EasyChair Preprint

№ 12721

Software Traceability Across SDLC: a Comprehensive Survey

Nakul Sharma and Amar Buchade

March 27, 2024

Software Traceability Across SDLC: A Comprehensive Survey

Nakul Sharma[1*], Amar Buchade[1,2]

[1)] *Dept of Artificial Intelligence and Data Science, VIIT, Pune, India*

E-mail: nakul777@gmail.com

[2)] *Dept of Artificial Intelligence and Data Science, VIIT, Pune, India*

E-mail: amar.buchude@viit.ac.in

***Abstract**: Evidence-Based Software Engineering has aided a lot in Software Engineering research. Evidence-based data collection, data synthesis, data analysis and its recommendation has been a lighthouse to aid researches across different fields within SE domain. Software traceability has been under investigation ever since the software projects suffered from poor quality and lack of user satisfaction. This paper gives a review of software traceability mechanism as it is implemented across the different phases in SDLC. Several approaches have been proposed to implement software traceability across SDLC life cycle. The insights mentioned can provide additional directions to perspective researchers.*

Keywords: EBSE, Software Traceability, SDLC, Requirement Traceability, Traceability

1. INTRODUCTION

Software traceability is defined as a link between two more software artifacts existing against each phase of SDLC as shown in Figure-1. SDLC phases produce software artifacts as each phase gets completed. In each phase of SDLC, various intermediate artifacts also get created.

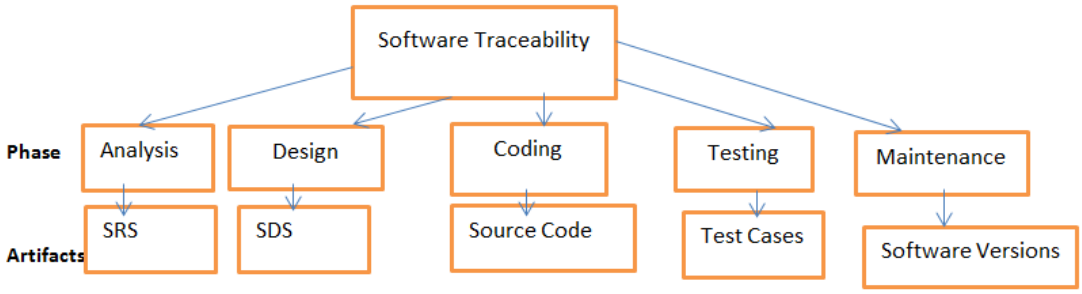

Fig. 2.1. Software Traceability for SDLC artifacts

Software traceability provides a medium to check linkages between different software artifacts. Traceability can be conducted among artifacts which are produced as a result

[*] Corresponding author: nakul777@gmail.com

completion of certain task, processes of phases. Traceability offers means to congregate different artifacts for various processes. Traceability can be between artifacts arising from the same phase of SDLC or from different phases of SDLC. There are several advantages of traceability. Traceability helps specific software developer in firing a query which aids in seeing the effect of changes being introduced. Traceability aids various stakeholders in carrying out their job profiles more effectively and proficiently. Traceability can aid software developers in studying the structure of the large source code projects. Traceability also helps in introducing changes to the software being developed. The list of stakeholders in software traceability research is shown in Figure 2.

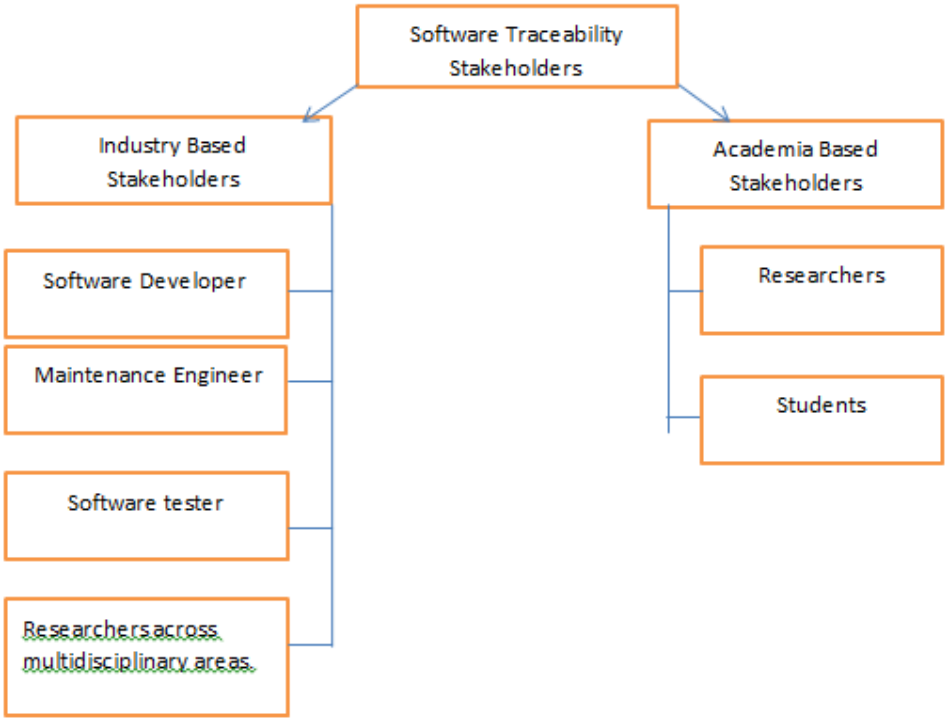

Fig. 2.2. Software Traceability Stakeholders

1.1 Background

Software Traceability has been under investigation ever since change management and analysis of these changes were needed by the maintenance engineers. Changes to the software projects tend to cost more time passes and changes introduced late in the software create rippling effect on different source code. There are various information models, automated ways of trace creation and its maintenance which have been conceptualized and made industry ready by the researchers. However, as the external environment has evolved, the methodologies and framework needed for conducting system trace also needs evolution. A comprehensive survey can address various facets of research presented in the literature.

1.2. Industry Based Stakeholders

1. Software developer
2. Maintenance engineer
3. Software tester
4. System analysts
5. System designers
6. Researchers across different multidisciplinary areas.

Software developer can enhance their program understanding and program comprehension by checking the traceability link as they exist between two or more software

artifacts. Maintenance engineers are tasked with making changes, creating versions and furthering software's product line. Hence, it is imperative that software traceability is conducted by maintenance engineers. Testing phase also involves tracing the bugs and errors present in the program. The researchers within the ambit of software engineering are stakeholders in software traceability research. The cross domain researchers specifically have interest in software traceability research.

1.3. Academia Based Stakeholders

Academic based stakeholders include researchers and their students. Researchers also collaborate with the industrial projects for the purpose of find linkages across software artifacts. Academic researchers can also collaborate on short and long term agreements to find traceability on large-scale software projects.

1.4 Traceability Categorization

Software traceability approaches are categorized as follows:-

1. Static

The static traceability achieves linkage between high ranked and lower ranked artifacts. It makes use of contextual information and the source code in achieving this goal.

2. Dynamic

The dynamic traceability needs executable compiler for the purpose of running previously defined scenarios for executing traceability in order to identify software systems.

3. Hybrid traceability

Several authors also propose combining both static and dynamic mediums for achieving traceability. The hybrid traceability approach takes some aspects from static and dynamic traceability.

2. LITERATURE REVIEW METHODOLOGY

The authors had aggregated papers from various databases by restricting it to years 2018 to 2023. The search strings passed were as follows:-

1. "software traceability" "survey"
2. "software" "traceability" "source code"
3. "software" "requirement" "traceability"
4. "software" "trace"

Using the citation available in Google scholar, more papers were downloaded. The papers to be included and excluded were decided based on criteria's. The inclusion and exclusion criteria were applied to all the papers which were downloaded. In case any paper did not have the keywords of "traceability" or "source code", it was excluded from the list of papers. In some of the instances, these terms were found in the references section of the paper; hence they were also excluded from the list of papers to be studied. Table- 1 gives the list of papers which were excluded and the reasons for removal.

Table 2.1 List of Papers Excluded from the current survey

Ref. No.	*Title of Research Paper*	*Type of Research (SLR/SMS/Survey/Research Paper)*	*Reason for Removal*
51	*COMEX: A Tool for Generating Customized Source Code Representations*	*Research*	*traceability term not found in paper*
52	*Selection of Digital*	*Research*	*traceability &*

	Watermarking Techniques for Medical Image Security		*source code term not found in paper*
53	*Code smells and refactoring A tertiary systematic review of challenges and observations*	*SLR*	*trace, traceability and source code term not found in paper*
54	*Identifying, Categorizing and Mitigating Threats to Validity in Software Engineering Secondary Studies*	*Survey*	*traceability and source code term not found in paper*
55	*Grey Literature in Software Engineering: A Critical Review*	*Survey*	*Traceability term not found in paper*
56	*A Novel Paper Recommendation Method Empowered by Knowledge Graph: for Research Beginners*	*Research*	*software Traceability term not found in paper*
57	*Structural and Semantic Similarity Measurement of UML Use Case Diagram*	*Research*	*software traceability term not found in paper*
58	*SysML Modeling Mistakes and Their Impacts on Requirements*	*Research*	*Traceability term not found in paper*
59	*Design Methods And Processes For Ml/Dl Models*	*Research*	*Traceability term not found in research work*
60	*Finding Trends in Software Research*	*Survey*	*Paper related to Topic Modeling but did not have trace, traceability terms wrt source code*
61	*A Survey on Deep Learning for Software Engineering*	*Survey*	*No direct reference to traceability was there*

Table-2 gives the summary of literature which was surveyed including key findings and the future scope of research work as provided by the authors.

Table 2.2 Literature Review Summarized

Ref. No.	*Type of Research (SLR/SMS/Survey/ Research Paper)*	*Key Findings/Contribution of the Paper*	*Future Scope of Research Work Suggested*
1	*Survey*	*1. In Software Artifacts, Recovering Traceability Links.* *2. Traceability Direction and Evaluation* *3. Supports Change Impact Set* *4. Different Traceability-based link*	*1. Automatic Traceability applications for tools enhancement.* *2. Link recovery between software's trace artifacts* *3. Traceability systems which accept other that textual inputs*

		Recovery Methods	
2	*Research*	*1. Finding security tactics in source code 2. Mapping security requirements to source code 3.Tactic based modules visualization and its related dependencies*	*Not Mentioned*
3	*Survey*	*The author proposed a feature model for software traceability.*	*1. Traceability meta-model for different traces. 2. Trace Data Security 3. Trace types library and semantics 4. Verification and Validation of Testing. 5. Traceability in general programming language*
4	*SLR*	*Existing code based query reformation approaches use weighting the terms, relevance of the feedback and data mining, to reformulate query and for supporting code query search in different contexts. Methodology adopted for code search query evaluation has lesser developer's involvement and number of queries searched for internet-based code search is small. Existing approach towards query reformation have major challenges and limitations, as a result cannot be used by software professionals directly. Statistically there is a increase in the number of publications related to automated query reformation and searching of source code recent years at top venues of SE. Hence it is predicted that these research areas will have greater research interest in coming years. There is considerable research in both the local as well as internet based code search. However, both these searches lack generalizability, weak evaluation, noise. There are some common best practices for query reformulation across closed and open source industry community for code search.*	*Future work on code search pertains to expanding queries to accommodate more contexts in keywords, designing the necessary function for GA-based solutions, using the structure from source code or NLP for delivering better queries for code search.* *for adapting keyword based on specific contexts, employing genetic algorithms based solutions, supporting algorithms of term weighting with the contextual information, making use of stackoverflow for the purpose of complementing local code search, employing developer's cognitive abilities to reduce query worsening, using pseudo relevance feedback for reducing noise and complementing code retrieval, standardization of query formulation.*
5	*Research*	*A tool for change analysis.* *Language which allows capturing of traceability information and utilizes this information for change analysis for feature models.*	*Improving the change analysis language for addressing complex constructs and to address complicated queries.*

6	*Research*	*Traceability mechanism leading the circular economy. Software traceability has its relevance in this context.*	*Close collaboration of different actors is needed. There will be resistance from different people for adoption of change. But with the potential benefits being told, it will be possible to overcome these obstacles. Testing traceability for different industrial segment must be encouraged.*
7	*Research*	*Authors propose traceability framework and tool that visualizes requirement's traceability data. The tool is effective to help increase user's understanding of the data. The tool can also be applied to complex artifacts.*	*The proposed tool can be enhanced by filtering the search technique and limiting the search only for seeing the relationships, direction, link number and properties.*
8	*Survey*	*Authors provide an overall framework for Machine Learning in SE research. The evidence based SE has more applicability of ML which are evaluated empirically. SDLC's relationship with ML tools, techniques has been discussed. Some of the papers related to requirement traceability have been discussed.*	*Usage of ML's advanced techniques in SE domain to find the focused state of art work is the future scope.*
9	*Research*	*A methodology for evolving feature model is proposed for changing requirements using formal methods.*	*Experimental tool developed can be optimized for the dynamic demand.*
10	*SLR*	*The authors investigated issue-based traceability based on problem, artifact pair, techniques, evaluation targets. The paper also discusses various challenges in issue-based requirement traceability.*	*Additional information can help in generating better trace links which can aid in increasing the accuracy and the automation in traceability domain. The Issue centric traceability should be conducted to link issue based traceability study to other artifacts. Accuracy and reliability of the trace link must be investigated further. Open source dataset must be encouraged for better understanding of issue-based traceability.*
11	*Research*	*Authors proposed methodology generates rationale for tracing links. The proposed methodology consisted of NLP pipeline, data mining techniques for more generalized usage across the domains.*	*The proposed work can be generalized to include more domains.*
12	*Research*	*The authors claim that software is more than just source code. A taxonomy of artifacts categorization is provided*	*Future scope proposed by authors is more detailed taxonomy of the artifacts. A definition of software for practitioners is also part of future-*

			work.
13	*Research*	*Data constraints in eight java systems were studied. Different types of constraints were identified*	*Constraint Implementation Patterns (CIP) can be studied for different programming languages. Best practices can be evolved once the relationship between constraint types and CIP is clear. CIPs and automated detectors can help in extracting business rules and recovering the traceability links*
14	*Research*	*Authors propose recovering traceability links between diverse software's documentation to source code*	*A mechanism of filtering out noisy traceability links must be evolved by understanding which principle the document focuses on. Cross validation of traceability links and APIs in different documents should be encouraged.*
15	*Research*	*Authors use close relationships existing between the artifacts to enhance traceability in IR*	-
16	*Research*	*Authors utilize network science properties & ML for recovering trace links in semantics. Some interesting patterns were found when trace link data was modeled as network structure.*	*Future scope includes assessing how custom link labels also get linked to meaningful semantic association. Network based metrics can be combined with prediction techniques which use data. Network science properties can be used along with ML techniques to enhance traceability mechanism of different paper.*
17	*SMS*	*Many enhancement techniques have been proposed which support software traceability link. Open source data sets are more preferred than closed source data sets. Overall quality of research work is good but needs more practical industrial setting.*	*Traditional ML models are applied more to RT. As both ML and DL get mature, they can apply to Requirement Traceability. ML based techniques when combined makes the accuracy of linked recognition related to features better. Suitable feature selection technique is needed to improve performance.*
18	*Research*	*Authors define what all artifacts are produced while developing free projects. Authors also propose an approach using ML to identify and classify software artifacts.*	*Authors planned to create gitHub plugin for the purpose of identifying and visualizing artifacts which are missing and which are present. IR based approaches were also to be applied for feature extraction*
19	*Survey*	*Authors conducted a study to identify how artifacts like IBM digital can trace links from the requirement. In order to accomplish this, the taxonomies must be linked to artifact*	*A semi automation of trace links can be carried out with the help of NLP. Maintainability of trace links across products-lifeline can be studied.*

		which is a challenging task.	
20	*Research*	*Authors justify the need of breaking the code segment describe its associated benefits towards software evolution.*	*Authors state that best practices are not followed by practitioners while defining dependencies. Libraries also conform to semantic versioning when conducting break of code. When a tool related to software evolution is developed, it is more accurate, applicable for conducting development in pull environment.*
21	*Research*	*Authors analyzed the existing data-sets related to software traceability. The study articulates several attributes related to datasets. A software metric named 'Dataset Diversity Ratio' is also proposed*	*The proposed work provides future directions for making evaluation and practicality of Software Traceability research. The information about data sets can help serve needs of different researchers*
22	*SMS*	*The authors studied the goal of existing approaches in achieving software traceability and what methods are used for evaluation. Authors found out primarily requirement artifacts are most used for conducting software traceability. The existing proposed techniques deal with novel techniques to achieve traceability. These techniques further aid in software maintenance and correctness.*	*Future researchers should state to which software artifact their approach applies to. The various traceability approaches must be measured against standard benchmarks and techniques. The overall cost and performance measures of their techniques in addition to the accuracy and benefits associated with the traces. There should be empirical evidence suggesting relation between traceability and the quality attributes. Future studies are likely to provide more concrete high quality evidences*
23	*Research*	*The authors propose a tool NLTrace which apply transfer learning techniques on real world software traceability data-sets.*	*Author's claim that for reaching to industries acceptable level, F2 and MAP scores must have higher level of accuracy. Tracing task can be made better by using different sources data in multiple transfer learning policies*
24	*Research*	*The authors proposed a unified approach towards extracting keywords from source code, its associated documentation and test data. The authors introduce concept similarity for this purpose.*	*The future work involves studying the traceability from the semantics for artifacts of the software. The current corpus could be expanded to include various domains, languages and categories.*
25	*Research*	*The authors proposed NLP pipeline for providing a visual explanation for trace links. The domain related concepts were extracted and mined including concept-related sentences.*	*The authors work can be expanded to include more domains.*

26	*SLR*	*The author examines the problems of traceability links evaluation and provides guidelines for evaluating traceability techniques with its benchmark and properties.*	*The future work entails defining in clear terms the properties of metrics related to software traceability selected for benchmarking.*
27	*Research*	*The authors examine that feature traces are independent of the developer's memory. This condition influences the overall program comprehension of the source code. Author's provide an experimental design, challenges in implementing the methodology and null results.*	*The design of studies conducted for developer's memory need improvement. Developer's memory can also be analyzed to indicate how they forget different knowledge based concepts. This can enable to improve automation of techniques needed for identifying experts. Newer hypothesis can be proposed to check if program comprehension and feature traces go hand in hand. Mapping different stakeholder's knowledge features to information of traces. Solving the feature traceability by providing feature annotation and ML based automation techniques.*
28	*Research*	*The authors propose a traceability recovery mechanism between test cases and the bug reports. The authors utilize LSA, LDA, BM25 and word vector for their work. Author's results indicated mild increase in the evaluation scores of precision & recall for all the traceability recovery techniques. Authors also recognize the basic NLP techniques are needed for the achieving better traceability of textual artifacts.*	*The current work can be expanded by including glossary of thesaurus for exploring traceability between bugs and test cases. Other data sets related to textual artifacts can be studied. The other traceability related algorithms, recovery techniques, can also be studied.*
29	*Research*	*The author's proposed a visual traceability related trace-map which showed inter-relations between different artifacts. The proposed framework provided both filtered and unfiltered view of the relationship.*	*The proposed work could be enhanced by increasing the level of abstraction for better program comprehension and for analyzing change impact tasks better. IR engine was proposed to be replaced by deep learning techniques to get better accuracy.*
30	*Survey*	*Authors found that irrespective of the project type and the means of development, the traceability costs wrt effort, time and money are main reasons preventing traceability adoption. Traceability is mainly done manually. Similar needs also need proper prioritization.*	*A follow-up study could be conducted for the purpose of checking and validating authors work. Another study can be done to check the software's practitioner's current traceability conditions and their needs in different environments and situations. Additional work can be conducted on specific methods, processes phases of SDLC or different environments. Software traceability*

			can be made focus of specific subtype of traceability applicability. Demographic conditions can also be included based on the datasets available. Empirically more data is needed for software traceability to grow at a much larger and greater pace.
31	*SMS*	*Authors conducted a study of different tools and approaches which are used for labeling source code elements. Taxonomy was proposed in terms of source, target, presentation and persistence.*	*it is necessary to filter the meta-data from the source code. In addition, how metadata evolves with changing source code also needs to be investigated. The meta data when combined with the source code can provide interesting information for the readers.*
32	*Research*	*The authors conducted a study to determine how intermediate artifacts can aid in increasing trace link's accuracy while considering the path from source entity, target entity and intermediate artifacts.*	*The future scope involves making use of deep learning based tracing algorithm. The intermediate artifacts could be used in graph based networks as well.*
33	*Research*	*The authors replaced call recommender with Boolean Matrix-Factorization and found several additional information in terms of discovering object usage and identifying corner cases that was not found out previously. The authors also use event streaming mining algorithm which learnt different code representations without using any prior domain knowledge. The resultant patterns were evaluated in terms of precision and recall. The results on both these metrics were better than the previous results.*	*The future work entailed combining data from multiple sources and adding different data sampling techniques, preprocessing the data to reflect the varied environments, automatic feature selection, providing support to the API patterns wrt quality and generalizability, providing a frequency threshold for different mining conditions, studying impact of new API usage in development process, learning from different code elements, providing parallel ML algorithms to source code input, creating newer applications based on API usage*
34	*Research*	*The authors created an Eclipse-based plugin for maintaining traces across similarly structured abstraction (horizontally) in complex systems. Traceability links included files such as html, source code, configuration files etc.*	*Not Mentioned*

35	*Research*	*The authors propose a semantic distance measurement for determining traceability between the software artifacts. Semantic distance metric is a relative metric on the scale of 0 to 100. The proposed metrics accuracy was observed by seeing the ranking order and score results. This accuracy was consistent with the changing scenarios. The scalability of the implemented tool was also acceptable.*	*The future scope of the work included making qualitative metrics better. The human factors reduce the overall accuracy of the tool and need to be considered.*
36	*Research*	*Authors propose a SysML model which allows interaction of SysML models within Virtual Reality (VR) environment. It also provides a facility to test tracing in VR environment.*	*Future work from this paper is creating models within VR, combining SysML tools with simulations, supporting stronger and more detailed verification system for evaluating usability of various stakeholders.*
37	*SMS*	*Authors identified a roadmap 44 major studies. The domain of result's specifications was from both software as well as automotive. FSM is mainly used while testing SPL. Behavioral and scenario based models are most used. In order to do evaluation, the case studies & experiments are used in Model-Based Testing (MBT) solutions. MBT based solutions don't have strong traceability solutions. Authors also propose to conduct user-models artifacts, tools, variability management, and traceability.*	*The proposed roadmap is useful for practitioners and future researchers to conduct traceability based MBT*
38	*Research*	*Authors state that in any SE tasks, the proposed methodology can be applied. The quality of query submitted by the developers was determined by the authors. Queries quality was checked through automation. The proposed Text Retrieval (TR) when applied for concept location determined the list of retrieved code elements which were related to change request. The proposed TR technique when applied to traceability link identified artifact which were difficult to trace due to low quality.*	*Quality assessment of the query reduces effort and time. It identifies software artifact which is difficult to trace.*

39	*Research*	*Authors present a methodology to improve precision and recall, which are the basic evaluation criteria's as well. In addition to the existing data sources in IR, some new data sources are also included for creating trace links. Interaction logs of developer's and the existing links between the artifacts are used to create the trace links. A hierarchical specification between source code & requirement specification was developed.*	*The proposed methodology could be integrated into Eclipse as a plugin for integration.*
40	*SMS*	*1. The IR models are utilized to recover traceable links between two or more artifacts. Developers have made a design decisions which affect the traceability in that same IR models produce different result. 2. Different software artifacts traceability link focus on requirements and source code with no extension to remaining phases of SDLC.3. IR based traceability were conducted on less than 500 artifacts.*	*Future study of the enhancement strategies reported could be done. A separate investigation on the patterns in which the contexts are applied could be carried out. Another future work is to map different dimensions according to other frameworks related to CoEST research topics*
41	*Research*	*1. A review of different characteristics based datasets employed for software traceability are recorded. 2. A framework to evaluate the datasets employed for software traceability. 3. The results of study are used to generate datasets for three baseline approaches related to training data.*	*The proposed approach could be enhanced to include different software metrics related to data quality and sampling techniques. NLP based techniques could be used to enhance the proposed technique. The link between domain knowledge of the person creating the query and quality of queries generated can be investigated.*
42	*SLR + Research*	*Author summarized the limitations of existing techniques in the SLR. Author's proposed graphs for weighing terms by using dynamic and source code based document structures.*	*The future work included applying Keyword based searching algorithms to IR related bug localization. Genetic algorithms can also be applied for IR-based bug localization algorithm. Introducing context in term and improving the term weighting algorithms. Improving the pseudo-relevance feedback.*
43		*In the context of ML application to SE, simple-neural networks are most used. ML applications also include developing recommendation system for helping managers make better informed decisions.*	*Author suggests creating a feedback loop for improving requirement traceability. In order to have better reliability of the results and to have better prediction accuracy, more experimentation is needed along with larger data sets. The authors also suggested Ml based solution of complex system integration problem.*

44	*SMS*	*The author's claim that traceability practices, impact software maintenance & software evolution. The studies also revealed that the proof of traceability's impact on maintenance and evolution is strong enough. Traceability link establishment and maintenance is costly. Authors also identified several barriers to apply traceability to the software maintenance.*	*More effective methods are required to measure rate benefit ratio using traceability for maintenance and software evolution.*
45	*SLR*	*The authors analyzed the existing Feature Location Techniques (FLT) needed in software maintenance techniques and found only 27% of the techniques as reproducible. Since it is difficult to reproduce majority of the research in FLT, the comparison of these proposed methodology in FLTs is not possible.*	*The authors indicate the strong need of standardizing empirical research in FLTs. The standards must be made for the FLTs in order to allow comparison.*
46	*Research*	*Author's evaluate the effect of word based similarity measures on ArDoCo tool.*	*The other complex strategies for evaluation of ArDoCo tool could be used.*
47	*Research*	*Author's propose a graph-based trace link recovery approach which gave precision value of 40% and lag of 50%.*	*Authors state that knowledge based representation can provide explanations and filter better results in trace link recovery*
48	*Research*	*The statistical data which represents links between software artifacts was duplicated in the given context. The statistical model can help study relatedness of the software artifacts using the artifact's properties.*	*The mathematical properties of the artifacts could be studied for making traceability more automated.*
49	*Research*	*Author contributes a data set on re-engineering variant rich systems. The developer's memory and knowledge needs can be enhanced by providing suitable documentation techniques. The feature traces impact developer's program understanding.*	*A Decision Support System could be developed for the re-engineering variant rich systems. New tools from different sources could be developed to elicitate required information for enhancing developer's memory. A management framework for feature traces could be developed. A real world recommendation system could be developed for different stakeholders. The current work could also be replicated for more conditions and different contexts.*

50	*Research*	*Whenever traceability link need to be found out between software artifacts, the corresponding personnel's who have worked on these artifacts should be consulted. The organizational structure must be known beforehand and traceability information may get affected by such a structure. In complex project, the different information regarding the same artifact may be present at different locations. In order to combine such type of information needs awareness of the company's norms and regulations. A fully automated solution may not be possible. Hence, human intervention may be needed for introducing traceability. Interoperability between the systems are needed for forming traceability links.*	*The authors would conduct traceability study for their industrial partners.*

3. DISCUSSION ON LITERATURE REVIEW

3.1 SLR/SMS/Survey

Among the literature, there are some SLRs, SMS and surveys which have focused on software traceability in different aspects along with the specific categories. There are SLRs focusing on the role of automatic traceability link in change impact analysis who investigate the literature according to traceability approaches, impact analysis sets, degree of evaluation, trace direction and methodology used for recovering traceability link. The SLR provide further directions of integrating deep learning and machine learning for automating traceability[1].

Some of survey focus on extracting feature model from the traceability approaches. The paper proposes traceability framework that can aid in industrial implementation of traceability mechanism. The feature model can be expanded to inculcate AI in Software Engineering domain, with its implications being seen in requirements, testing and source code [3].

Another SLR was conducted to access automated query reformulations for source code search. The author's focused on surveying the existing literature using different approaches used in this context. The SLR dives deep into term weighting, relevance feedback, semantic relations, thesaurus lookup, data mining which are the approaches used in query formulation. The survey provides various future directions for enhancing query reformulation on source code search [4].
There have been SLR's on the specified topic of Issue-Based Requirement Traceability. The authors discuss for each related literature the questions of problems, artifact pairs, techniques and evaluation targets. There were 40 papers specific to I-RT which was analyzed. The authors also presented future scope and direction arising out of the literature review [10].

There have been systematic mapping studies conducted on applying ML techniques to requirement-based traceability. The author identified 26 literatures as primary studies which had ML's applicability for RE based traceability. The author's opine applying multiple ML techniques to improve accuracy in feature related traceability recognition. The appropriate feature selection approach is necessary for improving performance of models [17].

There are different surveys conducted in context of requirement-based traceability and its associated benefits. The authors survey all the existing literature on software traceability for requirements. The authors conclude that it is new to consider traceability knowledge organizational structure to aid in requirement traceability. However, certain concerns related to the taxonomic enabled trace-links must be addressed for faster industry adoption [19].

3.2 Software Requirement

Traceability for software requirements have been done visually as well. The tools use data visualization technique to represent relationship between artifacts and requirements. The graphical representation developed can be traversed using impact analysis method. The tool developed helps in better understand the requirements [7].

The author employs the close relationships existing between different requirements level documents for conducting traceability. The authors use IR techniques and the relationship between documents to create trace links. The methodology proposed is evaluated using public datasets and standard evaluation metrics of precision and recall are used for scoring [15].

The author studies various attributes of data-sets for software traceability. A new metric related to data-set has also being proposed. A host of research directions are also provided in this research related to data-set based research [21].

3.3 Source code

Authors study the implementation of data constraints in java programming language. The author's manually identified four types of data constraints. The implementation patterns within the source code were then identified [13].

3.4 Change Impact analysis

Software testing based traceability approach helps achieving security control in source code. The security controls use principle of security by design. The author's proposed methodology showed that source codes having security design principles could be easily identified [2].

Another research on improving traceability management and change impact analysis is to create specific domain specific languages for feature modeling. The traceability related information is collected and uses model-based approach for its storage [5].

3.5 Software Product Line (SPL)

Authors create evolving feature model for SPL which change as the feature models changes. The author's proposed model is effective is useful in predicting evolving nature of SPL in line with the requirements [9].

The benefits of breaking the code are show-cased by the authors. In order to undertake breaking of code, first an assessment of the break must however, be carried out. The best practices can prevent the problems associated with breaking of code. the syntactic breaking code and their effect on versioning of the software is also studied [20].

3.6 Software Maintenance

Software maintenance engineer and worker make use of traceability for change management. The trace links between the artifacts can be seen visually by the authors. This is achieved by using NLP pipeline including identifying domain-specific concepts, getting a corpus of sentences related concepts, mining concept explanations and their usage examples. The proposed methodology also identifies relationship between concepts for concept explanation [11].

The software traceability between source code and related documentation is explored as exploratory research as well. The author's take as case study Lucene project and collect documents pertaining to different Bug report, mail lists, stack overflow Q& A documents and blogs. The author's then frame research questions [14].

The network science and machine learning models are also used in analyzing the semantic trace links. The authors herein utilize the network science concepts along with ML models for pattern identification. The trace links hence generated are useful for change and issue management [16].

The open-source codes can be studied as they are freely downloadable. The different granularities existing within the open source projects are studied by the authors. The author's propose ML-based techniques for software artifact identification [18].

3.7 General Literature Incorporating Traceability

Traceability framework has also been employed beyond software engineering domains. The traceability mechanism is being employed for the betterment of the circuit economy. Traceability is helping in attaining larger goals in assert industry as well. The meaning or interpretation may vary but underlining concept remains the same [6].

Machine learning applications to SDLC is studied as a survey. The author's found application of ML in finding software traces in requirements engineering. The applicability of ML is also accomplishes software traceability at artifacts level [8].

Traceability is also discussed in a limited way while describing what constitutes software. The author's categorized software into as different artifacts of 19 concrete categories. The author's concluded that source code itself consists of different types of code, different data associated with the source code and its associated documentation. The utility of this research related to software artifacts is that before software traceability can be carried out, it is essential to determine what constitute software [12].

Ref. No.	*Type of Research (SLR/SMS/Survey /Research Paper)*	*SDLC Phase addressed*	*Sub-area/Processes of SDLC addressed*
1	*SMS*	*Software Change Management, Software Maintenance*	*change impact analysis, Change Management*
2	*Research*	*Software Testing*	*Security Testing*
3	*Survey*	*Analysis*	*Traceability*
4	*SLR*	*Software Maintenance*	*Search-Based Software Engineering, Reverse Engineering*
5	*Research*	*Software Maintenance*	*Software product lines*
6	*Research*	*Information Lifecycle Management*	*sustainability, digitization, circular economy, build assert industry, digital threading*
7	*Research*	*Requirement Analysis*	*Requirement traceability, Traceability visualization, Visual framework*
8	*Survey*	*Across SDLC*	*Not applicable*
9	*Research*	*Software Maintenance*	*Software product lines*
10	*SLR*	*Analysis, Software Maintenance*	*Requirement traceability*
11	*Research*	*Software Maintenance*	*Traceability*
12	*Research*	*Software Maintenance*	*Software Organization*
13	*Research*	*coding*	*design patterns traceability*
14	*Research*	*Across SDLC*	*Traceability*
15	*Research*	*Analysis*	*Traceability*
16	*Research*	*Software Maintenance*	*Change management, issue management*
17	*SMS*	*Analysis, Software Maintenance*	*Requirement Traceability*
18	*Research*	*Software Maintenance*	*Software Organization, Software Notation, Software libraries*
19	*Survey*	*Analysis*	*Requirement Traceability, Trace Link*
20	*Research*	*Software Maintenance*	*Software Evolution, Software Product Line, Backward compatibility*
21	*Research*	*Analysis*	*Requirement Traceability*
22	*SMS*	*Across SDLC*	*Requirement Traceability*

23	*Research*	*Analysis*	*Requirement Traceability*
24	*Research*	*Coding*	*Software Traceability*
25	*Research*	*Software Maintenance*	*Software Traceability*
26	*SLR*	*Across SDLC*	*Software Traceability, Traceability Metrics, Evaluation, Benchmarking*
27	*Research*	*Coding*	*program comprehension, feature orientation, software traceability*
28	*Research*	*Software Testing*	*Software Traceability, Traceability Metrics*
29	*Research*	*Across SDLC*	*Visualization, Software Traceability*
30	*Survey*	*Project Management*	*Software Traceability*
31	*SMS*	*Coding, Software Maintenance*	*Software Traceability*
32	*Research*	*Coding, Software Maintenance*	*Software Traceability*
33	*Research*	*Coding*	*Software Traceability*
34	*Research*	*Coding*	*Software Traceability*
35	*Research*	*Software Testing*	*Defect detection, Issue detection*
36	*Research*	*Software Testing*	*Requirement Traceability*
37	*SMS*	*Software Testing*	*Software product lines*
38	*Research*	*Software Maintenance*	*Software Traceability*
39	*Research*	*Analysis, Coding*	*Software Traceability*
40	*SMS*	*Across SDLC*	*Software Traceability, Information Traceability*
41	*Research*	*Software Testing, Software Maintenance*	*Software Traceability*
42	*SLR + Research*	*Software Testing*	*Software Traceability*
43	*Research*	*Software Maintenance*	*Software Traceability*
44	*SMS*	*Software Maintenance*	*Software Evolution*
45	*SLR*	*Software Maintenance*	*Requirement traceability, Feature location*
46	*Research*	*Software Maintenance*	*change impact analysis, software traceability*
47	*Research*	*Software Maintenance*	*Software traceability*
48	*Research*	*Software Maintenance*	*Software Traceability*

49	*Research*	*Software Maintenance*	*Re-engineering, Software Product Line*
50	*Research*	*Software Testing*	*Software Traceability*

4. OBSERVATIONS FROM LITERATURE REVIEW

4.1 Segregation of Literature

The literature was first divided based on inclusion and exclusion criteria. The included papers were further classified according to the specific research focus they had addressed. There were research based literature as well as survey based literature.
Some of literature was also categorized in terms of providing future directions. These papers are torch-bearer for research which provides many research directions.

4.2 Software Traceability across SDLC

Source code is created at the coding stage of SDLC. There have been researches conducted to point out the forward as well backward traceability of source code artifacts. However, little work has been done to conduct a single traceability running across the different phases of SDLC. There are following challenges in this process:-

1. Lack of Standardized Format

Although formal specification have been evolved, still large amount of documents are still in textual format. As the SDLC phase changes the software artifact's and its corresponding documentation format also changes.

2. Lack of Interdisciplinary Vision in Software Evolution and SE Processes

The traditional software evolution and SE processes are constantly evolving. The requirements are no longer restricted to a particular skill-set of a domain and its corresponding development. The interdisciplinary vision is difficult to achieve due to the varied reasons associated with education and industrial norms. The Interdisciplinary vision is different from integration of multiple areas of research.

3. Lack of Integration with Multidisciplinary Areas of Research

Software Traceability already encompasses multiple areas of research as shown below:-
1. Software requirement
2. Software Product line
3. Software Maintenance
4. Software Configuration Management
5. Software Testing

Software Traceability can be integrated further with latest Continuous Integration (CI) and Continuous Deployment (CD) to create better vision for the new prospective software maintainers.
There is a need to develop following which will enhance integration prospects:-
1. Coding
2. Testing
3. Delivery and Maintenance

Once the software is ready after testing, it is delivered onsite. Onsite software needs constant maintenance due to constant changes in the environment wherein the software is deployed. The software maintenance engineer is tasked with introducing changes to software. There are four different models of software maintenance:-
1. Quick-Fix Model
2. Iterative-enhancement Model
3. Reuse-oriented Model

All of these models can be used along with traceability to see effect of changes being introduced in the software.

5. CONCLUSION AND FUTURE SCOPE

Traceability is also outside the scope of software engineering as it is being employed to achieve greater digital transformation sustainability and agility. It is essential part of developing a circular economy. The traceability approaches for non-IT domains can be integrated with the IT domain as it object trails across its production lifecycle.
There are also attempts to provide a visual framework for software requirement traceability [7]. These visual frameworks can be extended to provide integration of views across different phases of SDLC. The traceability approach can also be extended beyond software to include activities, processes, products to check conformance of different standards and rules.

References

1. Aung, T. W. W., Huo, H., & Sui, Y. (2020, July). A literature review of automatic traceability links recovery for software change impact analysis. In Proceedings of the 28th International Conference on Program Comprehension (pp. 14-24).
2. Okutan, A., Shokri, A., Koscinski, V., Fazelinia, M., & Mirakhorli, M. (2023). A Novel Approach to Identify Security Controls in Source Code. arXiv preprint arXiv:2307.05605.
3. Batot, E. R., Gérard, S., & Cabot, J. (2022, March). A survey-driven feature model for software traceability approaches. In International Conference on Fundamental Approaches to Software Engineering (pp. 23-48). Cham: Springer International Publishing.

4. Rahman, M. M., & Roy, C. K. (2021). A Systematic Review of Automated Query Reformulations in Source Code Search. ACM Transactions on Software Engineering and Methodology.
5. Kahraman, G., & Cleophas, L. (2022, September). A tool for modeling and analysis of relationships among feature model views. In Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B (pp. 103-109).
6. Davari, S., Jaberi, M., Yousfi, A., & Poirier, E. (2023). A Traceability Framework to Enable Circularity in the Built Environment. Sustainability, 15(10), 8278.
7. Madaki, A. A., & Zainon, W. M. N. W. (2022). A visual framework for software requirements traceability. Bulletin of Electrical Engineering and Informatics, 11(1), 426-434.
8. Ren, J., Liu, L., Zhang, P., & Zhou, W. (2019). A Method of Automatically Evolving Feature Models of Software Product Lines. IEEE Access, 7, 39253-39272.
9. Lyu, Y., Cho, H., Jung, P., & Lee, S. (2023). A Systematic Literature Review of Issue-Based Requirement Traceability. IEEE Access.
10. Liu, Y., Lin, J., Anuyah, O., Metoyer, R., & Cleland-Huang, J. (2022, May). Generating and visualizing trace link explanations. In Proceedings of the 44th International Conference on Software Engineering (pp. 1033-1044).
11. Pfeiffer, R. H. (2020, June). What constitutes software? An empirical, descriptive study of artifacts. In Proceedings of the 17th International Conference on Mining Software Repositories (pp. 481-491).
12. Florez, J. M., Moreno, L., Zhang, Z., Wei, S., & Marcus, A. (2022). An empirical study of data constraint implementations in Java. Empirical Software Engineering, 27(5), 119.
13. Zou, Y., Cao, Y., & Xie, B. (2018, September). An Exploratory Study on Codes in Heterogeneous Software Documents. In Proceedings of the 10th Asia-Pacific Symposium on Internetware (pp. 1-6).

14. Wang, H., Shen, G., Huang, Z., Yu, Y., & Chen, K. (2021). Analyzing close relations between target artifacts for improving IR-based requirement traceability recovery. Frontiers of Information Technology & Electronic Engineering, 22(7), 957-968.
15. Nicholson, A. (2019). Analyzing semantic trace links using network science and machine learning. McGill University (Canada).
16. Wang, H., Shen, G., Huang, Z., Yu, Y., & Chen, K. (2021). Analyzing close relations between target artifacts for improving IR-based requirement traceability recovery. Frontiers of Information Technology & Electronic Engineering, 22(7), 957-968.

17. Li, X., Wang, B., Wan, H., Deng, Y., & Wang, Z. Applications of Machine Learning in Requirements Traceability: A Systematic Mapping Study.
18. Ma, Y., Fakhoury, S., Christensen, M., Arnaoudova, V., Zogaan, W., & Mirakhorli, M. (2018, May). Automatic classification of software artifacts in open-source applications. In Proceedings of the 15th International Conference on Mining Software Repositories (pp. 414-425).
19. Abdeena, W., Unterkalmsteinera, M., Chirtogloub, A., Schimanskib, C. P., Golib, H., & Wnuka, K. (2022). Taxonomic Trace Links-Rethinking Traceability and its Benefits.
20. Venegas, L. M. O. (2023). Break the Code?: Breaking Changes and Their Impact on Software Evolution.
21. Sharma, P. (2017). Datasets Used in Fifteen Years of Automated Requirements Traceability Research. Rochester Institute of Technology.
22. Charalampidou, S., Ampatzoglou, A., Karountzos, E., & Avgeriou, P. (2021). Empirical studies on software traceability: A mapping study. Journal of Software: Evolution and Process, 33(2), e2294.
23. Lin, J., Poudel, A., Yu, W., Zeng, Q., Jiang, M., & Cleland-Huang, J. (2022). Enhancing automated software traceability by transfer learning from open-world data. arXiv preprint arXiv:2207.01084.
24. Pauzi, Z., & Capiluppi, A. (2021). Extracting and comparing concepts emerging from software code, documentation and tests. In 20th Belgium-Netherlands Software Evolution Workshop, BENEVOL 2021 (pp. Code-176287). CEUR Workshop Proceedings.
25. Theunissen, Theo, Uwe van Heesch, and Paris Avgeriou. "A mapping study on documentation in Continuous Software Development." Information and software technology 142 (2022): 106733.
26. Shin, Y., Hayes, J. H., & Cleland-Huang, J. (2015, May). Guidelines for benchmarking automated software traceability techniques. In 2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability (pp. 61-67). IEEE.
27. Krüger, J., Çalıklı, G., Berger, T., & Leich, T. (2021, March). How explicit feature traces did not impact developers' memory. In 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 610-613). IEEE.
28. Santos, L. R. J., Gadelha, G., Ramalho, F., & Massoni, T. (2020, October). Improving traceability recovery between bug reports and manual test cases. In Proceedings of the XXXIV Brazilian Symposium on Software Engineering (pp. 293-302).
29. Aung, T. W. W., Huo, H., & Sui, Y. (2019, September). Interactive traceability links visualization using hierarchical trace map. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 367-369). IEEE.
30. Hu, J. Y. (2019). Investigating the perceived value of software traceability in practice (Master's thesis).
31. Sulír, M., & Porubän, J. (2017, September). Labeling source code with metadata: A survey and taxonomy. In 2017 Federated Conference on Computer Science and Information Systems (FedCSIS) (pp. 721-729). IEEE.
32. Rodriguez, A. D., Cleland-Huang, J., & Falessi, D. (2021, September). Leveraging Intermediate Artifacts to Improve Automated Trace Link Retrieval. In 2021 IEEE

International Conference on Software Maintenance and Evolution (ICSME) (pp. 81-92). IEEE.
33. Cergani, E. (2020). Machine Learning as a Mean to Uncover Latent Knowledge from Source Code.
34. Lozano, A., Noguera, C., & Jonckers, V. (2016, March). Managing traceability links with matraca. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (Vol. 1, pp. 665-668). IEEE.
35. Nasser, M. (2017). Measuring Semantic Distances between Software Artifacts to Consolidate Issues from the Development and the Field. LU-CS-EX 2017-09.
36. Oberhauser, R. (2023). VR-SysML+ Traceability: Immersive Requirements Traceability and Test Traceability with SysML to Support Verification and Validation in Virtual Reality. International Journal on Advances in Software Volume 16, Number 1 & 2, 2023.

37. Petry, K. L., OliveiraJr, E., & Zorzo, A. F. (2020). Model-based testing of software product lines: Mapping study and research roadmap. Journal of Systems and Software, 167, 110608.
38. Mills, C., Bavota, G., Haiduc, S., Oliveto, R., Marcus, A., & Lucia, A. D. (2017). Predicting query quality for applications of text retrieval to software engineering tasks. ACM Transactions on Software Engineering and Methodology (TOSEM), 26(1), 1-45.
39. Hübner, P. (2016). Quality Improvements for Trace Links between Source Code and Requirements. In REFSQ Workshops.
40. Borg, M., Runeson, P., & Ardö, A. (2014). Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering, 19, 1565-1616.
41. Zogaan, W. A. (2019). Towards an Intelligent System for Software Traceability Datasets Generation. Rochester Institute of Technology.
42. Rahman, M. M. (2019). Supporting Source Code Search with Context-Aware and Semantics-Driven Query Reformulation (Doctoral dissertation, University of Saskatchewan).
43. Shafiq, S. (2022). Supporting the Triaging Process in Software Development/eingereicht von Saad Shafiq.
44. Tian, F., Wang, T., Liang, P., Wang, C., Khan, A. A., & Babar, M. A. (2021). The impact of traceability on software maintenance and evolution: A mapping study. Journal of Software: Evolution and Process, 33(10), e2374.
45. Razzaq, A., Wasala, A., Exton, C., & Buckley, J. (2018). The state of empirical evaluation in static feature location. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(1), 1-58.
46. Werber, K. (2022). Assessing Word Similarity Metrics for Traceability Link Recovery (Doctoral dissertation, Karlsruher Institut für Technologie (KIT)).
47. Mills, C., Escobar-Avila, J., Bhattacharya, A., Kondyukov, G., Chakraborty, S., & Haiduc, S. (2019, September). Tracing with less data: active learning for classification-based traceability link recovery. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 103-113). IEEE.
48. Mills, C., Escobar-Avila, J., Bhattacharya, A., Kondyukov, G., Chakraborty, S., & Haiduc, S. (2019, September). Tracing with less data: active learning for classification-based traceability link recovery. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 103-113). IEEE.
49. Krüger, J. (2021). Understanding the re-engineering of variant-rich systems: an empirical work on economics, knowledge, traceability, and practices.
50. Fucci, D., Alégroth, E., & Axelsson, T. (2022). When traceability goes awry: An industrial experience report. Journal of Systems and Software, 192, 111389.
51. Rasekh, Amir Hossein, Amir Hossein Arshia, Seyed Mostafa Fakhrahmad, and Mohammad Hadi Sadreddini. "Mining and discovery of hidden relationships between

software source codes and related textual documents." Digital Scholarship in the Humanities 33, no. 3 (2018): 651-669.

52. Ahmad, Masood, Mohd Nadeem, Mohd Islam, Saquib Ali, and Alka Agrawal1and Raees Ahmad. "Selection of Digital Watermarking Techniques for Medical Image Security by Using the Fuzzy Analytical Hierarchy Process." Recent Advances in Computer Science and Communications 16 (2023): 1-7.
53. Lacerda, Guilherme, Fabio Petrillo, Marcelo Pimenta, and Yann Gaël Guéhéneuc. "Code smells and refactoring: A tertiary systematic review of challenges and observations." Journal of Systems and Software 167 (2020): 110610.
54. Ampatzoglou, Apostolos, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies." Information and Software Technology 106 (2019): 201-230.

55. Kamei, Fernando, Igor Wiese, Crescencio Lima, Ivanilton Polato, Vilmar Nepomuceno, Waldemar Ferreira, Márcio Ribeiro et al. "Grey literature in software engineering: A critical review." Information and Software Technology 138 (2021): 106609.

56. Ampatzoglou, Apostolos, Stamatia Bibi, Paris Avgeriou, Marijn Verbeek, and Alexander Chatzigeorgiou. "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies." Information and Software Technology 106 (2019): 201-230.
57. Siahaan, Daniel, and Yenny Desnelita. "Structural and semantic similarity measurement of UML sequence diagrams." In 2017 11th International Conference on Information & Communication Technology and System (ICTS), pp. 227-234. IEEE, 2017.
58. Alenazi, Mounifah, Nan Niu, and Juha Savolainen. "SysML modeling mistakes and their impacts on requirements." In 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 14-23. IEEE, 2019.
59. John, Meenu Mary. "Design Methods and Processes for ML/DL models." PhD diss., Malmö universitet, 2021.
60. Mathew, George, Amritanshu Agrawal, and Tim Menzies. "Finding trends in software research." IEEE Transactions on Software Engineering (2018).
61. Yang, Yanming, Xin Xia, David Lo, and John Grundy. "A survey on deep learning for software engineering." ACM Computing Surveys (CSUR) 54, no. 10s (2022): 1-73.