studies): MetaModel has widely opted for intermediate representation, where model attributes offer additional explainability for test transformation.
- *Graph-based Representation* (38 studies): Graph is an advanced representation method that reflects basic information and indicates the co-relations among different elements.
- *Test-Specification-based* (5 studies): Some studies introduce test-related intermediate representation but cannot classify it into parts of the test outcome. Thus, the test-specification-based representation is especially considered a category. We didn't identify too many test-specification-based representations in selected studies.

5.3.4. Findings: Trend of Transformation Techniques Over the Years

The transformation techniques in REDAST have largely been influenced by advancements in other fields, such as machine learning and deep learning. To illustrate the evolution of these techniques, we integrate publication year data with RQ2 to analyze trends in REDAST studies in Figure 15. Our findings indicate that rule-based, graph-based, and metamodel-based techniques initially dominated transformation approaches. However, following the introduction of the NLP pipeline in 2004 and machine learning techniques in 2012, graph-based approaches gradually declined in popularity. Additionally, we observed an increasing diversity in transformation techniques over time, reflecting a broader range of methodologies being adopted in REDAST.

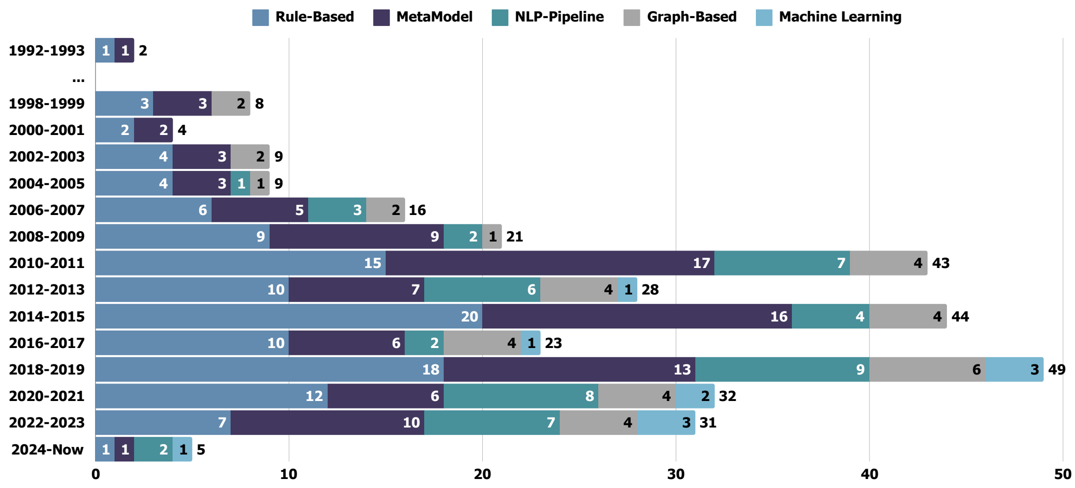


Figure 15. Trend of Transformation Techniques by Years

5.3.5. Findings: Cross-Analysis of Requirements Input and Transformation Techniques

During the transformation process, the requirements input is further processed to the intermediate representations or end test artifacts. Thus, we cross-discuss the relations between requirements input and transformation techniques, which are depicted in Figure 16. The results are not surprising that (1) rule-based techniques are still commonly opted for parsing different requirements input, 36%, 39%, 40%, 39%, and 33%, respectively, in textual, model-based, formal, constraint-based, and tabular categories, (2) metamodel-based transformation techniques are not only applicable for model-based requirements but also for the other requirements expressions, 27%, 41%, 50%, 47%, and 33%, respectively, in textual, model-based, formal, constraint-based, and tabular categories, which suggests that meta-models are flexible and applicable to both textual and structured requirement representations. As for NLP-pipeline-based approaches, they are almost exclusively applied to textual requirements. Surprisingly, we found that, in the model-based categories, there are still five papers that introduce NLP-Pipeline-based techniques in their methodologies. P28 P28 introduced a sequence-diagram-based use case, which both enables the flexibility of textual requirements and the structural ability of model-based requirements, where the NLP-pipeline is introduced to parse the CNL or structured expression. Similarly, the NLP pipeline in P57 P57 parsing the dependency in the introduced UML-based use case. Thus, the selection bias of NLP pipelines can be avoided by combining textual requirements with the other requirements specifications, which can also additionally provide flexibility from textual requirements for the method.

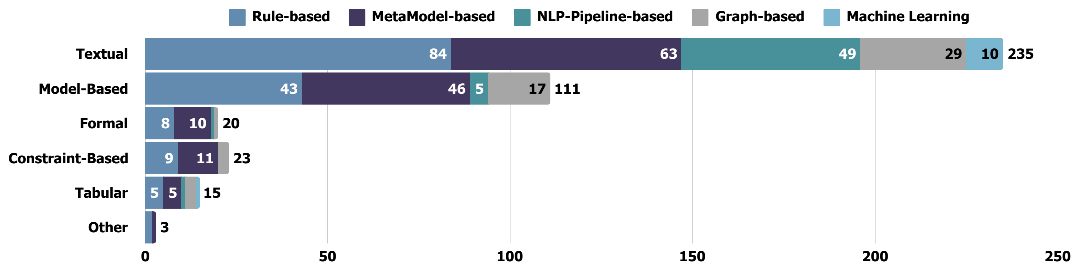


Figure 16. Cross-Distribution of Requirements Input and Transformation Techniques

[mybox, title=RQ2 Key Takeaways] • Rule-based and model-based approaches dominate REDAST due to their structured, interpretable nature and low dependency on training data. While AI techniques—particularly NLP—are increasingly incorporated for automation, they primarily support rule-based and model-driven frameworks rather than serving as standalone transformation methods.

• Requirements are generally considered sufficient input for REDAST, as few studies integrate additional data sources (e.g., system implementation, source code, or historical test logs). Instead, intermediate representations serve as essential enablers for translating requirements into structured test artifacts. A significant number of studies employ multi-step transformations, making intermediate representations a key bridge between raw requirements and test automation artifacts.

5.4. RQ3: Generated Test Artifacts in REDAST

RQ3 aims to discuss the final outcomes of the REDAST process, where we studied the test abstraction level, test type, test notation, and test coverage.

5.4.1. Test Abstraction Level

Varying on different usage scenarios, the test artifacts could be described at various levels. For example, when we plan to verify user behaviors, the abstract test scenario is one of the best choices for engineers to check the correctness of each step. Thus, we introduce three categories for test abstraction level classification, including executable, abstract, and report. The idea of each category is described as:

- *Executable Level* includes artifacts, such as code and scripts, that can be directly executed. Typically, the test artifact in P94 P98 is on the executable level, where the test case consists of several sections, including "Target Entities", "Test Intent", "Extracted Triplets", "Context Sub-graph", "Test Case", and so on.
- *Abstract Level* includes the artifacts that cannot be directly executed, e.g., textual scenarios, test diagrams, etc. We identify P71 P71 as an example of this category. P71 generates test plans consisting of activities and acceptance criteria, where the acceptance criteria in this study don't give specific operation or system behavior, e.g., "Is bill paid?", "Is ID card valid?" or some similar statements.
- *Report Level* mainly refers to the results after executing test artifacts. We separately present this because some studies also provide the executing tool. In P29 P29, this study introduces an automatic tester using generated test data. The final output is the corresponding test report from the tester.

Table 20. Test Abstraction Level of REDAST Studies (RQ3)

Abstraction Levels	Paper ID	Num.
Executable	P1, P2, P3, P4, P5, P7, P8, P9, P11, P12, P13, P15, P16, P19, P21, P23, P24, P25, P26, P27, P28, P29, P30, P31, P32, P33, P34, P35, P36, P37, P38, P39, P40, P43, P44, P45, P48, P49, P50, P51, P52, P53, P54, P55, P56, P57, P58, P59, P60, P61, P62, P63, P64, P65, P66, P67, P68, P70, P73, P76, P77, P78, P79, P80, P82, P83, P84, P86, P88, P90, P91, P92, P94, P95, P96, P98, P99, P100, P101, P103, P104, P105, P106, P107, P108, P109, P110, P111, P112, P113, P114, P116, P118, P119, P121, P123, P124, P125, P127, P128, P129, P130, P131, P132, P133, P134, P135, P136, P138, P140, P141, P142, P143, P144, P145, P146, P147, P148, P149, P151, P152, P153, P155, P156, P157, P158, P159, P160, P161	129
Abstract	P5, P6, P7, P10, P14, P17, P18, P20, P22, P26, P41, P42, P46, P47, P69, P71, P72, P73, P74, P75, P80, P84, P89, P91, P93, P102, P115, P117, P122, P125, P126, P137, P139, P140, P147, P150, P154	37
Report	P15, P29, P57, P89, P149	5

5.4.2. Test Formats

In this section, we present detailed information about the generated test artifacts, adopting the categories of test type and notation to describe them in detail. While the most commonly used test types are test cases and scenarios, we also found some task-specific test artifacts. The results of the test type are illustrated in Table 21.

Table 21. Test Formats of REDAST Studies (RQ3)

Test Formats	Paper IDs	Num.
Test Case	P1,P2,P3,P4,P5,P6,P7,P9,P11 , P13 (optimized), P15,P16,P19,P21,P23,P24,P25,P26,P27,P28,P30 , P31 (acceptance), P32,P35,P36,P37 , P38 (prioritized), P39 (prioritized), P43 , P44 (system and acceptance), P45,P48,P49,P50,P51 , P52 (tabular), P54,P55,P56,P57, P58 (failure-revealing), P59,P60, P61,P62,P68, P70,P72,P74,P76,P77,P79,P80 , P82 (prioritized), P83 (security), P84 , P86,P88,P90,P91,P92,P94,P95,P96,P98,P99,P100,P101,P102,P103 , P104 (prioritized), P105,P106,P107,P108,P109,P110,P111,P112 , P113 (passive), P114,P116 , P118 (prioritized), P119 , P121 , P122, P123,P124,P125,P127,P128,P130,P131,P132,P133,P134,P135 , P136 (prioritized), P138 (prioritized), P140 , P141 (prioritized), P142 (prioritized), P143 , P144 (prioritized), P145 , P146 (prioritized), P147,P148,P149,P151,P153,P156,P157 , P158 (prioritized), P159 (prioritized), P161	116
Test Scenario	P8,P11,P12,P28,P29,P31,P32,P33,P51, P53 (prioritized), P56,P57,P62,P64,P65, P66 (prioritized), P67 (prioritized), P73,P95,P96,P129,P145,P160,P161	24
Other	P14 (Test-Path), P20 (Test Procedures), P21 (Test Mutant), P26 (Scenario Tree), P34 (Test Verdict), P41 (Key Value Pairs), P47 (Test Requirements), P71 (Test Plan, Acceptance Criteria), P75 (Test Description), P112 (Interface Prototype), P115 (Test Bench), P122 (Test Suggestion), P140 (Function Chart), P154 (Safety Properties)	14
Test Model	P18, P22, P73, P80,P137, P150	6
Test Report	P15, P29, P57, P89, P149	5
Test Suite	P40, P79, P95, P152, P155	5
Test Oracle	P7, P34, P63, P91	4
Test Sequence	P7, P17, P69, P78	4
Test Guidance	P93, P117, P126	3
Test Specification	P5, P6, P20	3
Test Script	P2, P51, P75	3
Test Graph	P42, P125, P139	3
Test Goal	P10, P46	2
Traceability Matrix	P84, P147	2

We first illustrate the categorization results of test formats. In this section, the test format generally refers to the higher dimension of generated test artifacts. We will not use the exact notation in this part. The detailed results are in Table 21, which based on the following classes:

- *Test Case, Test Scenario, and their Variations* are the most commonly derived test formats in REDAST papers. Test scenarios enable a high-level description of the test objectives from a comprehensive point of view. Test cases include more detailed, stepwise instructions or definitions by focusing on a specific software part.
- *Test Requirements, Guidance, Plan, Suggestion, and Acceptance Criteria*. These test artifacts generally offer high-level, abstracted, or constructive objectives and suggestions for software testing. Rather than specifically match every step in software testing, they enable high-level instruction for test structure.
- *Test Suite, Script, and Oracle*, compared with the other formats, are advanced in applicability and usability. Generally, assertion, code, or any executable source are used in these formats, which are designed for execution.
- *Test Sequence, Model and Goal*. These test formats are designed to meet the specific test objectives in the structural testing process. The model or sequence in these formats enables better traceability compared with the other formats.

In the results, we can find that the test case is the most adopted test format in the selected studies; test scenarios and test reports are the second most studied categories, followed by "other" formats, test reports, test models, and so on. However, we found that we identified some test artifacts that are hard to automate, such as test oracles, test goals, and traceability matrix. As the typical test artifacts, test oracle is well-known for its difficulty in determining the correctness based on the given input and the complexity of software systems [64–66], which is believed that the human invention is still needed [67]. In REDAST studies, the automation of the test oracle is realized with the help of precise requirements specification, wherein the two papers under the test oracle category, (1) P7 P7 ensure the correctness of the test oracle by preliminarily checking the correctness of requirements, then from the derived test cases to formulate the test oracles, (2) P63 P63, by converting RUCM (Restricted Use Case Model) to OCL expression, the given requirements can be details checked for the generation of test oracle. This suggests that, by incorporating appropriate requirements specifications in the REDAST process, correctness and completeness can be assured. Furthermore, the preliminary correctness checking can reduce the effort to post-check the correctness of test artifacts.

5.4.3. Test Coverage Methods

Test coverage is defined as whether our test cases cover the testing objectives [68]. However, there are various methods available for this purpose. We design our categorization method for the coverage types in the selected studies and finalize our results in Table 22.

Table 22. Test Coverage of REDAST Studies (RQ3)

Test Coverages	Paper IDs	Num.
Requirements Coverage	P2, P16, P22, P26, P33, P38, P39, P41, P42, P45, P49, P60, P63, P71, P72, P74, P75, P82, P84, P91, P93, P95, P98, P102, P104, P105, P106, P108, P109, P112, P113, P115, P117, P118, P121, P123, P124, P125, P127, P133, P136, P138, P141, P145, P146, P153, P155, P156, P158, P159, P161	52
Behavioral/Scenario Coverage	P1, P4, P6, P7, P11, P24, P32, P37, P46, P51, P52, P53, P54, P56, P57, P58, P62, P64, P66, P67, P68, P70, P73, P77, P78, P89, P92, P96, P111, P114, P129, P130, P144, P148, P150, P151, P154, P157, P160	37
Path Coverage	P5, P8, P12, P14, P17, P30, P35, P55, P59, P65, P88, P103, P110, P135, P143, P152	16
Functional Coverage	P13, P25, P27, P29, P61, P94, P107, P119, P122, P131, P134	11
Use Case Coverage	P9, P10, P15, P19, P43, P47, P76, P83, P86	9
Not Specified	P21, P34, P48, P80, P90, P126, P137, P139, P149	9
Decision/Branch Coverage	P50, P100, P101, P132, P140, P147	6
Statement Coverage	P23, P28, P36, P128, P142	5
Other	P116 (Boundary Coverage), P44 (Combinatorial Coverage), P18 (Structural Coverage)	3
Model-Based Coverage	P20, P69, P79	3
Rule-Based Coverage	P3, P31	2
Code Coverage	P40, P99	2

We found that requirements coverage is the most commonly used method in REDAST studies, followed by Behavioral/Scenario Coverage, Path Coverage, Functional Coverage, and so on. From the general point of view, code coverage is the most commonly adopted coverage method in automated test generation. The code coverage can simply and effectively assess the quality of generated test artifacts and measure which code are being covered in the generation [69,70]. However, the requirements coverage is leading the trend of test coverage in REDAST, where, moreover, the behavioral/scenario, use case, statement, and decision/branch coverage are also able to be classified into requirements coverage. We believe that the requirements can serve a similar role to code. By referring to different parts of requirements, such as behavior, path, or statement, the test artifacts are also able to be assessed and measured on the requirement level.

5.4.4. Findings: Trend of Test Abstraction Level Over the Years

Test artifacts are the closest step in REDAST, where the generated test artifacts are the end outcomes of the framework. Reviewing the results, we can notice that REDAST studies mainly maintain their generated test artifacts on the executable level. To better see the trend, we present the trend of abstraction level by year in Figure 17. Even if abstract and executable test artifacts are both useful for software engineering, the results show that executable test artifacts are now becoming more and more popular after 2007, which means executable test artifacts are more applicable and capable for recent tasks and usage scenarios.

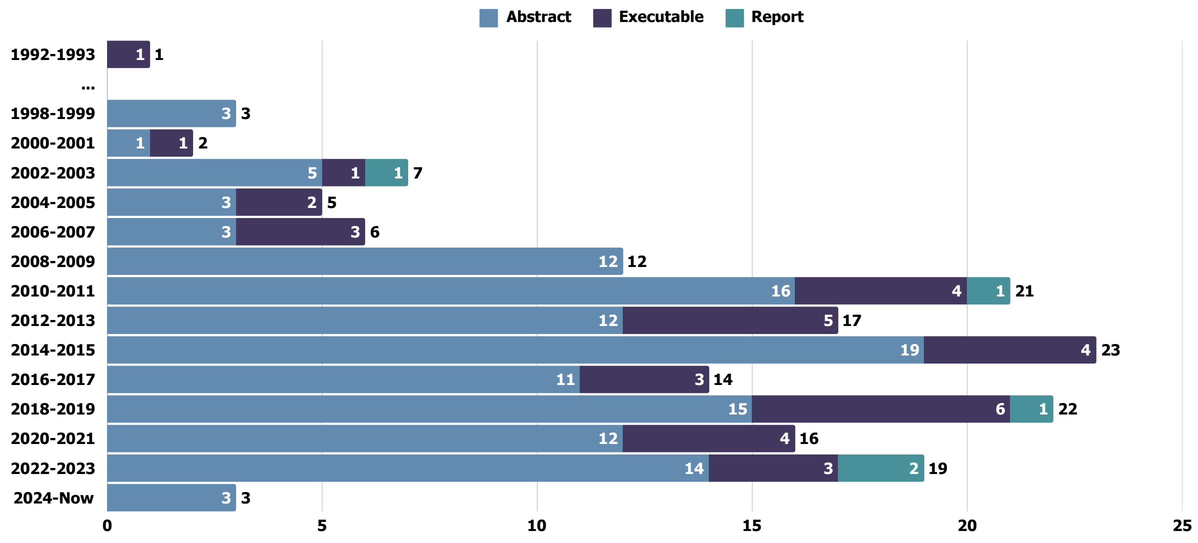


Figure 17. Trend of Test Abstraction Level by Years

5.4.5. Findings: Cross-Analysis of Transformation Techniques and Test Artifacts

Since test artifacts are the direct outcomes of transformation techniques, we aim to explore the connection between these artifacts and the transformation methods used to generate them. Figure 16 presents the cross-distribution of transformation techniques and test formats. Our findings indicate that rule-based, metamodel-based, and NLP-pipeline-based methods are widely adopted across various types of test artifacts. Specifically, (1) Rule-based and metamodel-based techniques are the most versatile and commonly applied across different test outputs; (2) NLP-pipeline-based and graph-based methods are used selectively, with NLP techniques being more influential in generating test cases and test scenarios; (3) Machine learning-based methods are the least utilized, suggesting that machine learning is still relatively underexplored in REDAST studies.

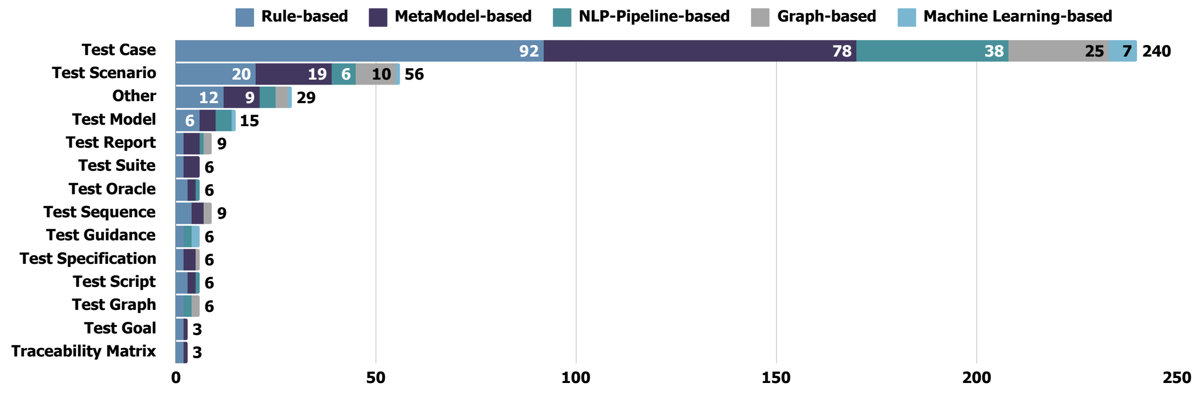


Figure 18. Cross-Distribution of Transformation Techniques and Test Artifacts

5.4.6. Findings: Cross-Analysis of Requirements Input and Test Artifacts

Similarly, we illustrate the stack chart in Figure 19 of requirements input and test outcomes. From the chart, we can find textual requirements are still leading the landscape in all formats for test outcomes, which is also strong proof of the flexibility of textual requirements. The test oracle, sequence, goal, and suite, which are widely believed to be advanced test strategies, we noticed that formal, constraint-based, tabular, and model-based requirements could support the generation of these specialized test artifacts.

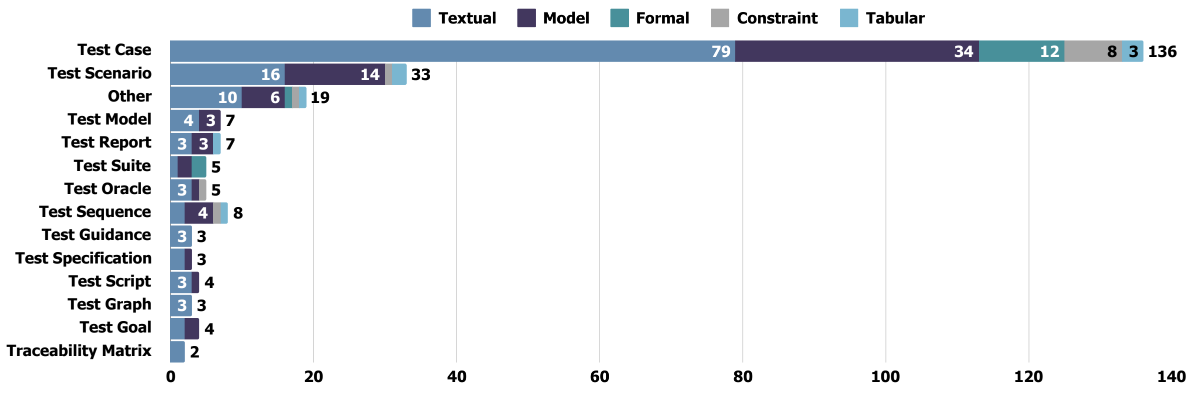


Figure 19. Cross-Distribution of Requirements Input and Test Artifacts

[mybox, breakable, title=RQ3 Key Takeaways] • Executable test artifacts (e.g., scripts, test suites, assertions) are the dominant output in REDAST studies.

• A large majority of the studies focus on concrete test cases as the outcome, followed by test scenarios. Harder-to-automate test artifacts (e.g., test oracles, traceability matrices) remain an open challenge, requiring structured requirements modeling. Traceability from requirements to test cases is considered challenging and a largely unresolved issue.

• Requirements coverage is the leading coverage criterion in REDAST, encompassing functional and behavioral coverage, followed by path coverage.

5.5. *RQ4: Evaluation Methods of REDAST Studies*

REDAST studies aim to generate reliable and usable test artifacts, from requirements to test artifacts. However, how to verify the efficacy of the generated test artifacts still remains in the mist. In this section, we plan to illustrate the related results of the selected studies to systematically discuss the evaluation methodologies for test artifacts. The results in this section consist of three parts: (1) types of evaluation methods, (2) target software platforms in the case studies, and (3) the usability of the studies.

5.5.1. Types of Evaluation Methods

In the selected studies, we identified two primary evaluation methods: (1) case studies and (2) evaluations using given datasets. However, some studies rely on conceptually designed case studies rather than industrial demonstrations based on real-world cases. To provide a clearer classification, we restructured the categories for evaluation methods into three types: conceptual case study, industrial case study, and evaluation using given datasets. The summarized results are presented in Table 23.

Table 23. Benchmark Type Results of Selected Studies (RQ4)

Benchmark Types	**Paper IDs**	**Num.**
Conceptual Case Demonstration	P1, P2, P4, P6, P7, P8, P10, P11, P12, P13, P14, P16, P17, P21, P22, P26, P27, P35, P43, P46, P48, P49, P51, P53, P62, P64, P66, P67, P68, P69, P71, P74, P75, P77, P82, P84, P86, P89, P90, P92, P98, P99, P100, P105, P106, P111, P112, P114, P115, P119, P123, P124, P128, P132, P133, P134, P136, P141, P144, P145, P149, P150, P151, P152, P153, P155, P156, P159, P161	69
Real Case Demonstration	P3, P5, P9, P12, P15, P18, P20, P23, P24, P25, P28, P29, P30, P32, P33, P34, P36, P37, P38, P42, P44, P45, P49, P50, P54, P55, P58, P59, P60, P61, P72, P78, P79, P80, P83, P88, P91, P94, P95, P101, P102, P107, P108, P109, P110, P113, P117, P118, P121, P123, P125, P126, P127, P129, P130, P131, P132, P134, P135, P137, P138, P140, P142, P147, P154, P160	66
NA	P19, P31, P39, P40, P47, P52, P57, P63, P65, P70, P73, P76, P93, P96, P103, P104, P116, P122, P139, P143, P146, P148, P158	23
Dataset Evaluation	P18, P41, P56, P157	4

- **Conceptual Case Demonstration**. These studies typically begin by designing a conceptual case and then applying REDAST methods to the designed case to demonstrate the test generation process. Commonly used conceptual cases include online systems (e.g., P11 P11, P21 P21, P119 P124), ATMs (e.g., P12 P12, P13 P13, P35 P35), and library systems (e.g., P6 P6), among others. While these demonstrations effectively illustrate the methodology procedure, they often lack compulsion and persuasiveness in demonstrating the efficacy of the results. However, the use of conceptual cases, which are based on widely familiar scenarios, enhances understandability. This familiarity benefits readers by making the methodologies easier to comprehend and follow.
- **Industrial Case Demonstration**. Studies in this category demonstrate their methods by incorporating industrial cases into their experiments. By re-organizing and utilizing data extracted from industrial scenarios, REDAST methods are validated under real-world conditions, offering stronger evidence and greater persuasiveness compared to conceptual case studies. Additionally, we observed that some studies intersect across categories, combining (i) both conceptual and industrial case studies, and (ii) industrial case studies with dataset evaluations (e.g., P28 P28 on several industrial cases from public paper, P155 P160 on postal systems). These overlaps occur because industrial cases not only serve as a basis for case studies but can also be used to formulate evaluation datasets, further enhancing their utility in validating methodologies.
- **Evaluation on Given Datasets**. Evaluations conducted on public or industrial datasets provide a compelling approach to demonstrating the efficacy and usability of a method. However, transitioning from discrete cases to datasets requires significant additional effort, including data cleaning, reorganization, and formulation. This challenge arises due to two primary factors: (1) the high usability requirements of REDAST studies, which are often difficult to demonstrate effectively using certain datasets, and (2) the absence of a standardized benchmark dataset for evaluating test artifacts. As a result, most studies rely on case demonstrations rather than systematic evaluations with well-formulated datasets. Only a small number of studies (e.g., P18 P18, P39 P39, P41 P41) employ dataset evaluations within the selected works.

The results indicate that conceptual case studies are the most commonly adopted method in the selected studies, with 69 instances, followed by industrial case studies with 66 instances, and dataset evaluations with 4 instances. Additionally, the "NA" category includes studies that do not provide any solid demonstration methods.

5.5.2. Software Platforms in Case Demonstration

Recognizing that case studies are the primary method of demonstration in REDAST studies, we further analyzed the details of these case studies by categorizing the software platforms used in their demonstrations (Table 24). The adopted software platforms are grouped into 17 categories, including shopping systems, resource management systems, financial systems, and more. These categories encompass the most common usage scenarios and are intended to guide the design of case studies in future research.

Table 24. Software Platforms in Case Study of Selected Studies (RQ4)

Software Platforms	Paper IDs	Num.
NA	P19, P31, P39, P40, P47, P52, P57, P63, P65, P70, P73, P76, P93, P96, P103, P104, P116, P122, P139, P143, P146, P148, P158	23
Automotive System	P5, P23, P24, P32, P45, P58, P59, P61, P79, P80, P98, P102, P107, P108, P115, P127, P128, P129, P154, P155	20
Control System	P27, P29, P30, P42, P44, P60, P64, P78, P94, P111, P113, P121, P125, P132, P140, P142, P144, P161	18
Safety System	P49, P50, P54, P71, P72, P78, P88, P89, P102, P108, P117, P126, P131, P132, P135, P147, P154	17
Order System	P4, P10, P12, P16, P17, P48, P53, P55, P66, P67, P68, P114, P132, P133, P134, P152	16
Business System	P18, P20, P23, P36, P37, P68, P77, P90, P91, P92, P124, P160	12
Resource Management System	P5, P25, P26, P33, P34, P68, P82, P118, P121, P126, P136, P157	12
Workforce System	P8, P22, P23, P28, P41, P46, P74, P75, P86, P95, P99, P145	12
Healthcare System	P15, P17, P34, P38, P41, P51, P58, P83, P110, P119, P159	11
ATM System	P12, P13, P17, P22, P35, P105, P109, P151, P153, P155	10
Aerospace System	P28, P35, P69, P80, P100, P101, P130, P149, P156	9
Banking System	P5, P21, P43, P68, P121	5
Authentication System	P1, P84, P106, P112	4
Mobile Application	P2, P34, P62, P68	4
Library System	P6, P7, P9	3
Booking System	P55, P56, P150	3
Education System	P11, P123, P159	3
Other	P137, P138, P141	3
Database System	P3, P117	2
Examination System	P14	1

The results reveal that, alongside domain-specific systems such as control and automotive systems, order processing systems and ATMs are frequently selected to demonstrate test generation methodologies. These systems are often chosen because they are widely recognized and understood in the public domain, making them effective tools for illustrating methodologies in a comprehensible manner.

5.5.3. Examples of Evaluation or Experiments

A critical aspect of any REDAST approach is the level of detail provided by the papers on the methodology and the evaluation of the approach. In this subsection, we classified the selected papers by separately analyzing the clarity and explanation provided in four key sections: (1) methodology explanation, (2) case examples, (3) experiments, and (4) discussions. Table 25 shows an overview of different evaluation and methodology sections covered in the selected studies. For example, the second row (Case example) lists papers that cover the description of case examples in some level of detail, which are also exemplified below for papers P159 and P24.

Table 25. Usability Results of Selected Studies (RQ4)

Usability	Paper IDs	Num.
The methodology explanation	Almost all of the papers enable clear methodology explanation, except P35, P36, P38, P138, P140, P156, P158	154
Case example	P1, P2, P3, P4, P5, P7, P8, P9, P11, P12, P13, P14, P15, P16, P17, P18, P21, P22, P23, P24, P26, P27, P28, P29, P32, P33, P34, P35, P36, P37, P38, P39, P40, P41, P42, P43, P44, P46, P48, P49, P50, P55, P58, P59, P60, P61, P62, P64, P66, P67, P68, P69, P71, P78, P79, P83, P84, P86, P88, P89, P90, P91, P92, P93, P94, P95, P96, P98, P99, P101, P102, P107, P108, P109, P110, P111, P112, P113, P114, P115, P116, P117, P118, P119, P121, P123, P124, P125, P126, P127, P129, P130, P131, P132, P133, P134, P135, P136, P137, P138, P140, P141, P143, P144, P145, P146, P147, P148, P149, P150, P151, P152, P154, P155, P156, P158, P159, P160, P161	120
Discussion	P1, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12, P13, P15, P17, P18, P19, P20, P21, P22, P23, P24, P25, P26, P27, P28, P29, P31, P32, P33, P35, P36, P37, P38, P40, P41, P42, P43, P44, P45, P46, P47, P48, P49, P50, P51, P52, P54, P55, P56, P58, P59, P60, P61, P62, P63, P64, P65, P66, P67, P68, P69, P70, P71, P72, P73, P75, P76, P77, P78, P79, P80, P82, P83, P84, P86, P88, P92, P94, P96, P99, P104, P113, P117, P121, P127, P128, P129, P130, P131, P132, P134, P135, P137, P141, P144, P147, P152, P154, P155, P156, P157, P158, P159, P160, P161	105
Experiment	P2, P3, P5, P7, P9, P12, P13, P14, P15, P17, P18, P22, P23, P24, P26, P27, P28, P30, P32, P34, P35, P37, P39, P40, P41, P42, P43, P44, P45, P52, P53, P56, P58, P59, P61, P70, P71, P72, P76, P79, P80, P82, P83, P86, P89, P91, P95, P99, P101, P102, P107, P108, P110, P113, P117, P118, P121, P125, P127, P128, P129, P130, P131, P132, P133, P134, P135, P137, P138, P140, P141, P142, P154, P157, P159, P160, P161	77

Case 1: P155 - Generating Test Scenarios from NL Requirements using Retrieval-Augmented LLMs: An Industrial Study

P155 P160 introduces an LLM-driven test scenario generation method. While LLMs offer flexible and powerful natural language generation capabilities, their limited controllability poses challenges for broader applications in REDAST studies. To address this issue and validate the methodology's reliability, P155 incorporates a case study based on an industrial usage scenario. The experiment evaluates performance using both quantitative metrics and qualitative human assessments. Quantitative metrics include standard machine translation evaluation measures such as BLEU, ROUGE, and METEOR. However, the core of the evaluation is a human interview with software engineers, offering deeper insights into the practical utility of the methodology. While metrics provide a partial perspective from an NLP standpoint, the human interviews emphasize the method's effectiveness and applicability in real-world scenarios.

Case 2: P24 - Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach

P24 P24 proposes an NLP-based framework for generating executable test cases. The industrial case-based experiments address three research questions, including evaluations of correctness and manual comparisons. Notably, in the manual comparison, instead of conducting direct human interviews for the generated test artifacts, the study employs a manual comparison with test cases created by domain experts. The results of these comparisons provide a robust demonstration of the generated test cases, as their alignment with established "golden truth" test cases serves as compelling evidence of the methodology's effectiveness.

5.5.4. Findings: Cross-Analysis of Demonstration Types and Target Software Systems

While researchers demonstrate the efficiency of their methods through evaluations or practical demonstrations, the target software system ultimately reflects the primary objective of REDAST studies. Our goal is to establish a connection between these demonstrations and the target software system to provide deeper insights into the preference for demonstration methods across different usage scenarios. From our findings in Figure 20, we observed that conceptual case studies remain the most commonly adopted demonstration method. In contrast, real (industrial) case studies are generally more challenging to conduct due to the difficulty of obtaining real-world data. However, in domains such as web services, safety-critical systems, and objective-oriented systems, real case studies are predominantly chosen. These critical systems often impose stricter pass-rate criteria for test outcomes, necessitating more rigorous validation. We suggest that future researchers pay particular attention to the demonstration aspect when dealing with such systems.

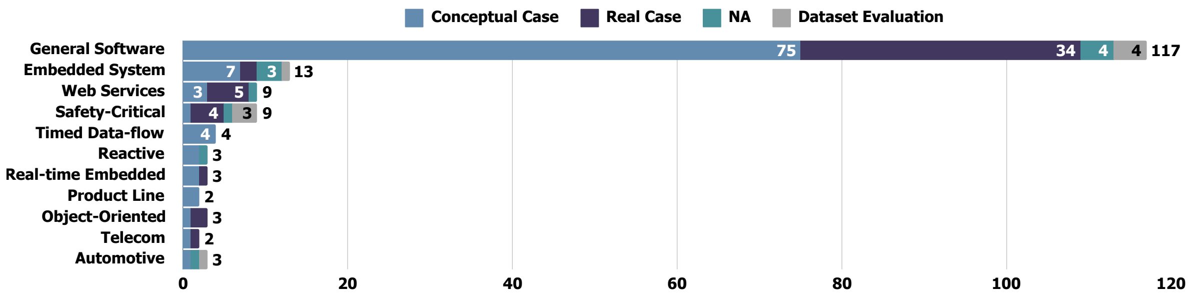


Figure 20. Cross-Distribution of Demonstration Types and Target Software Systems

[mybox, breakable, title=RQ4 Key Takeaways] • Conceptual case studies are the most common evaluation method in REDAST studies. While these (example) case studies enhance understandability, they often lack strong empirical validation compared to real-world industrial cases. Industry case study evaluations are also prominent in ≈40% studies, demonstrating the real-world applicability and need for REDAST approaches.

• There are very few studies conducted on any public datasets. This showcases a gap in the standardized evaluation of REDAST approaches due to the lack of standard benchmarks for requirements-driven test case generation.

5.6. RQ5: Limitation, Challenging and Future of REDAST Studies

5.6.1. Limitations in Selected Studies

In the selected papers, we observed that some studies explicitly discuss their limitations. This section presents the identified limitations, categorized into 14 concise types, as summarized in Table 26. Notably, 28 studies do not explicitly mention any limitations in their content. Consequently, the following discussion primarily focuses on the remaining papers that explicitly acknowledge their limitations.

Table 26. Limitation Results of Selected Studies (RQ5)

Limitations	Paper IDs	Num.
Limitations of Framework Design	P4, P8, P11, P12, P21, P22, P23, P28, P31, P32, P33, P35, P41, P44, P45, P46, P60, P62, P64, P67, P68, P86, P89, P91, P92, P102, P106, P109, P110, P117, P118, P121, P126, P131, P145, P146, P148, P150, P151, P156, P160	41
NA	P14, P15, P16, P18, P26, P29, P83, P90, P93, P95, P98, P99, P101, P114, P119, P123, P124, P125, P133, P134, P137, P138, P139, P140, P141, P142, P158, P161	28
Limitation of Evaluation or Demonstration	P7, P19, P39, P40, P48, P74, P75, P96, P100, P102, P105, P108, P110, P122, P129, P132, P143, P146, P153, P160	20
Scalability to Large Systems	P10, P30, P32, P35, P38, P47, P50, P55, P56, P66, P70, P75, P78, P80, P84, P107, P111, P126, P128, P159	20
Over Relying on Input Quality	P2, P17, P34, P36, P37, P43, P44, P51, P53, P55, P65, P71, P107, P127, P144, P157	16
Complexity of Methodology	P3, P54, P62, P69, P72, P73, P76, P77, P79, P82, P94, P116, P147, P149, P154	15
Automation (Methodology)	P6, P8, P24, P27, P38, P57, P58, P59, P107, P112, P115, P116	12
Incomplete Requirements Coverage	P13, P61, P88, P105, P108, P113, P131, P136, P155, P156	10
Automation (Test Implementation)	P2, P5, P25, P73, P104, P122, P130, P135	8
Automation (Requirements Specification)	P1, P9, P68, P103, P117, P151, P152	7
Requirements Ambiguities	P42, P49, P52	3
Limitation of Implementation	P20, P63	2
Time-Costing	P20, P39	2
Additional Cost of Requirements Specification	P7	1

Table 27. Automation Level of Selected Studies (RQ5)

Automation Levels	Paper IDs	Num.
Highly Automated	P2, P3, P4, P5, P6, P7, P8, P9, P10, P12, P16, P17, P18, P20, P22, P23, P24, P28, P29, P32, P41, P42, P43, P44, P47, P48, P49, P51, P53, P54, P56, P57, P59, P60, P69, P70, P71, P74, P75, P77, P78, P82, P83, P89, P91, P92, P93, P98, P100, P101, P102, P103, P105, P106, P107, P108, P109, P111, P112, P117, P124, P125, P126, P127, P132, P133, P134, P136, P137, P140, P145, P146, P149, P151, P153, P154, P160	77
Semi-Automated	P19, P26, P27, P38, P46, P58, P63, P64, P68, P84, P86, P90, P94, P95, P99, P104, P110, P113, P114, P115, P116, P118, P119, P121, P122, P129, P130, P131, P135, P138, P139, P141, P142, P148, P150, P152, P155, P156, P161	39
Fully Automated	P1, P11, P13, P14, P15, P21, P25, P30, P31, P33, P34, P35, P36, P37, P39, P40, P45, P50, P52, P55, P61, P62, P65, P66, P67, P72, P73, P76, P79, P80, P128, P143, P144, P147, P157	35
Low Automation	P88, P96, P123, P158, P159	5

Automation is a frequently mentioned limitation in the selected studies. We identified that automation limitations vary across three key areas: requirements specification, methodology conduction, and test implementation. To provide a more detailed analysis, we classified these limitations into three categories: Automation (Requirements Specification), Automation (Methodology), and Automation (Test Implementation). Given that the automation level is a critical aspect of REDAST, we present the estimated automation levels for all the selected papers in Table 27, as four levels, Fully automated - End-to-End Automation, highly automated - Automation-Dominant, semi-automated - Automation Supported, and low automated - Minimal Automation. The detailed definitions of automation are depicted in Section 4.6.

Scalability, framework design, incomplete requirements coverage, and requirements ambiguities are commonly identified limitations in the reviewed studies. Specifically, 41 studies reported incompatibilities with handling certain scenarios, 20 studies highlighted challenges in scaling to complex or larger systems, 10 studies acknowledged an inability to cover all requirements during test generation, and 3 studies discussed potential ambiguities in requirements. These limitations often stem from framework structure issues, where the methods fail to comprehensively account for diverse usage scenarios, leading to problems with generalization and applicability. For example, P38 P38 and P48 P48 discuss challenges with generalization due to difficulties in handling complex systems, while P124 P129 highlights performance gaps when dealing with systems of varying sizes. Framework design limitations also constrain methods in specific contexts, as evidenced by P146's P151 discussion of incompatibilities with non-functional requirements and P141's P146 primary focus on extra-functional properties.

Additional limitations arise from the complexity of some framework components, including over-reliance on input quality, methodological complexity, and time inefficiencies. Over-reliance occurs when predefined rules or input formats disproportionately influence the performance of test generation, making the process vulnerable to input quality issues. For instance, P44 P44 notes that dependency relations can affect test generation accuracy, while P103 P107 highlights challenges arising from the conjunctive statement format, which complicates stable test generation. Methodological complexity and time-cost issues stem from the algorithms employed in these frameworks, which significantly increase the difficulty and runtime of the processes. For example, P79 P79 mentions that the complexity of the test generation and analysis processes impacts performance and scalability, and P20 P20 critiques the Specmate technique for its excessive complexity, which undermines runtime efficiency.

Furthermore, 20 studies identified limitations in evaluation or demonstration, emphasizing that their experiments were insufficient to validate the efficacy of methodologies in other scenarios, particularly from an industrial perspective (e.g., P7 P7, P19 P19, P74 P74).

5.6.2. Insight Future View from Selected Studies

This section aims to discuss the key challenges and directions for future research identified in the selected studies. Notably, these studies share common themes regarding challenges and proposed future work, including improving or extending existing methodologies, conducting further evaluations, and addressing unresolved issues. To provide a structured analysis, the challenges and future work are categorized into several different areas. We illustrate the results in Table 28.

Table 28. Future Direction Results of Selected Studies (RQ5)

Future Directions	Paper IDs	Num.
Extension to Other Coverage Criteria or Requirements	P5, P11, P13, P14, P15, P16, P17, P18, P21, P22, P23, P28, P30, P32, P34, P35, P36, P42, P44, P45, P46, P47, P48, P49, P50, P51, P53, P54, P55, P57, P58, P59, P60, P61, P62, P65, P66, P67, P68, P73, P75, P78, P86, P92, P95, P98, P100, P101, P102, P103, P104, P105, P106, P107, P108, P110, P129, P130, P132, P133, P134, P137, P145, P146, P147, P148, P150, P151, P152, P158, P159, P160	72
Further Validation	P5, P6, P9, P14, P15, P19, P20, P21, P26, P38, P48, P54, P59, P72, P74, P75, P86, P89, P93, P96, P101, P108, P109, P110, P113, P115, P118, P119, P121, P125, P126, P127, P138, P140, P142, P143, P144, P145, P146, P156, P159, P160	42
Completeness Improvement	P24, P35, P36, P40, P45, P51, P52, P69, P71, P79, P80, P89, P92, P93, P95, P96, P112, P117, P119, P125, P126, P128, P129, P131, P134, P138, P140, P142, P149, P154, P155, P157	32
Automation Improvement	P4, P5, P6, P10, P11, P12, P46, P53, P56, P58, P67, P72, P73, P76, P88, P95, P104, P106, P115, P139, P144, P151, P152, P154	24
Extension of Other Techniques	P9, P19, P20, P22, P27, P30, P31, P33, P34, P37, P43, P44, P63, P70, P71, P76, P77, P88, P94, P127, P128, P147, P153, P157	24
Extension to Other Domains or Systems	P8, P10, P24, P26, P28, P29, P33, P41, P42, P47, P56, P61, P63, P64, P66, P69, P73, P77, P79, P80, P100, P102, P107, P135	24
Extension to Other Phases or Test Patterns	P1, P2, P7, P12, P16, P21, P25, P26, P37, P51, P52, P57, P62, P65, P74, P98, P105, P106, P111, P112, P114, P137, P153, P156	24
Performance Improvement	P4, P27, P42, P52, P94, P106, P107, P130, P132, P133, P138, P143, P149, P155	14
Real Tool Development	P1, P7, P13, P54, P56, P64, P78, P110, P112, P113, P114, P116, P117, P140	14
Benchmark Construction	P3, P39, P116, P139	4
Traceability Improvement	P31, P39, P121, P137	4
Robustness Improvement	P18, P118, P141	3
NA	P82, P83, P84, P90, P91, P99, P122, P123, P124, P136, P161	11

Extensions to other coverage criteria or requirements, test phases or patterns, and domains or systems are the three most commonly identified future directions and challenges in the reviewed studies. These directions correspond to limitations in framework design, as most frameworks are unable to address all usage scenarios, such as requirements coverage, test phases, or diverse software systems. Consequently, many studies propose expanding their scope to cover additional scenarios. For instance, P67 P67 plans to emphasize non-functional requirements in their test generation process, rather than focusing solely on functional requirements. P65 P65, having designed a method for generating test scenarios, intends to broaden their approach to encompass other phases of software development or testing. Similarly, P25 P25, which focuses on acceptance test case generation for embedded systems, plans to extend their application beyond embedded systems to enhance generalization.

Although limitations in automation were discussed in the previous section, several papers propose future plans to reduce human intervention in the test generation process. We identified 24 papers outlining plans to improve automation levels (e.g., P67 P67, P72 P72, among others).

Robustness, completeness, and traceability are widely regarded as critical factors in automated software testing research. While it is challenging to qualitatively assess these factors for a REDAST method, certain steps can be identified that may negatively impact framework robustness, completeness, or traceability. In the selected papers, some studies explicitly discuss their future directions for improving these aspects. For example, P114 P118 plans to enhance fault-handling diversity to strengthen framework robustness. P93 P96 aims to add a user input monitoring function as part of future work to improve completeness. Meanwhile, P116 P121 intends to further investigate the impact of requirement changes within the REDAST process, enabling better traceability between requirements and test artifacts.

Benchmark construction, real tool development, and further validation reflect researchers' efforts to broaden the impact of their studies. By enhancing the post-generation environment, the potential capabilities of these studies can be more thoroughly explored. For instance, P3 P3 plans to build benchmark test suites that are independent of specific model-based testing languages, which could significantly advance future research in related fields. P56 P56 aims to investigate opportunities for integrating their approach into larger industrial applications to enhance work efficiency. Additionally, recognizing that their current case study is insufficient to fully demonstrate the methodology's efficacy, P84 P86 intends to conduct more comprehensive case studies to further evaluate the performance of their method.

Some studies discuss the extension of existing techniques or performance improvements in their work, often focusing on introducing new techniques or frameworks to enhance test generation performance. For example, P19 P19 plans to expand NLP capabilities and explore more advanced automation techniques as part of their future work. Similarly, P91 P94 intends to reduce the state space of the studied model to improve the efficiency and effectiveness of their test case generation methods.

In the result, we identified that most of the studies pose an extension to other coverage criteria or other requirements (72 papers), followed by further validation (42 papers), improvement of the completeness in their future work (32 papers), and so on, which matches the results in the limitation results.

5.6.3. Findings: Cross-Analysis of Demonstration Types and Validation Challenges

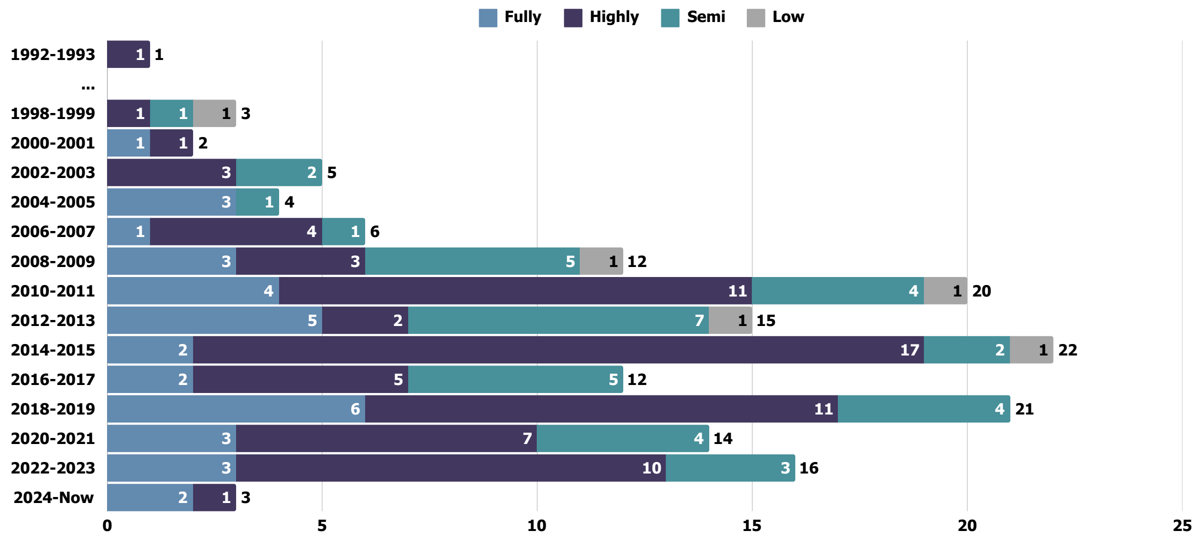
In analyzing the limitations and future directions of existing studies, we found that many studies identify evaluation or demonstration constraints and emphasize the need for further validation. This finding highlights the distribution of different demonstration types among studies that acknowledge validation-related limitations and future research directions. From our results in Table 29, 15 out of 69 papers that adopted conceptual case studies reported validation-related issues in their limitations or future directions, while 14 out of 69 identified similar concerns. Similarly, 22 out of 66 papers that employed real case studies reported validation-related issues in their limitations, with 13 out of 66 mentioning them in their future research directions. These findings suggest that validation challenges persist regardless of the chosen demonstration method. Additionally, we observed that papers employing real case studies more frequently reported validation-related issues. We hypothesize that this may be because conceptual case studies, being designed specifically for demonstration purposes, can more effectively represent the REDAST process in a controlled manner.

Table 29. Cross Distribution of Demonstration Types and Validation Related Future Direction or Limitation

	Conceptual Case Study	Real Case Study	NA	Dataset Evaluation
Further Validation	15	22	5	0
Limitation of Demonstration or Evaluation	14	13	3	1

5.6.4. Findings: Trend of Automation Level Over the Years

As a practical and impact-driven research area, REDAST places significant emphasis on automation, particularly in the context of industrial applications. Given that technological advancements contribute to automation in REDAST, we analyze the trend of automation levels over time, as illustrated in Figure 21. For clarity, we classify automation levels into two categories: fully and highly automated systems, which represent a high level of automation, and semi- and low-automated systems, which indicate a lower level of automation. Our findings reveal that while the proportion of low and semi-automated approaches has gradually declined, fully and highly automated methods have become increasingly dominant over the years. This trend suggests that technological advancements have progressively improved the automation level in REDAST studies. However, despite this overall improvement, the proportion of fully automated systems has not increased significantly. We attribute this to a key limiting factor: while technological advancements enhance automation capabilities, achieving full automation still requires an appropriately designed framework. This aspect is largely independent of technological progress and instead relies on methodological and architectural considerations in REDAST research.



Period	Fully	Highly	Semi	Low	Total
1992-1993		1			1
...					
1998-1999		1	1	1	3
2000-2001	1	1			2
2002-2003		3	2		5
2004-2005	3		1		4
2006-2007	1	4	1		6
2008-2009	3	3	5	1	12
2010-2011	4	11	4	1	20
2012-2013	5	2	7	1	15
2014-2015	2	17	2	1	22
2016-2017	2	5	5		12
2018-2019	6	11	4		21
2020-2021	3	7	4		14
2022-2023	3	10	3		16
2024-Now	2	1			3

Figure 21. Automation Level by Year

[mybox, breakable, title=RQ5 Key Takeaways]

• The framework design improvement and extension is the most common limitation and future direction reported in REDAST studies. The selected papers frequently mention better coverage of the usage scenario and model configurations.

• Most REDAST studies are not fully automated, with some human invention still necessary in some key steps.

• Extension of evaluation and demonstration is always considered in the future directions. REDAST studies are strongly related to real applications, and hence, the evaluation and demonstration have a key future direction and need for evaluation in real practical settings.

6. Threats to Validity

6.1. Internal Validity

In this study, the first author designed the SLR protocol, which was reviewed and refined collaboratively with the second, third, and fourth authors before formal implementation. The detailed topics and search strings were iteratively adjusted and executed across multiple databases to optimize the retrieval of relevant results. To accommodate the varying search policies of these databases, the search strings were customized accordingly. The selection of studies followed a multi-stage filtering process to minimize selection bias. The first round of filtering was based on titles and abstracts. The second round involved brief reading and keyword matching, while the third round consisted of a comprehensive reading of the papers. The final selection was validated by all authors to ensure robustness. Following study selection, a data extraction process was designed using Google Forms.

All authors participated in a pilot test to refine the data extraction procedure and ensure consistency in capturing the necessary information.

6.2. Construct Validity

To mitigate threats to construct validity, we conducted the search process across six widely used scientific databases, employing a combination of automated and manual search strategies. Extensive discussions among all authors were held to refine the inclusion and exclusion criteria, ensuring they effectively supported the selection of the most relevant studies for this SLR. Some of the selected studies included vague descriptions of their methodologies, posing potential threats to the validity of the study. These cases were carefully reviewed and deliberated upon by the first and second authors to reach a consensus on their inclusion.

6.3. Conclusion Validity

The threat to conclusion validity was minimized through a carefully planned and validated search and data extraction process. To ensure the extracted data aligned with our study requirements, we designed the data extraction form based on the predefined research questions (RQs). The first author initially extracted data from a subset of selected papers using this form, after which the extracted data was reviewed and verified by the other authors. Once validated, the first author used the refined form to extract data from the remaining studies. During data analysis and synthesis, multiple discussions were conducted to determine the most effective categorization and representation of the data, ensuring robust and meaningful conclusions.

6.4. External Validity

To address the threat to external validity, we employed a combination of automated and manual search strategies, adhering to widely accepted guidelines [26,71]. Our methodology section provides a detailed explanation of the inclusion and exclusion criteria. Specifically, we focused on peer-reviewed academic studies published in English, excluding grey literature, book chapters, opinion pieces, vision papers, and comparison studies. While these criteria may exclude some potentially relevant works, they were implemented to minimize bias in the selection process. We adopted a broad inclusion approach, considering studies regardless of their publication quality. Furthermore, our search encompassed publications from 1992 to the present, ensuring comprehensive coverage of advancements in the field of REDAST.

7. Discussion and Roadmap

Based on the results of our predefined RQs, we present several guidelines, insights, and recommendations for future research in REDAST.

7.1. Data Preprocessing for REDAST

Data is the basis of the REDAST methodology. REDAST methodology is a data-oriented process, while the acquired data primarily determine the framework design in specific usage scenarios. We found that most of the selected papers specified their data usage strategy. We identified many papers that adopted industrial data for their development and demonstration, e.g., P85 P88, P109 P113, P155 P160, etc. These data, however, originated from raw industrial cases and require further pre-processing for development. Thus, we recommend researchers customize the pre-processing to match their framework design. Besides, we also identified that some studies require additional data for the development, e.g., the training data for ML-based methods (P18 P18, P52 P52, P92 P95, etc). Our other recommendation is to align data for development with the methodology framework and the experimental demonstration, which maintains performance consistency throughout the process, from design to development to evaluation and implementation.

7.2. Requirements Input for REDAST

The results of our research questions (RQs) highlight the diverse preferences for requirements input. Findings in Section 5.2.4 suggest that the adoption of requirements specification is closely related to the intended usage scenario. Therefore, rather than recommending a specific combination of format and notation, we first suggest that researchers select requirements specifications based on their actual application scenarios.

Under general end goals, textual requirements offer the greatest flexibility and broad applicability across various target software systems, allowing for diverse notational choices to accommodate different tasks. For specialized or critical systems, structured formats such as model-based and tabular requirements are typically preferred, while formal and constraint-based requirements are more commonly adopted in high-reliability domains. However, our findings indicate that the distinction between different types of requirements input is not always significant, as textual requirements are frequently used even in specialized systems.

Furthermore, the choice of requirements specification is not only influenced by the usage scenario but also affects subsequent implementation and scalability. The results of RQ1 demonstrate that a variety of requirement types and notations have been employed in previous REDAST studies. While only a few studies have successfully managed multiple types of requirements input, it is unrealistic to expect a single framework to accommodate all requirement formats. Thus, appropriately adapting the requirement type within the REDAST methodology can significantly expand its application scope and enhance its scalability.

7.3. Transformation Techniques for REDAST

The transformation techniques used in REDAST correspond to RQ2, where we categorize them into machine learning (ML)-based, NLP-pipeline-based, rule-based, metamodel-based, and search/graph-based approaches. Given that test artifacts are generally structured data, rule-based and metamodel-based approaches—being the most commonly adopted techniques (appearing in 122 and 102 out of 156 papers, respectively)—facilitate structural transformations from requirements to test outcomes. Findings from RQ2 and RQ3 indicate that recent studies increasingly adopt diverse transformation techniques, regardless of the types of requirements inputs or test artifacts used. Based on these observations, we recommend employing a combination of transformation techniques rather than relying solely on conventional methods. While NLP-pipeline- and ML-based methods were previously considered "uncontrolled, " the era of large language models (LLMs) has introduced advanced flexibility and generalization capabilities, which have driven significant advancements in various domains, including REDAST studies. By integrating these emerging techniques with traditional rule-based approaches, the risks associated with uncontrolled behavior in cutting-edge methods can be mitigated, ensuring a balanced and effective transformation process.

7.4. Test Artifacts Output for REDAST

We identified a lack of details for the specifications of test artifacts. Even though we categorized the test outcomes in RQ3 based on their technologies, they are not formally reported in the papers. We formulated the test outcomes on the abstraction level, format level, and notation level. Another factor that should be considered for test artifacts is the executability. The need for executability varies with respect to different testing stages.

In general, we recommend that, in future studies, (1) the implementation details, such as abstraction, format, notation, and so on, are encouraged to be specified in the technical descriptions; (2) the executability should be seriously specified under the consideration of test stages or phases.

7.5. Evaluation Solutions for REDAST Studies

In RQ4, we identified evaluation and demonstration methods in the selected studies, categorizing them into case studies and dataset evaluations. However, we found that dataset evaluation is rarely adopted due to the limited availability of data resources for pairing requirements with test artifacts, appearing in only 5 out of 156 papers. Regarding case studies, researchers typically choose between real-world and conceptual cases. Our findings indicate that both conceptual and real case studies can provide strong persuasive value. However, as highlighted in Section 5.5.4, real case studies are generally preferred in certain specialized domains, such as web services, safety-critical systems, and objective-oriented systems. Despite their advantages, both real and conceptual case studies frequently report demonstration limitations.

To enhance the demonstration of methodological efficacy, we suggest that future studies incorporate both conceptual and real case studies within a single study. Conceptual case studies can serve to illustrate the methodological framework, while real case studies can be introduced in the final evaluation and demonstration phase to strengthen empirical validation especially.

Additionally, we observed that publicly available datasets for REDAST—and even in broader requirements engineering (RE) and software testing domains—are extremely limited. We urge the research community to focus on developing and maintaining public datasets for REDAST, as this would significantly improve the research environment and facilitate further advancements in the field.

7.6. Other Suggestion for REDAST Studies

Besides the above suggestions from technical or demonstration perspectives, other points should be taken into account.

Automation is a significant factor for REDAST studies. We identified that even though some studies report "automation" in titles or methodologies, human operations are still necessary within the generation process, or there is a lack of description of the implementation details of the automation. We strongly suggest that future researchers keep the automation details transparent.

Reproducibility. While REDAST methods are designed for industrial applications, the implementation developers and end users expect the methods to be directly applied. The reproducibility of the methodologies determines if the method can be successfully spread among industrial workflows. Thus, we recommend that researchers provide sufficient implementation details for reproducing or directly attaching code links to the paper.

8. Conclusion

Requirements Engineering-Driven Automated Software Testing (REDAST) presents a significant yet challenging task in contemporary software engineering research. Automating the generation of test artifacts from requirements has the potential to greatly enhance the efficiency and effectiveness of the testing workflow. However, the absence of systematic guidelines and comprehensive literature reviews on REDAST methodologies complicates research efforts and impedes progress in the field. This article presents a systematic literature review (SLR) on the technical approaches and solutions proposed for REDAST across various software systems.

Our review identified 156 relevant studies from an initial pool of 27, 333 papers through a rigorous multi-stage filtering, selection, and processing methodology. These studies were analyzed from five key perspectives: requirements input, transformation techniques, test outcomes, evaluation methods, and limitations. Our results show that (1) Functional requirements, model-based specifications, and natural language (NL) requirements are the most frequently used types, formats, and notations, respectively; (2) Rule-based techniques dominate in REDAST studies, while machine learning (ML)-based techniques are relatively underexplored; (3) Most frameworks are sequential, employing a single intermediate representation; (4) Studies frequently focus on concrete test artifacts; (5) Test cases, structured textual formats, and requirements coverage are the most commonly discussed types, formats, and coverage approaches, respectively; (6) While most studies conduct conceptual demonstrations, relatively few utilize dataset-based evaluations; (7) Although most studies provide robust methodological explanations, only half report promising experimental outcomes; (8) Only 35 studies achieve full automation, with most requiring unnecessary manual intervention; (9) Framework design remains the most frequently cited limitation, particularly the inability to handle complex configurations; (10) Many studies propose extending coverage criteria or addressing other requirement types.

Building on these findings, we propose several recommendations to advance REDAST research. The remarkable advancements in large language models (LLMs) highlight the potential of AI-based techniques for transformation tasks. Emphasis should also be placed on enhancing automation and reproducibility to realize the full efficiency gains promised by REDAST methodologies.

This study focuses exclusively on requirements-driven testing due to the vast volume of related literature. However, other stages of verification and validation are equally critical for comprehensive exploration. We aim to expand future research to cover broader alignments within the software development lifecycle (SDLC), bridging gaps across the entire verification and validation spectrum.

References

1. Bertolino, A. Software Testing Research: Achievements, Challenges, Dreams. In Proceedings of the Future of Software Engineering (FOSE '07), 2007, pp. 85–103. https://doi.org/10.1109/FOSE.2007.25.
2. Baresi, L.; Pezzè, M. An Introduction to Software Testing. *Electronic Notes in Theoretical Computer Science* **2006**, *148*, 89–111. Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004), https://doi.org/https://doi.org/10.1016/j.entcs.2005.12.014.
3. Barr, E.T.; Harman, M.; McMinn, P.; Shahbaz, M.; Yoo, S. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* **2015**, *41*, 507–525. https://doi.org/10.1109/TSE.2014.2372785.
4. Unterkalmsteiner, M.; Feldt, R.; Gorschek, T. A taxonomy for requirements engineering and software test alignment. *ACM Trans. Softw. Eng. Methodol.* **2014**, *23*. https://doi.org/10.1145/2523088.
5. Mustafa, A.; Wan-Kadir, W.M.; Ibrahim, N.; Shah, M.A.; Younas, M.; Khan, A.; Zareei, M.; Alanazi, F. Automated test case generation from requirements: A systematic literature review. *Computers, Materials and Continua* **2021**, *67*, 1819–1833.
6. Unterkalmsteiner, M.; Feldt, R.; Gorschek, T. A taxonomy for requirements engineering and software test alignment. *ACM Trans. Softw. Eng. Methodol.* **2014**, *23*. https://doi.org/10.1145/2523088.
7. Garousi, V.; Joy, N.; Keles, A.B. AI-powered test automation tools: A systematic review and empirical evaluation. *ArXiv* **2024**, *abs/2409.00411*.
8. Berger, C.; Rumpe, B. Engineering Autonomous Driving Software. *ArXiv* **2014**, *abs/1409.6579*.
9. Kitchenham, B.; Madeyski, L.; Budgen, D. SEGRESS: Software engineering guidelines for reporting secondary studies. *IEEE Transactions on Software Engineering* **2022**, *49*, 1273–1298.
10. ur Rehman, T.; Khan, M.N.A.; Riaz, N. Analysis of requirement engineering processes, tools/techniques and methodologies. *International Journal of Information Technology and Computer Science (IJITCS)* **2013**, *5*, 40.
11. Arora, C.; Grundy, J.; Abdelrazek, M., Advancing Requirements Engineering Through Generative AI: Assessing the Role of LLMs. In *Generative AI for Effective Software Development*; Nguyen-Duc, A.; Abrahamsson, P.; Khomh, F., Eds.; Springer Nature Switzerland: Cham, 2024; pp. 129–148. https://doi.org/10.1007/978-3-031-55642-5_6.
12. Glinz, M. A glossary of requirements engineering terminology. *Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version* **2011**, *1*, 56.
13. Pohl, K. *Requirements engineering: fundamentals, principles, and techniques*; Springer Publishing Company, Incorporated, 2010.
14. Zowghi, D.; Coulin, C. Requirements elicitation: A survey of techniques, approaches, and tools. *Engineering and managing software requirements* **2005**, pp. 19–46.
15. Kotonya, G.; Sommerville, I. *Requirements engineering: processes and techniques*; Wiley Publishing, 1998.
16. Chung, L.; Nixon, B.A.; Yu, E.; Mylopoulos, J. *Non-functional requirements in software engineering*; Vol. 5, Springer Science & Business Media, 2012.
17. Sneha, K.; Malle, G.M. Research on software testing techniques and software automation testing tools. In Proceedings of the 2017 international conference on energy, communication, data analytics and soft computing (ICECDS). IEEE, 2017, pp. 77–81.
18. Alaqail, H.; Ahmed, S. Overview of software testing standard ISO/IEC/IEEE 29119. *International Journal of Computer Science and Network Security (IJCSNS)* **2018**, *18*, 112–116.
19. Rafi, D.M.; Moses, K.R.K.; Petersen, K.; Mäntylä, M.V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In Proceedings of the 2012 7th international workshop on automation of software test (AST). IEEE, 2012, pp. 36–42.

20. Deming, C.; Khair, M.A.; Mallipeddi, S.R.; Varghese, A. Software Testing in the Era of AI: Leveraging Machine Learning and Automation for Efficient Quality Assurance. *Asian Journal of Applied Science and Engineering* **2021**, *10*, 66–76.
21. Balaji, S.; Murugaiyan, M.S. Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. *International Journal of Information Technology and Business Management* **2012**, *2*, 26–30.
22. Mathur, S.; Malik, S. Advancements in the V-Model. *International Journal of Computer Applications* **2010**, *1*, 29–34.
23. Atoum, I.; Baklizi, M.K.; Alsmadi, I.; Otoom, A.A.; Alhersh, T.; Ababneh, J.; Almalki, J.; Alshahrani, S.M. Challenges of software requirements quality assurance and validation: A systematic literature review. *IEEE Access* **2021**, *9*, 137613–137634.
24. Unterkalmsteiner, M.; Gorschek, T.; Feldt, R.; Klotins, E. Assessing requirements engineering and software test alignment—Five case studies. *Journal of systems and software* **2015**, *109*, 62–77.
25. Petersen, K.; Vakkalanka, S.; Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* **2015**, *64*, 1–18. https://doi.org/https://doi.org/10.1016/j.infsof.2015.03.007.
26. Wohlin, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In Proceedings of the Proceedings of the 18th international conference on evaluation and assessment in software engineering, 2014, pp. 1–10.
27. Shamsujjoha, M.; Grundy, J.; Li, L.; Khalajzadeh, H.; Lu, Q. Developing mobile applications via model driven development: A systematic literature review. *Information and Software Technology* **2021**, *140*, 106693.
28. Naveed, H.; Arora, C.; Khalajzadeh, H.; Grundy, J.; Haggag, O. Model driven engineering for machine learning components: A systematic literature review. *Information and Software Technology* **2024**, p. 107423.
29. Mustafa, A.; Wan-Kadir, W.M.; Ibrahim, N.; Shah, M.A.; Younas, M.; Khan, A.; Zareei, M.; Alanazi, F. Automated test case generation from requirements: A systematic literature review. *Computers, Materials and Continua* **2021**, *67*, 1819–1833.
30. Klaus, P.; Chris, R. *Requirements Engineering Fundamentals*, 1st ed.; Rocky Nook, 2011.
31. Wagner, S.; Fernández, D.M.; Felderer, M.; Vetrò, A.; Kalinowski, M.; Wieringa, R.; Pfahl, D.; Conte, T.; Christiansson, M.T.; Greer, D.; et al. Status quo in requirements engineering: A theory and a global family of surveys. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **2019**, *28*, 1–48.
32. Kumar, L.; Baldwa, S.; Jambavalikar, S.M.; Murthy, L.B.; Krishna, A. Software functional and non-function requirement classification using word-embedding. In Proceedings of the International Conference on Advanced Information Networking and Applications. Springer, 2022, pp. 167–179.
33. Sutcliffe, A. Scenario-based requirements analysis. *Requirements engineering* **1998**, *3*, 48–65.
34. De Landtsheer, R.; Letier, E.; Van Lamsweerde, A. Deriving tabular event-based specifications from goal-oriented requirements models. *Requirements Engineering* **2004**, *9*, 104–120.
35. Van Lamsweerde, A. Goal-oriented requirements engineering: A guided tour. In Proceedings of the Proceedings fifth ieee international symposium on requirements engineering. IEEE, 2001, pp. 249–262.
36. Mordecai, Y.; Dori, D. Model-based requirements engineering: Architecting for system requirements with stakeholders in mind. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE). IEEE, 2017, pp. 1–8.
37. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.V.; Batista-Navarro, R.T. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–41.
38. Yang, X.; Zhang, J.; Zhou, S.; Wang, B.; Wang, R. Generating Test Scenarios using SysML Activity Diagram. In Proceedings of the 2021 8th International Conference on Dependable Systems and Their Applications (DSA), 2021, pp. 257–264. https://doi.org/10.1109/DSA52907.2021.00039.
39. Hooda, I.; Chhillar, R. A review: study of test case generation techniques. *International Journal of Computer Applications* **2014**, *107*, 33–37.
40. Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; Wang, Q. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* **2024**.
41. Clark, A.G.; Walkinshaw, N.; Hierons, R.M. Test case generation for agent-based models: A systematic literature review. *Information and Software Technology* **2021**, *135*, 106567.
42. Anand, S.; Burke, E.K.; Chen, T.Y.; Clark, J.; Cohen, M.B.; Grieskamp, W.; Harman, M.; Harrold, M.J.; McMinn, P.; Bertolino, A.; et al. An orchestrated survey of methodologies for automated software test case generation. *Journal of systems and software* **2013**, *86*, 1978–2001.

43. Jiang, J.; Wang, F.; Shen, J.; Kim, S.; Kim, S. A Survey on Large Language Models for Code Generation. *arXiv preprint arXiv:2406.00515* **2024**.
44. Ahmed, A.; Azab, S.; Abdelhamid, Y. Source-Code Generation Using Deep Learning: A Survey. In Proceedings of the EPIA Conference on Artificial Intelligence. Springer, 2023, pp. 467–482.
45. Gurcan, F.; Dalveren, G.G.M.; Cagiltay, N.E.; Roman, D.; Soylu, A. Evolution of Software Testing Strategies and Trends: Semantic Content Analysis of Software Research Corpus of the Last 40 Years. *IEEE Access* **2022**, *10*, 106093–106109. https://doi.org/10.1109/ACCESS.2022.3211949.
46. Umar, M.A. Comprehensive study of software testing: Categories, levels, techniques, and types. *International Journal of Advance Research, Ideas and Innovations in Technology* **2019**, *5*, 32–40.
47. Atifi, M.; Mamouni, A.; Marzak, A. A comparative study of software testing techniques. In Proceedings of the Networked Systems: 5th International Conference, NETYS 2017, Marrakech, Morocco, May 17-19, 2017, Proceedings 5. Springer, 2017, pp. 373–390.
48. IEEE Standard for System, Software, and Hardware Verification and Validation - Redline. *IEEE Std 1012-2016 (Revision of IEEE Std 1012-2012/ Incorporates IEEE Std 1012-2016/Cor1-2017) - Redline* **2017**, pp. 1–465.
49. Tran, H.K.V.; Unterkalmsteiner, M.; Börstler, J.; bin Ali, N. Assessing test artifact quality—A tertiary study. *Information and Software Technology* **2021**, *139*, 106620.
50. Hinton, G.E.; Osindero, S.; Teh, Y.W. A fast learning algorithm for deep belief nets. *Neural computation* **2006**, *18*, 1527–1554.
51. Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **2008**, *9*.
52. Loucopoulos, P.; Karakostas, V. *System requirements engineering*; McGraw-Hill, Inc., 1995.
53. Sommerville, I. Software engineering 9th **2011**.
54. Loniewski, G.; Insfran, E.; Abrahão, S. A systematic review of the use of requirements engineering techniques in model-driven development. In Proceedings of the Model Driven Engineering Languages and Systems: 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part II 13. Springer, 2010, pp. 213–227.
55. Bruel, J.M.; Ebersold, S.; Galinier, F.; Naumchev, A.; Mazzara, M.; Meyer, B. Formality in software requirements. *CoRR* **2019**.
56. Nikitin, D. SPECIFICATION FORMALIZATION OF STATE CHARTS FOR COMPLEX SYSTEM MANAGEMENT. *Bulletin of National Technical University" KhPI". Series: System Analysis, Control and Information Technologies* **2023**, pp. 104–109.
57. Shafiq, S.; Minhas, N.M. Integrating Formal Methods in XP-A Conceptual Solution. *Journal of Software Engineering and Applications* **2014**, *7*, 299–310.
58. Dulac, N.; Viguier, T.; Leveson, N.; Storey, M.A. On the use of visualization in formal requirements specification. In Proceedings of the Proceedings IEEE Joint International Conference on Requirements Engineering. IEEE, 2002, pp. 71–80.
59. Committee, I.C.S.S.E.S.; Board, I.S.S. *IEEE recommended practice for software requirements specifications*; Vol. 830, IEEE, 1998.
60. Henderson-Sellers, B. UML-the Good, the Bad or the Ugly? Perspectives from a panel of experts. *Software & Systems Modeling* **2005**, *4*.
61. Veizaga, A.; Alferez, M.; Torre, D.; Sabetzadeh, M.; Briand, L. On systematically building a controlled natural language for functional requirements. *Empirical Software Engineering* **2021**, *26*, 79.
62. Pa, N.C.; Zain, A.M. A survey of communication content in software requirements elicitation involving customer and developer **2011**.
63. Barata, J.C.; Lisbôa, D.; Bastos, L.C.; Neto, A.G.S.S. Agile requirements engineering practices: a survey in Brazilian software development companies **2022**.
64. Barr, E.T.; Harman, M.; McMinn, P.; Shahbaz, M.; Yoo, S. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* **2015**, *41*, 507–525. https://doi.org/10.1109/TSE.2014.2372785.
65. Duque-Torres, A.; Klammer, C.; Pfahl, D.; Fischer, S.; Ramler, R. Towards Automatic Generation of Amplified Regression Test Oracles. *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* **2023**, pp. 332–339.
66. Molina, F.; Gorla, A. Test Oracle Automation in the era of LLMs. *ArXiv* **2024**, *abs/2405.12766*.
67. Wang, Y.; Mäntylä, M.; Liu, Z.; Markkula, J.; Raulamo-Jurvanen, P. Improving test automation maturity: A multivocal literature review. *Software Testing* **2022**, *32*.
68. Zhu, H.; Hall, P.A.; May, J.H. Software unit test coverage and adequacy. *Acm computing surveys (csur)* **1997**, *29*, 366–427.

69. Sykora, K.; Ahmed, B.S.; Bures, M. Code Coverage Aware Test Generation Using Constraint Solver. *ArXiv* **2020**, *abs/2009.02915*.
70. Tufano, M.; Chandel, S.; Agarwal, A.; Sundaresan, N.; Clement, C.B. Predicting Code Coverage without Execution. *ArXiv* **2023**, *abs/2307.13383*.
71. Kitchenham, B.; Brereton, O.P.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology* **2009**, *51*, 7–15.