IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 28, 2017, accepted May 11, 2017, date of publication June 19, 2017, date of current version July 7, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2710421

Multilingual Source Code Analysis: A Systematic Literature Review

ZAIGHAM MUSHTAQ, GHULAM RASOOL, AND BALAWAL SHEHZAD
COMSATS Institute of Information Technology, Lahore Campus, Lahore 54000, Pakistan
Corresponding author: Zaigham Mushtaq (zmqazi@gmail.com)

ABSTRACT Contemporary software applications are developed using cross-language artifacts, which are interdependent with each other. The source code analysis of these applications requires the extraction and examination of artifacts, which are build using multiple programming languages along with their dependencies. A large number of studies presented on multilingual source code analysis and its applications in the last one and half decade. The objective of this systematic literature review (SLR) is to summarize state of the art and prominent areas for future research. This SLR is based on different techniques, tools, and methodologies to analyze multilingual source code applications. We finalized 56 multi-discipline published papers relevant to multilingual source code analysis and its applications out of 3820 papers, filtered through multi-stage search criterion. Based on our findings, we highlight research gaps and challenges in the field of multilingual applications. The research findings are presented in the form of research problems, research contributions, challenges, and future prospects. We identified 46 research issues and requirements for analyzing multilingual applications and grouped them in 13 different software engineering domains. We examined the research contributions and mapped them with individual research problems. We presented the research contributions in the form of tools techniques and approaches that are presented in the form of research models, platforms, frameworks, prototype models, and case studies. Every research has its limitations or prospects for future research. We highlighted the limitations and future perspectives and grouped them in various software engineering domains. Most of the research trends and potential research areas are identified in static source code analysis, program comprehension, refactoring, reverse engineering, detection, and traceability of cross-language links, code coverage, security analysis, cross-language parsing, and abstraction of source code models.

INDEX TERMS Software engineering, reverse engineering, software design, software architecture, software maintenance.

I. INTRODUCTION

Source code analysis provides valuable information for architectural extraction, reverse engineering and reengineering of software applications. It helps in program understanding, software optimization, maintenance, and reuse. It is estimated that size of software in 2025 will be more than 1 trillion lines of code [11] that reflect the importance of source code analysis and manipulation in future. More than 29 applications of source code analysis for different domains of software engineering are highlighted by Kitchenham *et al.* [11]. The source code can be analyzed statically, dynamically or with a combination of both (Hybrid analysis) [5], [6].

Modern software applications are moving from homogenous single source code applications towards the heterogeneous multilingual environment. The applications of these heterogeneous multiple source code systems can be seen in web applications, enterprise applications (like J2EE) and complex embedded systems. The development paradigm is shifting from single language and technology to multiple languages and technologies. A number of applications and components developed in multiple language environments [19]. Various technologies are found in multiple languages in the form of heterogeneous applications (e.g., Java, XML, SQL etc.) [14], [15]. The source code analysis of these applications is an important task and is necessarily required in application optimization, reuse and in reverse engineering. In this paper, we identified and discussed different multilingual applications and analysis of these applications through a systematic literature review.

The focus of this paper is to present a comprehensive systematic literature review in the domain of multilingual source code analysis and its applications. A large number of

TABLE 1. Research questions.

Research Questions		Motivation
Q1	What are the key research issues for multilingual source code analysis (MLSCA)?	To understand the requirements and concerns in the current state of the art for the analysis of multilingual applications with respect to different software engineering domains.
Q2	What are the research contributions to address the problems of multilingual source code analysis?	To identify different approaches for the successful analysis of multilingual applications. The contributions are categorized in the form of analysis mechanism, the role of the study, type of evaluation and model representation etc.
Q3	What are the shortcomings and future of current research in multilingual source code analysis (MLSCA)?	To identify the research challenges and future prospects in the current state of the art focus for the analysis of multilingual applications.

studies presented on this topic before the last decade. It is important to collect, analyze, classify and summarize state of the art research. To the best of our knowledge, there is still no systematic review on multilingual source code analysis and applications of multilingual source code in the literature. This SLR highlights different features of the research in the field of multilingual source code analysis and applications developed with multiple technologies. To conduct an effective review study, we formalized basic search string to collect relevant research available in the domain of multilingual applications. We devised the assessment criteria, besides quality assessment criteria prescribed in [13]. We focused on different publications from renowned journals, conferences, and workshops. Based on the systematic review criteria, we finalized 56 research papers for further review and analysis, out of 3820 total papers. The selected papers are empirically and qualitatively evaluated through multiple aspects and presented in the form of different views. We found rising trends towards the development and analysis of multilingual applications, still, there is a need for a generic and extendable solution for analysis of multilingual applications.

We subdivided the study into four main sections. In Section II, we present research methodology for conducting systematic literature review by defining research questions, domain for literature review, source of information, search criteria, search string, information extraction procedure and study selection/assessment criteria. In Section III, we synthesize the selected papers and present the results of systematic literature review in the form of multiple summarized Tables (1-25). In the end, we present the conclusion of the whole study in Section IV.

II. RESEARCH METHODOLOGY

The goal of conducting this systematic review is to recognize and categorize the best available procedures, models, techniques and tools used to analyze multilingual applications. The process of systematic literature review helps in discovery and analysis of research available with the concerned domain of studies [10]. The existing research is empirically evaluated in accordance with predefined criteria. The results of the review provide scientific evidence by classifying the relevant studies. The systematic search procedure is provided by Kitchenham *et al.* [9] and the selection of primary studies method followed in [10]. We also selected guidelines for writing a literature review from Pautasso [41]. The population is composed of publications found in the selected sources which apply procedures or strategies related to analyzing multilingual applications.

A. REVIEW PROTOCOL

The development of review protocol is the 1st step towards systematic literature review. The SLR protocol helps to designate the search plan in the form of search strategies for the extraction of relevant literature. This process includes research questions, research scope, source of information, inclusion & exclusion criteria and literature assessment criteria. The process of conducting the systematic literature review follows the steps mentioned in Fig 1.

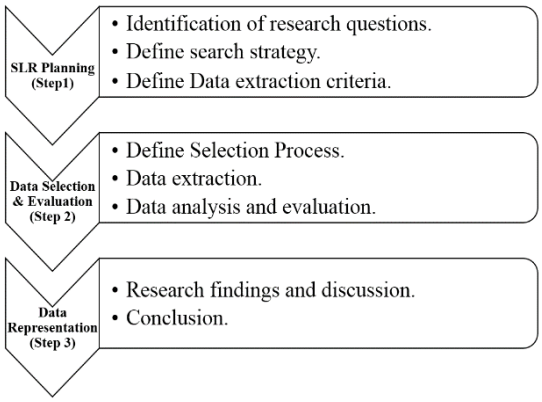

FIGURE 1. Systematic literature review process.

1) RESEARCH QUESTIONS

In order to conduct a systematic review, it is essential to formulate the primary research questions. After specifying the research questions, the review procedure involves building the search strategies to recognize and extract relevant studies [8]. The answers to these research questions are searched in the published literature using the procedures of systematic literature reviews as proposed by Kitchenham [10] and DARE/CDR criteria [13]. The basic intent of this review is to summarize the current state of the art research in MLSCA (Multilingual Source Code Analysis) domain and to identify efficient techniques used for MLSCA. We searched for MLSCA techniques that were empirically evaluated to identify needs for future research. The research questions are developed to evaluate the significance of multilingual applications as mentioned in Table 1.

TABLE 2. Keywords for the search term.

Terms	Keywords and Alternate keywords
Multi*	Multi, Multiple, Multilanguage, Multilingual, etc.
Cross*	Cross Language, Cross Lingual, Cross Links, Cross Source Code etc.
Hetero*	Heterogeneous, Heterogeneity, Heterogeneously, Heterogeneousness.
Hybrid*	Hybrid System, Hybrid Components, Hybrid Languages, Hybrid Software, Hybrid Source Code etc.
Inter*	Inter System, Interoperability, Interoperability links, Inter dependencies.
depend*	Dependencies, depend, dependent etc.
Language	Programming language, Languages, Domain-specific Languages etc.
Artifacts	Cross-language Artifacts, Multilingual Artifacts
Analy*	Analysis, Analyzing, Analyze etc.
Recover*	Recovery, Recovering, Recover etc.
Revers*	Reverse, Reverse Engineering, Reversing etc.
Invest*	The investigation, Investigate etc.
Synthes*	Synthesis, Synthesizes etc.
Detec*	Detection, Detect, Detected etc.
Discov*	Discovery, Discover, Discovered etc.
Software*	Software tools, Software Systems, Software Code etc.
Source*	Source Code, Source Code Files, Source Code Links, Source Code Analysis etc.
Application	Software Applications, Multilanguage Application, Multilingual Application
Program*	Programs, Programming, Programmed etc.
Pars*	Parsers, Parsing, Parsed etc.

2) SEARCH STRATEGY

A well-organized research is required for extracting appropriate information and filtering irrelevant studies from focused research areas.

The planning and formulation of effective search is an important step to finding out the meaningful research available in the respective domain. We followed both automatic and manual search mechanism for the exploration of the search term. At first, we performed automatic search followed by the manual search is executed. The automatic search is based on search string and is performed on search engines of relevant electronic data repositories. The purpose of the manual search is to gather more literature relevant to multilingual source code analysis domain. The manual search includes reference lists of relevant primary studies and gray literature. In order to ensure the extraction of relevant information, we limit our search terms on following conditions.

- Identification of major search keywords, based on formulated research questions. -
- Search for alternate words and synonyms for major keywords.
- Developing a search string by combining keywords with Boolean operator ''AND'', and alternate keyword with Boolean operator ''OR''.

Search Term: The search term is the combination of keywords that precisely returns the relevant literature from a large number of studies. In order to ensure the reliability of search term, we analyzed the main concepts and terminologies in the domain of multilingual source code analysis. We recognized keywords used in the literature that is related to research questions. The meta-sentence for this literature review contains **"Analysis of Software Systems** developed using the **source code** of**Multiple** Programming **Languages"**. We finalized initial keywords in Table 2 mentioned below required to be incorporated in search term relevant to research questions.

After describing the keywords, we considered synonyms, alternatives, and hypernyms for each keyword. The resulting data was discussed with local researcher's community. In order to formulate an effective search string aligned with the Metasearch sentence, the finalized keywords were concatenated with Boolean operators ('AND' and 'OR') and wildcard character ('*'). The synonyms were combined with the help of 'OR 'operator. The use of wildcard allows the consideration of multiple alphanumeric characters as an alternative of keywords. The use of OR operator provides an additional search space, whereas, the AND operator reduces the search space to be more specific with the relevant papers. For example, the first pair of ORs combines several keywords for querying all the literature that refers to aspect **"Many/Multiple"** in the domain of source code analysis.

The search term is subdivided into three components. The first part of the search term is related to the population for Multilingual or Multilanguage applications, the second part is related to their analysis and the third part corresponds to the source code of applications. For the effective query, we need to ensure the existence of all three components in a search term, therefore we applied AND operators among the components. In order to ensure that completeness of search results we applied OR operator in synonyms and relevant key terms. Finally, the search string with the combination of Boolean operators and the wildcard is mentioned as.

((Multi*) OR (Cross*) OR (Hetero*) OR (Hybrid*) OR (Inter*) OR (*operability*) OR (*depend*) OR (*Lingual) OR (*Language*) OR (Artifact)) AND ((Analy*) OR (Recover*) OR (Revers*) OR (Invest*) OR (Synthes*) OR (Detect*)) AND ((Software*) OR (Source*) OR (Application) OR (Program*) OR (Pars*)).

3) LITERATURE RESOURCES

We conducted the primary search from online research databases and search engines (IEEE explore, Springer, ACM

TABLE 3. Attributes of the research study.

Attributes	Sub-Attributes	Description
General Detail	Paper #, Publication type, Title, Authors, Publication Year, Type, Institute, Country, Proposed Technique/ Model/Tool, Research Domain, Environment, Website/email, Implementation, Languages, Case Study	Describe the corresponding details related to the research publications.
Research Publications & Proceedings	Paper #, Research Title, Publisher, Publication Type, Proceeding Year, Origin, Research Domain, Brief, Focus of study	Provide complete information about the selected research proceedings.
Multilingual Applications Tools	Paper #, Technique, Tool, Model, Analysis mechanism, Language Support, Experimental Case Study	Provide complete information about tool support of multilingual applications.
Assessment Criteria	Paper #, Problem definition, Proposed Solution, Findings/ Benefits/ Contributions, Weakness/ Future work	The selected papers are evaluated on the basis of research purpose, importance, contribution and shortcomings.
Conference & Workshop Proceedings	S. #, Acronyms, Conference/Workshop Name	Describe the type of the research proceedings along with the abbreviations.
Detail of Publications with the Origins	Papers #, Country, Publishers, References, Papers, Publisher name, References, Total Papers, Journals, Conferences, Books, Workshops	Describe the details of the selected research papers.
Status of Selected Publications	Papers Selected on Search Criteria, Papers After Step-I Criteria, Papers After Step-II Criteria, Papers After Step-III Criteria	Shows the number of research papers selected after applying the selection criteria.
Selected Publications	Total papers, Journals, Conferences, Workshops, Books	Represents number of publications

Digital Library, Science Direct and Web of knowledge), journal publications and conference proceedings. We also used Google and Google Scholar, however, the extracted results were also identified by the existing databases. Therefore, the results were not accumulated to overall count.

4) DOMAINS FOR SYSTEMATIC REVIEW

This systematic review focuses on the three domains, including Web-based Applications, Embedded Applications, and Enterprise Applications. The rationale to select these domains was to categorize the huge literature fetched during an initial search with the help of peer reviews. The factors to select these applications include trends of software development, domains with more extensive work and relevancy of the domain with cross-language analysis.

5) THE FOCUS OF THE STUDY

This process includes the extraction of related information by considering the type of study, important ideas, key factors and significant strategies in each study which is essentially required to establish objective and subjective results [41]. The study methodology is followed in [9]; these categories contains case studies, literature reviews, experiments, simulations, and surveys.

6) KEY RESEARCH ATTRIBUTES

The information is extracted for further analysis from each research paper in the form of specified template. Table 3 presents the attribute specified for the extraction of information from primary studies.

7) STUDY INCLUSION/EXCLUSION CRITERIA

The initial search criterion is set to extract maximum publications in MLSCA domain. To ensure most relevant research, the publication period from January 2001 to January 2016 is considered. The database fields of title and abstract are searched from previously mentioned resources. Kitchenham [10] recommended that exclusion based on languages should be avoided [19]. However, only papers written in English are included. Following criterion is determined to evaluate and select the research publications.

- The valid range of the research publication years must be from January 2001 to January 2016.
- The research papers must be of full-length papers.
- The selected papers must be relevant to the specified search string.
- Research papers must be written in English.
- The research must address the assessment criteria.

B. SELECTION PROCESS

Pautasso [41] proposed the research inclusion criteria. This criterion corresponds to the relevancy of the study with the research domain. After analyzing inclusion criteria, the exclusion criterion is applied by removing the studies which discuss the domain of MLSCA but do not provide any significant research contribution. The search procedure produced 3820 initial studies. Out of these studies, 163 are selected as being relevant, and 56 are selected as primary studies (the complete list of primary studies is shown in Table 20 (Appendix). Table 4 shows the distribution of studies found according to the sources used. In order to obtain independent assessments, three steps selection process is conducted, as illustrated in Fig 2.

1) TITLE BASED SEARCH

In the first stage, duplicates and irrelevant papers are excluded manually based on titles. In our case, the share of irrelevant papers was extremely large because research related to MLSCA cannot be distinguished from papers in the database search. After the first stage, only 163 papers remained for next phase.

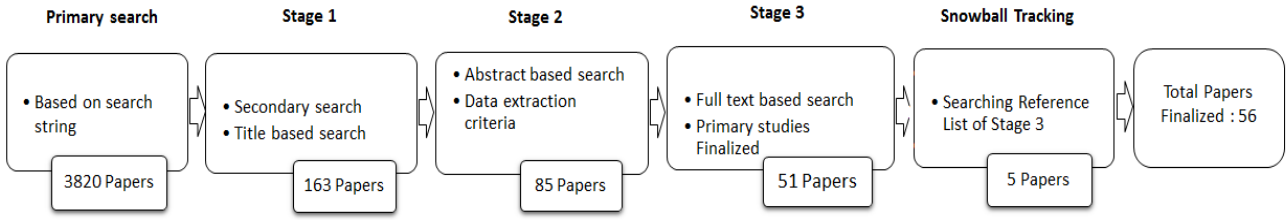

FIGURE 2. Selection process.

TABLE 4. Selection process.

Stages	Selection Criteria	Description
Pre-Stage	Primary search	String based Search from relevant journals, conferences, and workshops etc.
Stage 1	Title based search	Select potential primary studies.
Stage 2	Abstract based search	Extract primary studies.
Stage 3	Full text-based search	Critical empirical evaluation.

2) ABSTRACT BASED SEARCH

In the second stage, information in abstracts is analyzed and the papers are classified along with research approach for the analysis of multilingual applications. Research approaches include experiments, case studies, surveys, reviews, theories, and simulations. At this stage, we do not judge the quality of the empirical data. After this stage, 85 papers remained in the list.

3) FULL TEXT-BASED ANALYSIS

The empirical quality of the papers is completely evaluated at this stage. A full text based analysis is executed on the remaining 85 papers. The evaluation criteria involve DARE (Database of Abstracts of Reviews of Effects) and CDR (Centre for Reviews and Dissemination) criteria [13]. In order to conduct final data extraction following research questions are defined.

- Are the review's inclusion and exclusion criteria described and appropriate?
- Is the literature search likely to have covered all relevant studies?
- Did the reviewers assess the quality/validity of the included studies?
- Were the basic data/studies adequately described?

All the questions are assessed and scored on the following criteria prescribed in [8]: Y (yes): the criteria clearly defined in the research, P (Partly), the criteria partially defined; N (no) the criteria not defined in the research. Each questioned is scored as $Y = 1$, $P = 0.5$, $N = 0$. The trend of the selection process is generally more inclusive. Only irrelevant papers are barred.

Snow Ball Tracking: After applying all these filters, we applied snowball tracking by searching through reference list of each finalized study and ensured that no important study was missed. It is important to mention that we finalized these papers after assessing the exclusion/inclusion criteria and quality assessment criteria. After applying the snowball tracking we identified 5 more studies and totally added up to 56 primary studies. The result of 56 finalized papers are presented in Table 20 (appendix).

III. DATA ANALYSIS AND RESULTS

This section summarizes the results and provides the descriptive evaluation of each study in a tabular format. This section discusses the problem definition, proposed solution, strengths and weaknesses of the included studies. Based on research evidence results, conclusion and recommendations are drawn.

A. EVALUATION OF QUALITY ASSESSMENT CRITERIA

This is the most important section as prescribed in Section II for a full text-based search that determines the quality of research papers and relevancy with the research topic. The Q1 is about inclusion and exclusion criteria and the results show that 80.2% described criteria is appropriate. The Q2 is about the relevancy of literature with the selected papers, the result shows that 55.7% of presented literature is appropriate to the subject. The result of Q3 shows 61.3% relevancy of quality and validity of the included studies. The result of Q4 highlights 64.2% adequacy of the selected papers with the research topic. The overall quality assessment of the selected papers is 65.4% which is quite healthy. The overall quality assessment score of primary studies finalized is mentioned in Fig 3, whereas the individual quality assessment result of finalized studies is shown in Table 18 (appendix).

B. SEARCH RESULTS

We finalized 56 papers in stage-III out of 3820 searched papers (shown in Table 20 appendix), published during 2001 to 2016. The publications in the form of different conferences, workshops and journals from renowned (publishers) digital libraries are considered for consultation, as mentioned in the research criteria. The selection ratio of finalized papers includes 60% from IEEE explore, 15% from Springer, 15% from ACM, 6% from Science Direct and 4% from IET and JUSC Journals. All the stage-III papers are evaluated rigorously through assessment criteria and quality assessment criteria formulated in the form of research questions. The status of the papers along with the ratio of selection is mentioned in Table 5.

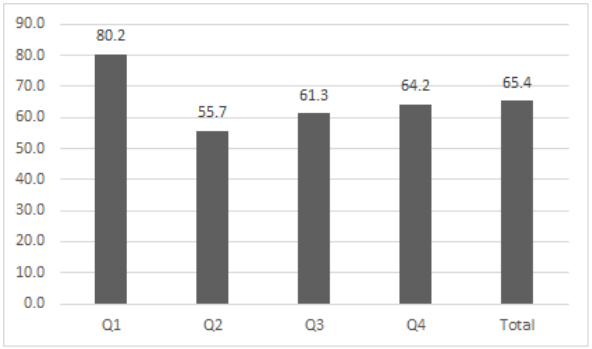

FIGURE 3. Quality assessment score (Percentage).

TABLE 5. Status of publications.

Sr. #	Source of Digital Library	Pre-Stage	Stage1	Stage 2	Stage 3	Snowball	Ratio %
1	IEEE Explore	375	88	36	32	2	61
2	Springer	1457	28	18	8	-	14
3	ACM	822	22	17	8	-	14
4	Science Direct	1166	25	14	3		5
5	IET Journal	-	-	-	-	2	4
6	JUCS Journal	-	-	-	-	1	2
	Total	3820	163	85	21	5	100

TABLE 6. Selected publications and proceedings.

Sr. #	Type	Total	Journals	Conferences	Workshops
1	IEEE	34	1	30	3
2	ACM	8	0	6	2
3	Springer	8	1	7	0
4	Elsevier	3	3	0	0
5	IET	2	2	0	0
6	JUCS	1	1		
	Total	56	8	43	5
	Ratio %		14	77	9

The detail of the papers after final selection is listed in Table 6. Most of the selected papers (34) are from IEEE Explore i.e. 61 %, whereas most the papers (43) are from conference proceedings i.e. 77%.

The Table 19 (appendix) elaborates the selected papers in the form of their origin, publishers, and type of research proceedings. The multiple research publications are also shown. Moreover, the categories (i.e. journal, conference, book or workshop) of the research publications are also mentioned in Table 19 (appendix).

Fig 4 represents the number of publications w.r.t. year of publication. It is observed that majority of the publications i.e. 42 out of 56 starts from 2010 (75%) and out of these publications, 30 are from 2012 onwards, in which most of the proceedings published during 2013 to January 2016 i.e. 25/56, 45%.

The detail information of each research publications and proceedings is provided in Table 20 (appendix) in the form of Paper ID, Reference Number, Title of Selected Research

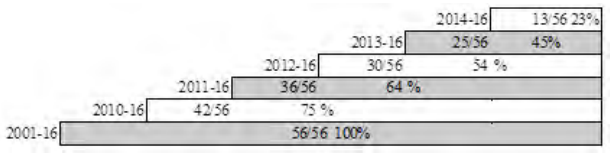

FIGURE 4. Year-wise distribution of papers.

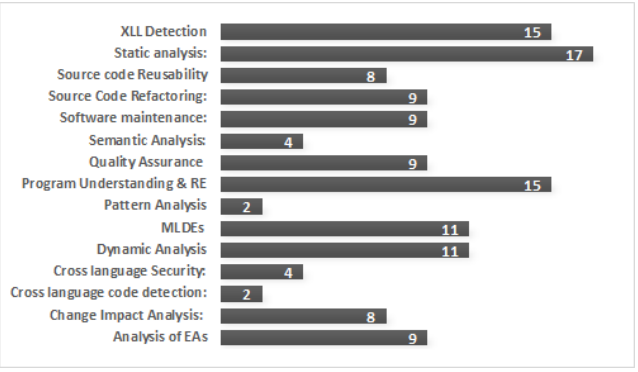

FIGURE 5. Domain wise detail of research issues & requirements.

Publications and Proceedings, topics/domain of the research study and Publication Venue. Each paper supports different and multiple domain and topic of study with respective platform and environment.

C. ASSESSMENT AND DISCUSSION OF RESEARCH QUESTIONS

Contemporary software applications are composed of multiple programming languages. The analysis of these applications is a key challenge for the software community. The importance of source code analysis in modern applications cannot be ignored [11]. Its significance is proved and demand is increasing with the increase in the size of the source code [21], [48]. In this section, we analyzed the finalized 56 primary studies based on our research questions. After the analysis of selected studies, we extracted the facts from diverse research domains of MLSCA. We discuss question wise assessment of the extracted information in this section.

1) ASSESSMENT OF Q1: WHAT ARE THE RESEARCH PROBLEMS/REQUIREMENTS FOR MLSCAs?

In this section, an insightful knowledge of the issues and requirements for the analysis of multilingual applications (MLAs) is presented. There are 46 different problems and requirements reported which are categorized in 13 software engineering domains. Each problem or requirement is marked by PR # (Problem/Requirement number). Table 7 provides the complete information about the concerns of multilingual source code analysis. We also mapped problems and requirements with their respective domains in Table 7.

Detail of issues and requirement for multilingual source code analysis (MLSCA):

In this subsection, domain-wise detail of issues and requirements are presented. Fig 5 represents the percentage

TABLE 7. Problems and requirements for multilanguage source code analysis.

Problems and Requirements (PR)		Domains												
		Dynamic Analysis	Quality Assurance	Source code Reusability	Change Impact Analysis	XLL Detection	Software maintenance	Static analysis	Source Code Refactoring	Cross language Security	Program Understanding & RE	Multiple Language Development Environments	Analysis of Enterprise Applications	Semantic Analysis
Inter System Interaction	PR1	1												
Extrarction and Analysis of Trace Links	PR2	32,39, 47												
Detection, Analysis and Managing code clones	PR3	39		66										
Integration of Cross Language Artifacts (XLA)	PR4	59						11				46		
Homogeneous Analysis Techniques	PR5		2		4	36, 49		22						
Concurrency Issues in MLAs	PR6		30											
Cross-language bug localization	PR7		62											
Complex Nature of MLAs	PR8		62	11			21	22			21,31, 33	54	42, 50	
Searching Relevant Source-code Snippets	PR9			3										
Analyze behavior & static verification	PR10			64									42, 50	
CIA in MLAs	PR11				4,29, 31						31			
Context Awareness Across MLAs	PR12				4									
Language independent change prediction & propagation	PR13				7		7						7	
Generic Approach for XLL specification	PR14					34, 49								
Cross Language Artifacts binding	PR15					35								
Identification & Refactoring XLLs	PR16					36			36		36			
Hidden Dependencies in XL Artifacts	PR17					44							44	44
Analyzing XLR in IDEs	PR18					46			61			45		
Cross Language Dependencies in MLAs	PR19					44, 50								
Developing XLL Rules and spec	PR20	58				58								
Low Productivity, Quality, Robustness & Scalability	PR21		30			58	65							37
High maintenance cost of MLAs	PR22		68				65	68			65			
Low dev. Cost & short time to market	PR23			20			20							
Intermingled source code & syntax	PR24	32						55						
Analysis of Large ML SW corpora	PR25							12						
Robust/ Multi Language Parser	PR26							25, 55, 68						
Automatic analysis, visualization	PR27							56						
SW Metrics	PR28							56						
Analysis of Web applications	PR29	39					20	60		16, 26	24			
XL support for DSLs	PR30							53						
Analysis & transf. of MLAs	PR31							40						
Automated MLR	PR32								51, 57					
DB Schema Modification in MLAs	PR33								52			52		
Cross Language refactoring	PR34								57			61		
XL Vulnerabilities & Information Leaks	PR35									16, 26				
Automated RE of MLAs	PR36										33		31	
RE of Legacy Applications	PR37										23			
Understanding Topology of the Web Aggregates	PR38										24			
Analysis of MLAs at Arbitrary Level of Granularity	PR39										17			
Support of ML in IDEs	PR40											54		
Portability Concerns of MLAs	PR41											27		
Insufficient Support of EAs	PR42												31	
Incorrect Transactional Scope	PR43												42	
Hidden Inconsistencies across ML Artifacts	PR44												43	
Analysis of multi commit mixed source code HAs	PR45							63						
XL code Identification and classification	PR46							67, 18						

of research issues and requirements of MLSCA which are alienated with software engineering domains. Most discussed issues include Static Analysis 17%, Program Understanding & Reverse Engineering 15%, Cross Language Link Detection 15%, Dynamic Analysis 11%, Analysis of EAs 9%, Multiple Language IDEs 11%, Refactoring 9%, Source Code Reusability 8%, CIA 8%, Quality Assurance 9%, SW Maintenance 9%, Cross Language Security 4% and Semantic Analysis 4%. Following domains comprehensively describe the problem and requirements from selected research.

a: DYNAMIC ANALYSIS OF MLAs

Complex heterogeneous systems are composed of multiple subsystems that are interdependent and interact with each other. The analysis of these applications is quite helpful in better program understanding, re-engineering and reverse engineering. The overall behavior of the system cannot be studied unless we understand about how these components interact with each other [1].

Due to complex and dynamic nature, the analysis of multilingual applications has become difficult and challenging [32]. Understanding interaction of multiple languages is difficult because of a large number of artifacts without integration and tool support [59]. The approaches at present can only extract and develop views only for the execution of the single system. The tool supports to recover views of application-level interaction behavior in complex heterogeneous systems is not available [1]. Dynamic WAs (web applications) are composed of intermingled source code of multiple programming languages (e.g., HTML, PHP, JavaScript, and CSS), which makes them complex and difficult to analyze and manage clones [39].

Traceability across multilingual artifices is another hot issue. Tractability analysis helps to spot the affected artifacts in the form of trace links. Traditional techniques are deficient to support MLAs. Automatic tracking of object relations is required to support co-evolution of multi-language software systems [47]. The execution traces of heterogeneous applications like Ajax based applications are difficult to analyze [32].

b: QUALITY ASSURANCE OF MLAs

Quality assurance and debugging is an essential requirement for software applications. The testing and debugging of distributed MLAs are challenging [2]. The existing software observational methods only support homogeneous applications and are less scalable to support MLAs. These techniques suffer from concurrency issues. An effective tool support is required to ensure the quality of cross-language software applications [30]. At present, the heterogeneous applications (HAs) suffer from cross-language bug localization problem. Current bug fixing techniques support homogeneous applications and in the existing research, the bug reports do not appear in the source code of HAs [62].

c: SOURCE CODE REUSABILITY

Modern software paradigm is focused towards cross language applications. Due to the complex nature of these applications, analyses, modifications, and reusability have become difficult and challenging [11]. A little support is available to analyze behavior and static verification of these systems [64]. These applications require multi-language reusable components, tied though different types of files. All possible instances of source code are required to be detected for reusability across MLAs. Moreover, the reusability of cross-language source code requires compiler support [66]. Searching relevant source-code snippets is another challenge, which is required for software reusability [3]. Existing systems necessitate an existing repository of relevant code samples. However, for many libraries, such a repository does not exist. Source code recommendation mechanism is required because many libraries lack API documentation of reusable components.

d: CROSS-LANGUAGE CHANGE IMPACT ANALYSIS (CIA)

Change propagation & impact analysis across the multilingual artifacts is an essential requirement to understand MLAs [31]. Predicting change propagation in the source code is a key challenge in analysis and maintenance of multilingual enterprise applications [7]. A language-independent technique is required for evaluating the impact of change, maintaining code history and preventing errors [7]. The existing CIA techniques only support single languages and lack context awareness [4]. These approaches cannot fully analyze the heterogeneous artifacts of different languages [29].

e: CROSS-LANGUAGE LINK DETECTION

The cross-language links (XLLs) are helpful in better program understanding, maintenance, error handling and refactoring of MLAs. Modern software applications, for example in Java Enterprise Applications (JEAs), are composed of cross-language artifacts which are interdependent to each other. These artifacts are referred through semantic links. However, the relationships among these artifacts are not organized and accustomed with hidden dependencies [44].

The detection and managing dependencies in large MLAs is quite hard and challenging due to their complex and heterogeneous nature [50]. There is no scalable, robust, general approach available for cross-language dependency detection. The existing techniques focused on single source code applications [49]. A generic approach is required to specify cross-language links (XLL) [34].

The available analysis tools are language specific. It is difficult to identify and refactor the cross-language links among the artifacts developed in a heterogeneous application [36]. A slight change in the code may affect the behavior of the application. Developing XLL rules and specification is a major challenge for research community [58]. Moreover, there is no standard approach available for binding multi-language artifacts [35]. The deficiency of XLLs across MLAs damages productivity & stability of the software [58]. Cross-language references and relationships are also a key issue in the development of cross-language development environments (IDEs). Existing IDEs are deficient to completely analyze cross-language relations [46].

f: SOFTWARE MAINTENANCE

Modern web applications (WAs) are composed of heterogeneous components. Due to low cost and a short time to market trends in software development, the maintenance and reusability have become challenging [20]. Another aspect of MLAs is the maintenance cost. As the size of the

application is increased the cost of maintenance is also increased, decreasing the quality & life of SW. Therefore, tools are required for efficiently understanding MLAs globally [65].

g: STATIC ANALYSIS

As the software applications have become more diverse and heterogeneous, their static analysis is more demanding and difficult [11], [12], [22], [67], [68]. A multilingual application is composed of multiple languages that exhibit diverse functionalities. Therefore, monitoring quality during software development and maintaining consistency in these applications is quite critical [68]. At present cross-language analysis, support is available only for homogeneous applications. Analyzing large software corpora [12] and clone detection of MLAs is difficult [22]. The analysis, visualization, and generation of software metrics of MLAs can only be cost effective by using automatic analysis tools that support the generation of source code metrics, dependency graphs, and software evolution analysis [56].

The analysis of MLAs at a high level of integration is challenging [11]. The existing tool support for source code analysis is insufficient. They support few revisions and deficient to address large scale multi-language applications [63]. The available tools either focus analysis or transformation. A combination of both analysis and transformation techniques is required [40].

The available cross-language code detection approaches [18] are compiler dependent and only support monolingual comparison. The major challenge in analyzing MLAs is to build a separate parser for each language participating in an application [25]. For example, web applications contain intermingled syntax of multiple source code languages. Therefore, it is difficult to parse source code of these languages. A robust multilingual parser is required to concurrently handle the source of multilingual applications [55].

The modern web application contains multi-language dynamic pages, jumbled with each other. Static analysis of web applications (WAs) is hard & challenging due to the presence of dynamic HTML code & interaction of multiple languages in an application [60].

Sugar libraries are a unique approach that extends the syntax of a programming language within the language. Extending syntax of programming language to support multiple domains with sugar libraries is quite hard because they cannot be used as the main extension mechanism of the programming language [53]. The support of a large number of DSLs within a host language is required.

Source Code Language Identification (SLI) techniques are used to identify multi-language and embedded code applications. These techniques use meta-information for source code identification and classification. However, using meta-information is not always precise enough. Moreover, SLI techniques require a parser for each language that affects the correctness, performance and increase the maintenance cost [67].

h: SOURCE CODE REFACTORING

Modern software applications are composed of multiple language artifacts that interact with each other. The automated multi-language refactoring (MLR) is not possible due to the different artifact types and modified definition of semantics [51]. The present refactoring tools are language-dependent that resist the smooth integration of development environments [57].

Database refactoring or simple schema modification in MLAs is challenging. SQL modifications are made manually to adapt their applications [52]. The cross-language link frameworks do not support object oriented code refactoring when DB schema changed [52]. Cross-language refactoring (XLR) support is required in modern multi-language development environments (MLIDEs) because the existing IDEs only support refactoring of single languages [61].

i: CROSS-LANGUAGE SECURITY

As the size and complexity of the web applications are increased, the vulnerabilities across the applications are also increased. There is a need of an automated source code reviewer to deal security vulnerabilities as the manual solutions are slow, expensive and insufficient [26]. In [16], the challenge of language-based security is discussed. A precise algorithm for language-based security is required that checks programs for information leaks.

j: PROGRAM UNDERSTANDING & REVERSE ENGINEERING (RE)

SW development trends require more efforts in understanding legacy applications that promote re-engineering & reverse engineering. Extracting software change information of different multilingual contents is essentially required in order to understand MLAs [31]. It is difficult to analyze HAs at an arbitrary level of granularity [17], which is essential for program understanding, re-engineering and reverse engineering. It is challenging for the software community to reverse engineer existing the legacy applications for source code improvement, evolution and modernization software applications [23]. The understanding topology of the web aggregates in WAs is another issue. A tool support is required for extraction, analysis, and visualization of aggregates for large hypertext web applications [24]. There is a desperate need for a framework or tool for automatic reverse engineering of complex heterogeneous applications. The available techniques to reverse engineer object-oriented applications cannot reverse engineer modern Enterprise applications [33].

k: MULTIPLE LANGUAGE DEVELOPMENT ENVIRONMENTS (MLDEs)

The present Integrated Development Environments (IDEs) do not support the development of multi-language systems. Their support for multiple languages is weak because they are language specific and cannot process cross-language applications [54]. Modern languages broadly use platform specific

APIs, causing interoperability concerns. They need to support portability of multiple languages across the platforms [27].

Existing development environment do not completely support relation across multi-language artifacts [46]. They do not visualize cross-language relations, deficient in static checking for consistency of cross-language and cannot offer refactoring of artifacts in different languages [45]. Moreover, modern IDEs do not support database refactoring or simple schema alteration [52]. A true multilingual IDE must cater Cross-language refactoring (XLR), multi-language meta information and interlanguage containment. A system is required that integrates IDE support across language boundaries. Common IDEs only support single language features [61].

l: ANALYSIS OF ENTERPRISE APPLICATIONS (EAs)

The information in large EAs is distributed across various components and their relationships. The analysis and reverse engineering of large EAs has become hard & challenging for the research community. The existing object-oriented reverse engineering techniques cannot support EAs [31]. The heterogeneous nature of EAs is difficult to analyze and verify desirable properties and architectural constraints [42], [50]. They may leave confusion in the form of conceptual errors in source code and particularly may conceal incorrect declarations of transaction scope. The developer's loose overview of the system and hide inconsistencies in the system [43].

m: SEMANTIC ANALYSIS

The clustering techniques for Semantic Analysis of MLAs uses similar feature types for estimating the distance between source code elements. The available techniques do not produce good quality results in absence of adequate inputs [37].

2) ASSESSMENT OF Q2: WHICH ARE RESEARCH CONTRIBUTIONS TO ADDRESS THE PROBLEMS OF MULTILINGUAL SOURCE CODE ANALYSIS?

This section is an important part of the literature review. In order to answer the research question, we conduct an in-depth analysis of each selected paper and extracted the proposed solutions and benefits of each research problem for MLSCAs. The research findings are presented in a more concrete way in the form of tool support, approaches, and surveys for analyzing MLAs. Moreover, language support for analyzing MLAs and domains of MLSCA are also presented.

a: SOURCE CODE LANGUAGES

In this subsection, we recognized 40 different kinds of languages including General Purpose Languages (GPLs), Domain Specific Languages (DSLs), Meta Programming Languages (MPLs) and Intermediate Language Representations (ILRs) (as mentioned in Fig: 6). The number of languages varies from 2 to 9 languages per research publication. Most commonly supported languages in the research publications include OOPLs (30 occurrences, 57%), Java/J# (28 occurrences, 53%), HTML/Applets (14 occurrences, 26%), Domain Specific Languages (11 occurrences, 21%),

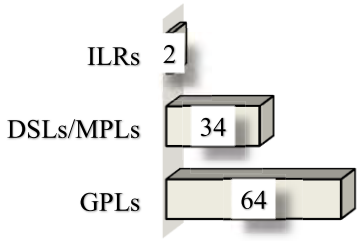

FIGURE 6. Distribution of languages (Percentage).

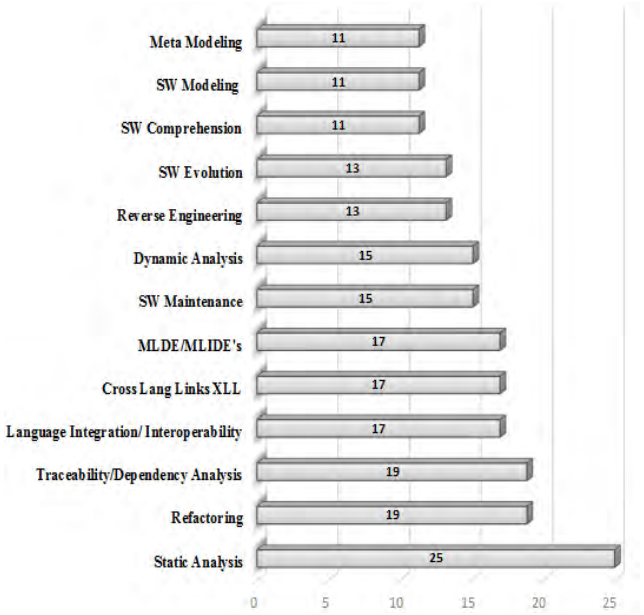

FIGURE 7. Distribution of MLSCA domains (Percentage).

C/C++/C# (9 occurrences, 17%), Jscript (9 occurrences, 17%), JEA/EJB (6 occurrences, 11%), SQL/RDBMS (7 occurrences, 13%) and XML (7 occurrences, 13%) etc. The ratio of remaining language representation is less than 10 %. The detail of source code languages is given in Table 9 (appendix).

b: DOMAINS FOR SOURCE CODE ANALYSIS

Multilingual source code analysis (MLSCA) is a quite diverse domain and its importance is realized in almost every domain of software engineering. In order to specify the research trends and contributions, we separately discuss research domains in multilingual source code analysis (MLSCA). The research contributions are grouped and categorized into 34 software engineering domains. Each model proffers separate analysis mechanism and support platform to accomplish their requirements. It is found that the research trends among all these domains, 62.3% (33 occurrences) of the population of the selected research studies relate to cross-language linking, dependency analysis, integration,

traceability, change analysis and interoperability. Some of the most important domains include Dynamic Analysis 15%, Language Integration/ Interoperability 17%, Meta-model 11%, Multilanguage development environment/ IDE 17 %, Pattern Analysis 6%, Program Understanding 9%, Re-engineering 6%, SW Refactoring 19%, Reuse 6%, Reverse Engineering 13%, Static Analysis 25%, SW Comprehension 11%, SW Evolution 11%, SW Maintenance 15%, SW Modeling 11%, Traceability/Dependency Analysis 19% and Cross-Language Analysis (XLL) 17% etc. Figure 7 shows the distribution of Multilanguage source code analysis domain (MLSCA), whereas the detail of domains is provided in Table 10 (appendix).

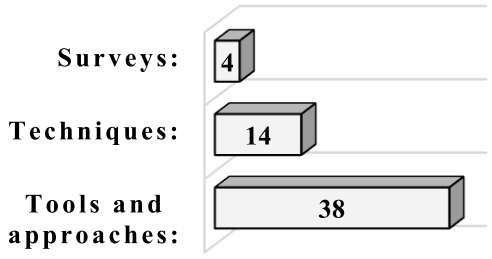

FIGURE 8. Segregation of selected research population.

Detail of Research Findings: In this section comprehensive review of research contribution is presented (marked with CN #). This research incorporates 56 multilingual source code analysis tools and approaches, including 38 tools, 14 techniques, and 4 survey/review papers. Research contributions of each research study in this SLR describe research aspect, analysis mechanism, research domain, representation, languages, evaluation and pros of the study. Figure 8 represents the division of selected research. This study is subdivided into tool support, technique and surveys & reviews for multilingual source code analysis.

c: TOOL SUPPORT

The tool support for Multilingual Source Code Analysis (MLSCA) is described in the form of the model, analysis mechanism, and experimental case studies. The complete detail about MLCSA tools is provided in Table 23 (Appendix). This section is comprised of different multilingual source code analysis tools, subdivided into dynamic analysis tools, static analysis tools, semantic analysis tools and hybrid analysis tools. It is observed through literature review, that multilingual source code analysis tools are more focused towards static analysis (shown in Fig: 9). Out of these 38 tools, 68% (26) of the studies focused towards static analysis of multilingual applications, whereas the contribution of dynamic analysis tools is 21% (8) and semantic & hybrid analysis tools are 11% (4).

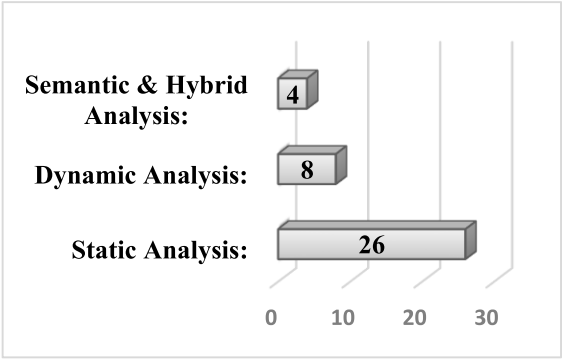

FIGURE 9. Type of tool support.

(i) DYNAMIC ANALYSIS TOOLS

In order to analyze MLAs, different dynamic analysis tools are presented that covers intermediate representations, Ajax based applications, web applications (WAs) and enterprise applications (EAs). The attributes for the dynamic analysis tools are presented in Tables 9 and 10 in the form of tool name, analysis mechanism, model representation, type of the tool, language support, experimental evaluation, focus of the problem, domain of study and research outcome.

The analysis mechanism includes holistic debugging [2], aspect weaving mechanism [28], trace link analysis [28], code clone detection [39], dependency analysis [44], [47], aspect-oriented programming [50], and evaluation of dynamic web contents [60]. These tools support analysis of general purpose languages (GPLs) including, Java, PHP, HTML, CORBA, J Script, EJBs etc. and domain-specific languages (DSLs) XML, Extend, Groovy, and UML etc. The tool support is also available for intermediate representation [2]. All of these tools are evaluated on the basis of different industrial environments [32], [39], [60], case studies [28], [44], simulated environments [2], prototype/standalone tools [47], [50] and plugins [28], [44].

Representation aspects of dynamic analysis tools is presented in the form of intermediate/ bytecode representation [2], [47], program transformation to JVM [28], GUI based FSM [32], high level language independent inter-component dependency & trace model [44], [47] and reverse engineering the interceptors to sequence diagrams (UML) [50].

The dynamic analysis tools show Robustness/ scalability [2], interoperability [28], effectiveness in model recovery [32], [47] and analysis of lightweight multilingual application [47]. In Table 8, the information of dynamic analysis tools is presented in the form of tool name, contribution #, analysis mechanism, model representation, language support and experimental evaluation.

Table 9 provides the information about how dynamic analysis tools addresses the issues raised in Question # 1. In this Table, problems and requirements are mapped to the contributions for better understanding about dynamic

TABLE 8. Dynamic analysis tools.

S #	Cn. #	Tool	Analysis Mechanism	Model	Type	Language Support	Evaluation
1	CN2	Normir [2]	Holistic Debugging,	Intermediate model, Dynamic modeling	Simulation model	Byte Code	Simics: Instruction set Simulator
2	CN18	AOP for Language Portability [27]	Aspect Weaving Mechanism.	Program transformation	Plugin	Java, DSL, C, JVM	Stratego (Eclipse Plugin), Case study
3	CN22	REAJAX [32]	Trace Link Analysis.	DOM GUI-based, State Models (FSM)	Industrial app	HTML, CSS, JScript, PHP, XML	http://pafm.source.net, http://tudu.source.net
4	CN29	Near-miss Clone Patterns in WAs [39]	Code Clone Detection	Patterns to exact near-miss code clones.	Industrial app	HTML, PHP, JavaScript, CSS, MVC	Clone detectors VisCad, NiCad. Industrial Was
5	CN33	GenDeMoG [44]	Explicit Dependency Analysis.	High-level inter-component dependency model	Plugin	Java & DSLs	Eclipse plug-in, Case Study: OFBiz.
6	CN36	Lässig [47]	Dependency Analysis and Modeling	Language independent traceability model	Prototype	Extend, Java, and Groovy	Autonomous trace links modeling framework.
7	CN39	I2SD [50]	Reverse Engineering. AOP.	Interceptors to sequence diagrams. Modular pipe filter architecture.	Standalone tool	EJB, JSP, HTML, CORBA, UML	Net Beans IDE, DataPortal, WasabiBeans
8	CN48	Dynamic Extraction & Analysis of WAs [60]	SW Evolution analysis.	UML-based explicit state model	Industrial app	Web Applications	ReWeb: Spider

TABLE 9. Dynamic analysis tools and support.

S #	Tool	Issue #	Cnt. #	Domain (short)	Support
1	Normir [2]	PR1	CN2	LIO, LIN, SWE, SWM, SWD	Robust/ scalable platform, Inspect distributed SW
2	AOP for Language Portability [27]	PR41	CN18	REU	Support aspects of interoperability in MLAs
3	REAJAX [32]	PR2, PR24	CN22	TRA	The results validate the effectiveness of recovering models.
4	Near-miss Clone Patterns in WAs [39]	PR2, PR3, PR29	CN29	CCD	Confirms patterns for cloning & tool Manage tangled code dynamic WAs
5	GenDeMoG [44]	PR17, PR19	CN33	DPA	Validate the presence of unknown dependencies across language artifacts in HAs.
6	Lässig [47]	PR2	CN36	DPA, SWM	Effective, lightweight, low-cost tool. automatic, comprehensive trace models.
7	I2SD [50]	PR8, PR10, PR19	CN39	REV, AOP.	UML representation of JEE interceptors. Usable with Net Beans or as a stand-alone tool.
8	Dynamic Extraction & Analysis of WAs [60]	PR29	CN48	SWE	Statistically, test & analyze navigational traits, identify inconsistent behaviors of WAs.

analysis tools. This Table contains problem/ requirement # (mentioned in Table 7), contribution #, Domain of study (mentioned in appendix Table 22) and outcome of the study.

(ii) STATIC ANALYSIS TOOLS

In this section intuitive knowledge about static analysis tools for analysis of multi-language applications is presented. These analysis tools help to analyze structure and intent of the software applications. These tools are used in Statistical/Empirical Analysis [12], Change Impact Analysis [22], [29], Content Analysis [24], Complexity Analysis [30], Object Relational Mapping [33], Software Visualization [42], [46], [57], Transactional Analysis [43], Analysis and modeling MLDE [45], [46], SQL Schema comparison [52], Refactoring [36], [51], [57], [65], Maintenance [42], [43], [64], Static low-level analysis [54], Lexical Analysis [55], Parsing source code metrics [56], Comprehension [62], Program Slicing [16], [64], Syntax Analysis [26], [53], multi commit software evolution analysis [63] and Pattern Analysis [65].

All of these tools are evaluated on the basis of different prototype tools, case studies, frameworks, multi-language development environments and plug-ins. These tools support both General purpose languages (GPLs) and Domain-specific languages (DSLs). General purpose languages include C/C++, C#, Java, J#, JSP, PHP, Java Beans, JavaScript, HTML, VB ,Net, SQL, Python, Smalltalk, and Ruby. The domain specific languages include SCRO, OWL, RDF, RDFS, XML, Groovy, Coral, UML, URN, J-unit, Prolog, Hibernate, Haskell, and Rascal. The analysis mechanism of these tools support statistical evaluation of meta-model [12], object sensitive analysis using program slicing and chopping [16], source code parsing based on graph-queries [17], [28], extended algorithm for analyzing structural change [22], MVC based navigational model for static analysis [23], [38], modular framework for content/ authority graph analysis of web documents [24], context and flow sensitive analysis tool for detection of cross-site & taint-style scripting vulnerabilities in web applications [26], multi-perspective rule based change impact analysis [29], complexity analysis by parsing .net languages [30], object-relational mapping (ORM) meta model for enterprise applications [33], [51], generalized approach for cross-language binding and refactoring [36], transactional analysis of enterprise applications [42], [43], analysis and integration of multi-language development environment [45], [46], schema comparison library for detection of changes in SQL schemes [52], SugarJ, a unique parsing mechanism for Java-based extensible language [53], language independent meta-model for low level static analysis [54] and refactoring [57], concurrent & robust parsing of

TABLE 10. Static analysis tools.

S#	Cnt. #	Tool	Analysis Mechanism	Model	Languages Support	Experimental Evaluation
1	CN7	Pangea [12]	Statistical & Empirical evaluation.	Language independent meta-model	Java, Smalltalk, C/C#	Famix Analyzer, VerveineJ
2	CN8	JOANA [16]	Program Slicing/ Chopping	Java Object-sensitive analysis	Java based languages	Eclipse Plug-in Dependency graph
3	CN9	GUPRO [17, 28].	Source Code parsing based on graph-queries.	Source Code Parsing	COBOL, CSP, Ada, C MVS/JCL, PSB, SQL.	GEOS, XFIG, COBOL etc
4	CN13	Diff/TS [22]	Structural change /Fine-grained Analysis.	Extend string differentiating algorithm	Python, Java, C/C++	Emacs editor
5	CN14, CN28	MoDISCO [23,38]	Static Analysis on MVC Frameworks	Navigational model.	HTML, JSP, XML, Java	Eclipse Plug-in Mia-Software (Sodifrance)
6	CN15	TARENTe [24]	Content Analysis. Authority Graph Analysis.	Adhoc modular framework.	Java, My-SQL	Open-source code WAs. Extract, explore web docs
7	CN17	Pixy [26]	Context /Flow Sensitive Analysis.	Detects cross-site, taint-style Vulnerabilities	PHP, XSS, HTML	Open source tool tested on PHP scripts
8	CN19	EMFTrace [29]	Change Impact Analysis. Dependency Analysis	Multi-perspective rule based CIA.	Java, J-Unit cases, UML, URN, OWL.	EMFTrace, Eclipse Framework
9	CN20	MMT [30]	Complexity analysis of MLAs.	Parsing Microsoft Intermediate Language	.Net based languages	NHibernate, MMT, Timecard CS Client
10	CN23	DATES [33]	Recover relational/ object oriented entities.	DATES (meta model) Third party API	Java, SQL	Tested on 3 studies KITTA, TRS, SALARY
11	CN26	XLL & Refactoring [36]	XLL Analysis, Binding & Refactoring	Generalized approach	Java, HTML, DSLs (HQL, HBM)	Jtrac support Spring, Hibernate, Wicket
12	CN31	FAMIX [42]	Statistical Analysis, SW Visualization, Maintenance	Expose and analyze transaction scope in EAs	EJB, JSP, HTML, Applets	FAMIX, Moose, Eclipse: Plugin.
13	CN32	MooseJEE [43]	Maintenance & Modeling, Transactional analysis	Code browser, Visualizations. Analyze architectural variants	Java Beans/ Script, JSP, HTML, Servlet	FAMIX platform
14	CN34	MLDE's Design [45]	Analysis & Multi-modeling of MLDEs.	Integration of MLDEs. Search based relation. Track XLLs	Java, JScript, HTML, XML. Groovy, Coral	TexMo, Coral
15	CN35	TexMo [46]	Static Analysis, Visualization	Analyze MLDEs by explicit R-Model.	Java/Script, HTML, XML	JTrac
16	CN40	Refactoring MLAs [51]	Multi-Language Structural Analysis & Refactoring	Object-relational mapping (ORM)	Java, Hibernate, SQL	Hibernate Application (HRM)
17	CN41	SQL Schema Comparison [52]	Database Analysis & Refactoring MLAs.	Schema compare library detect changes & validate SQL schemes	SQL DBMS	Eclipse plug-in
18	CN42	SugarJ [53]	Syntax Analysis, Random Context Free & Layout Sensitive.	Java-based extensible language. Unique parsing mechanism	DSLs, JavaScript, Prolog and Haskell	Spoofax-based IDE, five case studies
19	CN43	X-Develop [54]	Static low-level analysis,	Language Independent Meta-Model	C#, J#, VB	IText, .Net
20	CN44	Island Grammar [55]	Lexical Analysis, CFG.	Concurrent & robust parsing of MLAs	VB,HTML,Jscript, Rascal	McCabe-complexity
21	CN45	Analizo [56]	SW Evolution, Visualization, Dependency Analysis	Layered Style, Doxygen Parser. Generates source code metrics.	C, C++, Java	VLC project, Analizo
22	CN46	MOOSE [57]	Refactoring, Visualization, Type related Analysis	Language independent Meta model & Refactoring.	Java, Smalltalk	FAMIX model, Prototype tool
23	CN50	CrosLocator [62]	Program Comprehension	Bug localization algorithm to rank source code files in MLAs.	Ruby	Ruby-China
24	CN51	LISA [63]	Static Analysis, SW Evolution Analysis.	Language independent parsing. Translate AST into a graph structure.	Java based languages	JGit repositories
25	CN52	KDM [64]	Program Slicing/ Maintenance, Knowledge Engineering	Knowledge discovery Metamodel	C/C++, Java	Code Surfer plugin. Prototype tool evaluated on the large industrial code.
26	CN53	DeP [65]	Static analysis, Pattern Analysis, Refactoring,	Deprogramming. Pattern analysis & recognition.	Java	DeP tool

MLAs using lexical analysis and CFG [55], layered approach for the generation of source code metrics [56], bug localization and ranking source code files in MLAs [62], analysis of mixed code heterogeneous source code at an arbitrary number of revisions [63], knowledge discovery and reverse engineering heterogeneous artifacts [64] and deprogramming by patterns recognition [65].

In Table 11, the information about static analysis tools is provided with another aspect in which the problems and requirements are mapped with the contributions for better understanding. The contents of Table 11 include tool name, problem number (detailed domain based problems are mentioned in Table 7), contribution number, the domain of study (mentioned in Appendix Table # 22) and tool support.

(iii) SEMANTIC AND HYBRID ANALYSIS TOOLS

In addition to the above-mentioned tools, some hybrid tools are also proposed that support both static and dynamic analysis mechanism. These tools perform Semantic Analysis [3], [25], Static & Semantic Analysis [25], Static & Dynamic Analysis [20] and Static, Semantic and Dynamic Analysis [40].

The analysis mechanism in RECOS [3] provides explicit ontological model representations by using semantic knowledge base & point to analysis techniques. WARE [20] is used in program understanding and reverse engineer of web applications. This tool is used for extraction of source code in the form of graph repository. Authors in [25] provide Syntax and semantic analysis based parser for generation of abstract syntax tree (AST) and metrics of MLAs [25]. A meta-modeling tool is presented in [40] that is used to analyze heterogeneous applications at the high level of integration [40].

In Table 13 the issues raised in Q1 are mapped with the solutions (shown as CN #) in their respective domains. The semantic and hybrid analysis tools address four issues and requirements for analyzing MLAs. This Table highlights problem/ requirement # (mentioned in Table 7), contribution #, Domain of study (mentioned in appendix Table 22) and outcome of the study.

TABLE 11. Static analysis tools and support.

S#	Tool	PR #	Cnt. #	Type	Domains	Support
1	Pangea [12]	PR25	CN7	Prototype	STA	Analyze large MLAs. Reduces effort and cost of analysis. Comparative analysis of XLAs.
2	JOANA[16]	PR29,35	CN8	Plugin	CGP, PSC, LIN, LIO	Performs program IFC, integrity and confidentiality.
3	GUPRO [17, 28]	PR 39	CN9	Plugin	CGP	Use graph querying & graph algorithms extract source code into a graph repository.
4	Diff/TS [22]	PR5,8	CN13	Case Study	CIA, CCD, REU, STA, TMA	Structural analysis, identify patterns for managing clones in jumbled dynamic MLAs.
5	MoDISCO [23,38]	PR37	CN14, 28	Plugin	KEG, MM, PRU, REV, STA, SWC	Simple model based solution to reverse engineer legacy systems.
6	TARENTe [24]	PR29,38	CN15	Prototype	CAN, TRA/DPA	Provide multiple services i.e web crawling, mining, network analysis & visualization.
7	Pixy [26]	PR29,35	CN17	Prototype	DFA, STA, VLD	Validates detection of unknown vulnerabilities with the low false positive rate.
8	Rule-based CIA [29]	PR11	CN19	Framework, Plugin	CIA, DPA, REG, SWE, SMN, TRA	Determine impact propagation more reliably than distance-based dependency analysis.
9	MMT [30]	PR6,21	CN20	Prototype, Case Study	CGP, BCA, PTA	Helps to develop language independent parsers.
10	DATES [33]	PR8,36	CN23	Prototype, Case Study	MM, REV	Recovers design information & design quality of EAs.
11	XLL &Refactoring [36]	PR5,16	CN26	Prototype, Case study	XLL, RFT	Support better code understanding, reducing errors & maintain XLAs.
12	FAMIX [42]	PR8,10,43	CN31	Framework, Plugin	SVS, STA, SMN, TMA	Effectively analyze & expose structural/ behavioral conflicts of transactional JEAs
13	MooseJEE [43]	PR44	CN32	Framework	REV, SVS	Provides software visualizations for the recovery and analysis of transaction scope in JEAs.
14	MLDE's Design [45]	PR18	CN34	MLDE Case Study	LIN/LIO, MLDE, SWM	Tool support validated multi-modeling & tool building to analyze MLAs.
15	TexMo [46]	PR18	CN35	MLDE Case study	MLDE, SMN	Analysis, visualization & refactoring XLL of artifacts across language boundaries.
16	Refactoring MLAs [51]	PR32	CN40	Prototype F. work	RFT	Object-oriented & DB refactoring using Rename Method and Push Down Method for MLAs.
17	SQL Schema Comparison [52]	PR33	CN41	Plugin	MLDE, RFT, XLL	This library is used for database analysis and refactoring modern multi-language applications
18	SugarJ [53]	PR30	CN42	MLDE Case study	MM., MLDE, REU, STA	Supports syntax change in five language extensions of a source code file.
19	X-Develop [54]	PR8,40	CN43	Case study	BCA, MM, MLDE, PRU, RFT, XLL	The proposed Meta-model has the ability to accommodate new programming languages
20	Island Grammar [55]	PR24,26	CN44	Tool	LXA, SWC	Support adaptable parsing by extracting multiple, dissimilar & parser unfriendly features.
21	Analizo [56]	PR27,28	CN45	Prototype	CGP, SVS, SWE	Identify problems or source code enhancements.
22	MOOSE [57]	PR32,34	CN46	Prototype	DYA, LXA, RFT, SVS,	Evaluated the approach, reduce language dependency of tools.
23	CrosLocator [62]	PR7,8	CN50	Prototype	SWC	Support bug localization in MLAs
24	LISA [63]	PR 45	CN51	Tool	SWE, TMA	Analyze mixed code heterogeneous applications at an arbitrary number of revisions.
25	KDM [64]	PR10	CN52	Plugin Prototype	CMD, PSXC, KEG, SMN	Recover models from CBS, track information flow. Shows linear growth in execution time & size
26	DeP [65]	PR21,22	CN53	Prototype	DEP, DYA, BCA, PTA, RFT, STA, DPA	Identify/fix design problems, DP identification, copy-paste detection refactoring, fingerprint recognition & automated code documentation.

TABLE 12. Semantic and hybrid analysis tools.

S#	Cnt. #	Tool	Analysis Mechanism	Model	Languages Support	Type	Experimental Evaluation
1	CN3	RECOS [3]	Semantic Analysis	Ontological model & knowledge base.	SCRO, OWL, RDF/S	Plugin	Eclipse tool.
2	CN11	WARE [20]	Static & Dynamic Analysis	MDWE, Meta-Model	HTML JScript, XML	Tool	General Examples
3	CN16	MLAs SW Metrics [25]	Syntactic/Semantic Analysis. Extract AST	Open Source/ Extend Eclipse CDT Parser	C/C++, Java, JScript	MLDE Plugin	Eclipse IDE
4	CN30	Rascal [40]	Syntax Analysis. Dynamic Analysis, Semantic Links.	Analysis at high level of integration.	DSLs ASF, SDF, RScript, Java	Plugin	Meta Model. Eclipse IDE

d: TECHNIQUES FOR MULTILINGUAL SOURCE CODE ANALYSIS

In this subsection different techniques to analyze multilingual applications are presented in the form of the mathematical model, high-level model, graphical relational model, semantic model, UML model, framework, and algorithm. The analysis mechanism contains dynamic analysis, static analysis, semantic analysis, change propagation/impact analysis, data flow analysis, complexity analysis, fine-grained analysis, interoperability analysis, dependency graph, content

TABLE 13. Semantic and hybrid analysis tools and support.

S#	Cnt. #	Issues #	Domains	Model/Representation	Outcome
1	CN3	PR9	REU	Semantic knowledge base & point to analysis technique. Explicit ontological code representations.	Effectively evaluate multiple code libraries. Comprehensive program understanding & knowledge reuse
2	CN11	PR23, PR29	DYA, MLDE, STA	Use graph querying & graph algorithms, extracts source code into a graph repository.	Integrated querying and browsing Promote Program understanding, re/reverse engineering of MLAs.
3	CN16	PR26	TMA, AST, CGR	Adhoc modular framework. Extract, explore, analyze web docs	
4	CN30	PR31	DYA, MM, MLDE	DOM GUI-based, Build Finite State Models (FSM)	The results of case studies validate the effectiveness of recovering models.

TABLE 14. Techniques for multilingual source code analysis.

S#	Cat #	Description	Technique	Mechanism	Source Code Languages	Experimental Evaluation
1	CN1	Recovering Inter-System Interaction [1]	Mathematical Model. Extracts UML model of WAs.	Dynamic Analysis	Web Applications	FSM algorithm.
2	CN4	CIA of SPLs[4]	Conditional system dependence graphs for Multi-language SPLs.	Variability-Aware Program Analysis, Change Impact Analysis	NM	Case Study: Industrial automation(www.keba.cm), Prototype for CSDGs
3	CN5	Domain Based CPA [7]	A mathematical model, Mining version history.	Domain Based Change Propagation Analysis	NM	Case Study: BEMIS, evaluated on code coupling
4	CN10	Detection of XL Reuse [18]	Detection of cross-language source code reuse, character n- grams comparison.	Static Analysis	C++, Java and Python	Case Study
5	CN21	AST Based JAVA SW Evolution Analysis [31]	Improved tree matching algorithm, program change classification.	Change Impact Analysis. Evolution Analysis.	NM	Case study: medium sized projects.
6	CN24	XLA & Refactoring [34]	Generic MLSCA approach using semantic XLLs.	Dynamic Analysis, Semantic Link Analysis	Ruby on Rail, Android application.	Three case studies. Eclipse plug-in
7	CN25	ML Artifact Binding & Refactoring [35]	Detects related artifacts in each language for their bindings & refactoring.	Structural Analysis, Source code Refactoring, XLL Analysis	Java/ HTML, DSLs (HBM, HQL, Spring, Wicket).	Case study Spring, Hibernate, and Wicket,
8	CN27	Software clustering [37]	Extend information bottleneck theory. Semantic features	Architectural & Component Discovery; Syntax Analysis	NM	Case study (Code Clustering)
9	CN38	XL Parsing, Dependency finding [49]	Generic algorithm. Cross-language dependency detection	Statistical Filtering Algorithm	Java, JServer Faces, Spring, XML.	Spring web flow framework (SWF)
10	CN46	Patterns of XLL in Java Frameworks [58]	Extract Cross-Language Patterns in Java & generalize them with 3 case studies.	Static Analysis, Dynamic Analysis, Refactoring.	Java Beans, Spring, HQL, XML, Apache Meetings	case Study JTrac
11	CN47	A Model-Based Approach to Language Integration [59]	Seamless language integration using MPS. Cross-language constraints & error checking.	Dynamic Analysis, XLL Analysis, Refactoring, Comprehension.	Java	MPS Eclipse Platform
12	CN54	Cross Language Source Code Reuse Detection [66]	Cross language similarity & reusability detection using latent semantic analysis (LSA) approach.	Static analysis, Source code reusability.	Python, C, Java	Rosetta code repository, Java Converter, java2python
13	CN55	SLI with Natural Language Classifiers [67]	Source code classification techniques and unstructured code optimization.	SW code identification, interaction & optimization,	C/C++, Jscript, HTML, PHP, CSS	GitHub repository
14	CN56	Identifying source code programming using NLP [68]	multiple language identification and categorization	NLP based techniques like Naive Bayes, n-grams, skip-grams etc.	C, C++, PHP, HTML	GitHub repository

& authority graph, abstract syntax tree, navigational models, comparative analysis and future guidelines. The techniques are presented in the form of algorithms [1], [31], [49], mathematical model [7], case studies [4], [7], [18], [31], [34], [35], [37], [58], frameworks/platforms [49], [58], [59]. Table 14 provides brief of each contribution in the form of analysis technique, analysis mechanism, language support and experimental evaluation.

In Table 15 each approach is described with their problems with their solution, enhanced with research domains and research support. In this Table, the problems (PR #) are mapped to the solutions (CN #).

e: SURVEYS AND REVIEWS FOR MLSCA

In this subsection, the importance of source code analysis in MLAs is presented in terms of history, basics, development environments, and applications. The study is presented in the form of surveys and empirical evaluations. Table 16 presents the summarized information.

3) ASSESSMENT OF Q3: WHAT ARE THE SHORTCOMINGS AND PROSPECTS OF CURRENT RESEARCH IN MULTILINGUAL SOURCE CODE ANALYSIS (MLSCA)?

Each research has its limitations or future guidelines for conducting further investigation and improvement.

TABLE 15. Techniques for multilingual source code analysis and support.

S#	Description	PR #	Cnt #	Domains	Support
1	Recovering Inter-System Interaction [1]	PR 1	CN1	DYA, LIN, REV	Dynamic Analysis of HAs. No source code modification required
2	CIA of SPLs[4]	PR 5	CN4	CIA, DPA	Evaluate the impact of changes in XLAs. Identify cross-language patterns in SPLs.
3	Domain Based CPA [7]	PR13	CN5	CIA, SWC, SWE, SMN	Change impact estimation, maintain code history, helps to prevent errors.
4	Detection of XL Reuse [18]	PR 5	CN10	CCD, REU	Inspect comments, variables & reserve words in XLAs. Plagiarism detection of XLAs.
5	AST Based JAVA SW Evolution Analysis [31]	PR 8, 36,42	CN21	SWE, TMA	Validate performance, average change error less than 5.38%.
6	XLA & Refactoring [34]	PR14	CN24	PRU, RFT	Detects related artifacts & their bindings. Refactor bound artifacts in multiple languages.
7	ML Artifact Binding & Refactoring [35]	PR15	CN25	RTF, XLL	Results validate discovery, binding & refactoring of cross language artifacts.
8	Software clustering [37]	PR	CN27	CCD, CMD, REV, SWC	Helps in improving information quality and reducing noise
9	XL Parsing, Dependency finding [49]	PR14	CN38	SMN, REG, DEP	Validated SWF accurate, extendable algorithm for dependency detection.
10	Patterns of XLL in Java Frameworks [58]	PR20, 21	CN46	PORU, RFT, SWC	Common patterns of XLL of Java frameworks with DSLs are identified
11	A Model-Based Approach to Language Integration [59]	PR4	CN47	LIN, RFT	Independent creation of specific editors for any peculiar language.
12	Cross-Language Source Code Reuse Detection [66]	PR3	CN54	REU, NLP, STA	LSA approach efficiently differentiates re-used and associated codes without using compilers.
13	SLI with Natural Language Classifiers [67]	PR46	CN55	NLP, REg, STA, SWE, RE	Support multiple language identifications and embedded software.
14	Identifying source code programming using NLP [68]	PR22	CN56	NLP, STA, RE	Identification of source code language using NLP classfixation techniques

TABLE 16. Surveys and reviews for MLSCA.

S#	Description	Representation	Case Study	Purpose
1	XL Analysis & Refactoring [11]	Source code analysis trends, Survey paper	Studies NM	Provide baseline studies for source code analysis.
2	Importance of Source Code Analysis [21]	Provide history and foundation of source code analysis.	NM	Presented importance & future trends.
3	Empirical Analysis of MLAs [48]	Evolution and utilization of MLAs. (XML, Shell/Make, HTML/CSS)	GitHub case study	Evaluated 1150 open source projects language ecosystem in 3 DSLs in top 10 GPLs.
4	Towards MLDEs [61]	United source code models & metaprogramming as MLIDE.	NM	High level fabricated model. Diverse IDE for high-quality SW.

In this subsection, limitations and future work of different approaches are presented. The recent research comprehensively describes the trends and approaches for source code analysis of multilingual applications. There are a number of recent and future directions presented to analyze multilingual applications but their scope is limited [11].

In this paper, 56 issues are recognized individually from selected research contributions which are marked with LF #. These limitations or future work are separated into 13 software engineering domains. The comprehensive detail is mentioned in Table 17.

Detail of issues and requirements for multilingual source code analysis (MLSCA): In this subsection, domain-wise detail of shortcoming and research prospects are presented for further study. The percentage of research findings. These results include, Static Analysis 25%, Program Understanding & RE 22%, XLL Detection 16%, Dynamic Analysis 15%, Source Code Refactoring 11%, Change Impact Analysis 9%, Multiple Language IDEs 9%, Analysis of EAs 7%, Quality Assurance 5%, Code Reusability 4%, SW maintenance 4%, Cross-language Security 4% and Semantic Analysis 4%.

The following domains comprehensively describe the problem and requirements from selected research.

a: DYNAMIC ANALYSIS

In [1], recovery of Interaction across heterogeneous applications (HAs) is discussed. This approach is exclusively used for extraction and recovery of intersystem interaction behavior to analyze HAs. A graphical visualization and analysis of the concurrent behavior of generated views are required for effective analysis.

In [32], dynamic analysis and reverse engineering of Ajax applications (ReAjax) are performed by building FSM through execution traces. The proposed tool can be used for test case generation and reliability is required to be tested on a case study. Traceability across multilingual artifices is determined by a generic traceability modeling tool, Lässig [47]. This tool performs model-to-text transformations and mark affected artifacts in the form of trace links. However, in the complex transformation of different type objects this tool returns only a single trace link [47].

In [59], an effective approach for integration of multiple languages is presented using Meta Programming System (MPS) on Eclipse Platform. This approach needs empirical assessment and further improvements for persistence & categorization in cross-language constraints. In [60], a dynamic analysis of web applications (Was) is presented

TABLE 17. Limitations and future work for multilanguage source code applications.

Limitations and Future Work (LF)		Domains												
		Dynamic Analysis	Quality Assurance	Source code Reusability	Change Impact Analysis	XLL Detection	Software maintenance	Static analysis	Source Code Refactoring	Cross language Security	Program Understanding & RE	Multiple Language Development Environments	Analysis of Enterprise Applications	Semantic Analysis
Analysis of concurrent behavior of HAs.	LF1	1												
Holistic debugging technique to be tested in simulated environment	LF2		2											
Semantic analysis approach require improvement recovery of multiple source code	LF3													3
Accuracy for extraction of XLLs & CIA require improvement in CSDG	LF4				4	4								
CIA methods in EAs support fewer UICs & require empiricle evaluation	LF5				7								7	
General guidelines to analyze MLAs is limited	LF6	11						11						
limited OO repository for large MLAs	LF7							12						
Prevention of XL information leaks is required other then Java based languages	LF8									16				
Integerated tool support is required fo MLAs	LF9										17			
Cross language alignment based similarity analysis is future prospect	LF10					18								
Automatic & OO support required for WAs.	LF11						20				20			
The study provides general understanding of MLSCA	LF12	21						21						
Improvement in performance required in XLA & clone detection.	LF13							22						
Existing MDRE techniques in MoDISCO are suitable for small & medium projects	LF14										23			
Tool support for analysis & visualization of aggregates is available for WAs only.	LF15										24			
Enahance capability of parser to accommodate new languages	LF16							25						
Automated cross site vulnerability detection is required in large MLAs	LF17								26	26				
Portability of SPLs across ML platforms is required.	LF18											27		
Systematic dependency detection mechanism is required	LF19				29									
MSIL tool is supports only.NET based languages	LF20		30											
Change classification & evolution analysis in MLAs is limited.	LF21				31						31			
Relaibility testing of Ajax applications.	LF22	32									32			
Recovery of design information in Eas only support java based EAs	LF23										33		33	
Generic approach for refactiring XLLs is not complete	LF24					34			34					
Cross language artifact binding & refactoring in large MLAs is challanging	LF25					35								
Enhancement in clustering at multiple hierarchical levels is required	LF26													37
Support for the intergration of legacy system is required	LF27					39								
Evalaution & Language support needs improvement for RASCAL DSL	LF28							40						
Identification of desirable properties & architectural constraints of JEAs is hard.	LF29												42	
Generic and extensible representation to analyze JEAs.	LF30												43	
Algorithm for mining cross language dependencies is not generalized	LF31					44								
MLDE provides limited representation of XLRs	LF32											45		
Automated detection of XLRs is required MLDEs.	LF33											46		
Tracibility analysis & modeling of complex applications	LF34	47												
Assessment of DSLs and qualitative analysis is requirent.	LF35							48						
Accuracy & classfifcation for cross langauge dependencies is weak	LF36					49								
Extension & validation of I2SD tool is required.	LF37	50									50			
Generic automated refactoring in MLAs is hard	LF38								51					
Adapting DB schema in SQL schema comparer library is future concern	LF39								52					
Extending syntax of PLs & desugring sometime create confilicts.	LF40				53			53						
Generaluzed language independent approach is required for detection MLAs.	LF41								54					
Multilingual parsing using Island grammars is not generalized for MLAs.	LF42							55						
Doxygen parser in Analizo needs to parse WAs	LF43							56						
FAMIX require extension torefactor languages other then OOPLs	LF44								57					
Support for the intergration of GPLs & DSLs is required	LF45					58								
Persistence & categorization in cross-language constraints.	LF46	59												
Incomplete UML model of Dynamic Was	LF47	60												
Scalability analysis across multi-language components is required in MLIDEs.	LF48											61		
Query reformulations required for improving bug localization algorith	LF49		62											
XLLs need to be resolved for legacy applications.	LF50					36								
Static analysis of large MLAs requires indpendent parserfor each language.	LF51							63						
Knowledge discovery meta-model tobe extended for more languages	LF52			64							64			
Dep requiredmaintanace & optimization of code repositories	LF53						65				65			
LSA approach needs to be explored on other XL code resuablity scenarios.	LF54			66				66						
NLP based SLI technique requires universal IDE support	LF55							67			67	67		
Source code Identification using NLP approach needs precise classification.	LF56							68			68			

that recover UML model by analyzing the multi-language code during program execution. This approach has certain limitations. The comprehensive behavior of the applications resulted in incomplete UML model also needs empirical evaluation. Moreover, this technique does not support Java platform.

b: QUALITYASSURANCE OF MLAs

In [2], a holistic debugging tool for quality assurance of distributed MLAs is presented. The proposed approach is limited to examine few properties of source code. Testing software applications in a simulated environment are the future work of this tool.

The technique presented in the form of MSIL tool [30], support complexity analysis of MLAs & development of language independent parsers by using intermediate representation. However, this tool only supports languages of.Net development environment.

In [62], a cross-language bug localization algorithm & language translation tool (CrosLocator) is presented. The performance of tool needs further testing & improvement in bug localization and query reformulations methods to re-originate the terms of bug reports is the future prospects of the research.

c: SOURCE CODE REUSABILITY

In [64], analysis, reverse engineering, and reusability of heterogeneous multi-language artifacts are presented by building knowledge discovery meta-model [64]. In future, more source languages and component composition/configuration languages are needed to be incorporated in the analysis model.

A simple approach for cross-language source code investigation and reusability is presented in [18]. The proposed model utilize sliding windows for source code comparison. Cross-language similarity analysis is the future work of this approach.

d: CROSS-LANGUAGE CHANGE IMPACT ANALYSIS (CIA)

Domain-based change propagation and impact analysis approach [7] is helpful in evaluating the impact of change, maintaining code history and preventing errors in enterprise applications (EAs). This approach uses variability-aware CIA methods in multi-language software product lines (SPLs). However, this approach is insufficient for systems which throw functionality with few user interface components. The empirical assessment is also required to evaluate these systems.

Cross-language links (XLLs) help to understand change propagation & impact analysis across the multilingual artifacts. XLLs are built dynamically, therefore a combination of static and dynamic analysis techniques are required. The existing CIA techniques only support single languages and lack context awareness to improve the accuracy of conditional system dependency graph (CSDG) [4]. In order to analyze the heterogeneous artifacts of different languages a systematic and refined dependency detection mechanism is required [29].

e: CROSS-LANGUAGE LINK DETECTION

Dependency analysis / Cross-language links (XLL) can be helpful in better program understanding, analysis, maintenance, error handling and refactoring MLAs. There are a number of approaches presented to handle cross-language dependencies across MLAs. The detection and managing of dependencies in large MLAs are quite hard and challenging due to their complex and heterogeneous nature.

GenDeMoG [44] is a tool that determines cross-language dependency across various multi-language artifacts. This tool identifies explicit inter-component dependency across a multi-language enterprise system. But this tool is not generalized for all enterprise applications. The algorithm of GenDeMoG is not complete and only mines cross-component dependencies [44].

In [49], a simple, generic and extendable algorithm is proposed for cross-language dependency detection. The system needs classification according to language and its size. The behavior of the system needs to be examined at the component level. The accuracy of the system needs improvement.

Cross-language links are analyzed in Java and Domain Specific Languages (DSLs) [36], [58]. In [36], a multilingual platform is proposed for the discovery, management, and refactoring XLLs in Java & DSLs. However, the tool support for XLLs in DSLs and integration with legacy applications is required. Moreover, identification of cross-language links is also required in between two general-purpose languages or in two declarative DSLs [58]. Cross-language artifact binding and refactoring in another challenge in MLAs [35].

A generic approach is presented in [34] to analyze and refactor cross-language code by explicitly specifying and exploiting semantic links. However, the tool support is limited for refactoring languages and needs to support more languages and sample link specifications in future.

f: SOFTWARE MAINTENANCE

Analysis and reverse engineering of heterogeneous components of web applications are carried by WARE tool [20]. This tool is helpful for understanding, maintenance and evolution of heterogeneous web applications. However, the quality assessment of the tool needs empirical evaluation. The automatic, dynamic & object oriented support is also missing in this tool [20].

Deprogramming is a reverse process that converts source code into concept, designs, and patterns. The proposed tool, Dep (deprogramming) abstracts source code into dependency graph and then mines to design patterns. The Dep tool is needed to be commonly employed in managing source code repositories, SW optimizations, and recommendation of refactoring targets [65].

g: STATIC ANALYSIS

Static analysis of large multilingual software is carried out by Pangea [12]. The presented tool provides a data repository for more systems & allows empirical investigation and helps in comparative statistical analyses across programming languages. Pangea has limited data repository and requires a diverse data repository to accommodate more Object Oriented languages [12].

Cross-language analysis & clone detection is supported by Diff/TS tool. This tool performs fine-grained analysis of structural changes in the form of control flow graphs during virtual execution. However, the whole process is time consuming & inflexible. Future work includes accumulation of more compound and performance enhancement operations on large trees [22].

In the case of multi-language source code applications, the key challenge is to create a parser for each source code language [25]. This paper recommends parsers containing a modern multi-language feature that support complete information about source code and provide abstract syntax tree of multi-language applications. The proposed parser is limited to C/C++ languages. Future work is to use a systematic approach to analyzing the maturity of parsers using Open Maturity Model or to device some contemporary measurement techniques [25].

RASCAL is a Domain-specific language (DSL) that is used for analysis & transformation of complex heterogeneous applications (HAs) [40]. The problem with RASCAL is that it generally supports Object Oriented Languages. Moreover, its performance has not been completely evaluated in terms of its design and implementation.

Sugar libraries present a unique concept to extend the syntax of a programming language within the language. In [53], presented a SugarJ library, embedded with DSLs which is useful in artifacts reusability and handling software complexity. However, the composition of grammar and applying desugaring rules may cause conflicts in some case studies.

A lightweight robust multilingual parsing approach is presented in [55] using Island grammars for concurrent parsing of MLAs in the form of the parse tree. This approach is generalized only for JSP and other dynamic web content paradigms.

The analysis, visualization, and generation of software metrics of MLAs can only be cost effective by using automatic analysis tools that support the generation of source code metrics, dependency graphs, and software evolution analysis [56]. In [56], Analizo toolkit is presented that supports analysis and visualization of MLAs in the form of source code metrics, dependency graphs, and evolution analysis. This tool is based on Doxygen parser which cannot parse the source code completely. Moreover, this tool does not support web applications and a web-based version is the future work of Analizo toolkit.

An open source static analysis tool LISA is presented [63] to analyze language independent online source code repositories. This tool efficiently computes code metrics with multiple revisions. However, this tool requires an independent parser for each language. It lacks understanding of the global structure of software applications. Improvement in parsing speed is another issue.

h: SOURCE CODE REFACTORING

In [51], automated multi-language refactoring (MLR) is presented using object-relational mapping (Hibernate ORM) with the database. It is difficult to build a generic approach for automatic refactoring in MLAs. Preserving wide range of useful semantic modifications of single artifact types is quite hard.

In [54], a language independent meta-modeling tool X-DEVELOP is presented that support analysis and refactoring of multiple languages. However, this tool has weak recognition of multi-language components. Moreover, X-DEVELOP lacks support in generalization, dynamic language contents, upcoming languages and low-level languages [54].

In [57], a language-independent meta-model FAMIX (on a refactoring engine) is presented to refactor source code of object oriented programming languages (OOPLs). However, this tool requires evaluation on more case studies. The future work is required to extend the tool with more languages like C++, Ada & COBOL.

In [52], SQL schema compare library is presented to adapt SQL modifications for database analysis and refactoring of modern multi-language applications. To support database access with JDBC and Java Persistence API (JPA), the SQL schema compare library is needed to be integrated with Eclipse plug-in. The future aim is to highlight those SQL statements and JPA entities which mismatch with SQL schema.

i: CROSS-LANGUAGE SECURITY

Detection of cross-site scripting vulnerabilities in scripting languages is determined by an open source static source code analysis tool ''Pixy''. The process of vulnerability detection is manual, therefore an automated mechanism is required in developing web applications especially for large and complex applications [26].

Language-based security is important for prevention in cross-language information leaks. JAONA [16] uses security algorithms to check information leaks. However, this model only support Java based languages.

j: PROGRAM UNDERSTANDING, RE-ENGINEERING, AND REVERSE ENGINEERING IN MLAs

Understanding legacy applications require more effort for refactoring, re-engineering & reverse engineering. Generic & extensible solutions are required for evolution/modernization of existing systems.

GUPRO supports multiple program analysis techniques using graph technology that helps in program understanding, re-engineering and reverse engineering of multilingual software applications [17]. In future, this tool is needed to be integrated with other reengineering tools via GXL-interfaces.

MoDISCO [23] is an open source model that reverse engineer legacy artifacts into relevant model based reviews. The proposed solution is suitable for small & medium projects and requires empirical evaluation on industrial projects.

The topology of the web applications is provided by TARENT that helps in extraction, analysis, and visualization of aggregates for large hypertext web applications [24]. The available tool focuses on the analysis of web documents only. There is a need to formalize a general theory of aggregates (mathematically), including different kinds of topological models.

Extracting change information from source code helps in better understanding MLAs. In [31], an improved tree

TABLE 18. Quality assessment criteria.

S #	P #	Score				Total Score	QAC (%)	S #	P #	Score				Total Score	QAC (%)
		Q1	Q2	Q3	Q4					Q1	Q2	Q3	Q4		
1	1	1	0.5	0.5	0.5	2.5	62.5	29	40	1	0.5	0.5	0.5	2.5	62.5
2	2	0.5	0.5	0.5	0.5	2	50	30	42	1	0.5	1	0.5	3	75
3	3	0	0	0.5	0.5	1	25	31	43	1	1	0.5	0.5	3	75
4	4	1	0.5	1	0.5	3	75	32	44	1	0.5	0.5	1	3	75
5	7	0.5	0.5	0.5	0.5	2	50	33	45	1	0.5	1	1	3.5	87.5
6	11	0.5	1	0.5	1	3	75	34	46	1	0.5	1	1	3.5	87.5
7	12	1	0.5	1	1	3.5	87.5	35	47	1	0.5	1	1	3.5	87.5
8	16	0.5	0.5	0.5	1	2.5	62.5	36	48	0.5	0.5	0.5	0.5	2	50
9	17	1	1	1	0.5	3.5	87.5	37	49	1	0.5	1	0.5	3	75
10	18	1	0.5	0.5	1	3	75	38	50	0.5	0.5	0.5	0.5	2	50
11	20	1	1	1	1	4	100	39	51	1	0.5	0.5	0.5	2.5	62.5
12	21	0.5	0.5	0.5	0	1.5	37.5	40	52	1	0.5	0.5	0.5	2.5	62.5
13	22	0.5	0.5	0.5	0.5	2	50	41	53	1	1	1	0.5	3.5	87.5
14	23	1	1	0.5	1	3.5	87.5	42	54	1	0.5	1	1	3.5	87.5
15	24	0.5	0	0	0	0.5	12.5	43	55	1	1	1	1	4	100
16	25	1	0.5	1	1	3.5	87.5	44	56	1	0.5	0	0.5	2	50
17	26	1	0.5	0.5	1	3	75	45	57	0.5	0.5	0.5	0.5	2	50
18	27	0.5	0.5	0.5	0.5	2	50	46	58	1	0.5	1	1	3.5	87.5
19	29	1	0.5	1	0.5	3	75	47	59	1	0.5	0.5	1	3	75
20	30	1	1	0.5	0.5	3	75	48	60	0.5	0.5	0	0.5	1.5	37.5
21	31	0.5	0.5	0.5	0.5	2	50	49	61	0.5	1	0	0	1.5	37.5
22	32	1	1	0.5	1	2.5	87.5	50	62	0.5	0.5	0.5	0.5	2	50
23	33	1	0.5	1	0.5	3	75	51	63	0.5	0	0	0	0.5	12.5
24	34	1	0.5	0.5	1	3	75	52	64	1	0.5	0.5	0.5	2.5	62.5
25	35	1	0.5	1	1	3.5	87.5	53	65	0.5	0.5	0.5	0.5	2	50
26	36	1	0.5	1	1	3.5	87.5	54	66	1	0.5	0.5	1	3	75
27	37	0.5	0	0	0	0.5	12.5	55	67	1	1	1	0.5	3.5	87.5
28	38	0.5	0.5	0.5	0.5	2	50	56	68	1	0.5	0.5	0.5	2.5	62.5
				Total Score						42.5	29.5	32.5	34	138.5	
				Average Score						0.80	0.56	0.61	0.64	2.61	
				Percentage						80.2	55.7	61.3	64.2	5.4	

matching algorithm is presented. However, this algorithm has limited support for small and average projects only.

A reverse engineering tool I2SD is discussed in [50]. This tool visually represents sequence diagrams of the interceptor model. However, the I2SD tool is needed to be extended with recent extensions & validation is required through case studies.

Jiang *et al.* [33] presented a meta-modeling tool, DATES for reverse engineering of modern enterprise applications (EAs). The proposed tool recovers design information and design quality of object-oriented enterprise applications. However, DATES is limited for java based applications and requires more evaluation on other enterprise applications.

k: MULTIPLE LANGUAGE DEVELOPMENT ENVIRONMENTS (MLDE)

Automated cross-language references and relationship are key issues in the development of multi-language software systems for Integrated Development Environments (IDEs). The existing development environment does not completely support relations across multi-language artifacts.

The tool supporting MLDE provides a limited representation of cross-language relationships. Moreover, language representation needs to be extended for visual languages like UML languages [45].

In [46], a prototype tool, TexMo is presented for static analysis, refactoring and visualization of cross-language relations. However, this tool is manual and requires automatic detection of cross-language relations. The establishment of fixed and string transformation relations is the future work of this tool.

In [61], a high-level research design and model are presented that support Multilingual Integrated Development Environments (MLIDE). Analyzing scalability across multi-language components is required for the proposed model.

Portability across multiple language platforms is handled by an aspect weaving mechanism [27]. This paper proposes aspect-oriented programming to address portability with regard to languages that target multiple platforms. The future work in [27] is highlighted to address portability concerns of software product lines by adding aspects and addressing dependencies to multiple platforms.

l: ANALYSIS OF ENTERPRISE APPLICATIONS

In large applications like Java Enterprise Applications (JEA's), the information is distributed across various

TABLE 19. Distribution of research papers.

Sr #	Origin	Publishers	Papers	Journals	Conferences	Workshops
1	Australia	IEEE	1	1	0	0
2	Austria	IEEE/ ACM/Elsevier	1	0	1	0
3	Brazil	IEEE, ACM	4	1	3	0
4	Canada	IEEE, ACM/Elsevier	3	0	3	0
5	China	IEEE/IEEE Transactions, ACM,	3	1	2	0
6	Denmark	IEEE, ACM	3	0	3	0
7	France	IEEE, Elsevier	2	1	1	0
8	Germany	IEEE, ACM, Springer, Elsevier	12	2	9	1
9	Italy	IEEE, ACM, Springer, IET	4	1	1	2
10	Japan	IEEE/ACM	1	0	1	0
11	Netherlands	IET /IEEE SCAM./ACM/ Elsevier	5	1	4	0
12	Norway	IEEE/ACM/Elsevier	1	0	1	0
13	Romania	IEEE	1	0	1	
14	Singapore	ACM/IEEE	1	0	1	0
15	Spain	Springer/ACM	1	0	1	0
16	Sweden	IEEE/ACM	1	0	1	0
17	Switzerland	IEEE /ACM	5	0	4	1
18	UK	IEEE/ACM	1	0	1	0
19	USA	IEEE/ACM	6	0	5	1
		Total	56	8	43	5
		Ratio %		14	77	9

components and their relationships. It is hard to analyze the heterogeneous nature of JEAs and verify desirable properties & architectural constraints [42]. In [43], a tool MooseJEE is presented as an extension of Moose to analyze the transactional scope of JEAs. This tool visualizes structural and behavioral anomalies of transaction scope. MooseJEE is limited in terms of quality & requires a generic and extensible representation to analyze JEAs. Moreover, the tool is focused only for Java EnterpriseAapplications.

m: SEMANTIC ANALYSIS

Semantic analysis approach is presented in [3] and [37]. The research in [3] investigated the semantic annotations and domain ontologies for searching and recovery of source code snippets on multiple source code libraries. This approach needs improvement in search precision and ranking. It also requires a repository of relevant code samples.

The proposed approach in [37] combines multiple features types and applies automated weighing on features to improve information quality and to reduce noise. However, this approach requires enhancement for clustering at multiple hierarchical levels.

D. POTENTIAL AREAS FOR FUTURE RESEARCH

Before completing our discussion and proceeding towards conclusion section, it is pertinent to discuss some of the key areas that require improvement and has the potential for further research. We identified key areas after in-depth analysis and peer review of all the finalized studies. The outcome of multilingual source code analysis requires, visualization of cross-language source code artifacts [1], [45], [56], extraction and generalization of cross-language components and their relationships [44], [46], [54], [64], performance and accuracy of analysis mechanism [44], [49], scalability of cross-language analysis techniques [44], [61], code coverage of multilingual components [7], similarity analysis and clone detection [18], [22], advanced parsing techniques to analyze range of multilingual applications [25], [55], [56], [63], context awareness of multilingual applications [4], generation of complete source code documentation [60], completeness of approach for change impact and dependency analysis by examining the cross-language artifacts and their links [4], [7], [29], optimization and maintenance of multilingual source code repositories [65], generalized approach for vulnerability detection and prevention of information leaks [16], [26], program understanding, reverse engineering and refactoring support for medium and large scale multilingual applications [23], [31], [33], [34], [50], [51], change classification and evolution analysis [56] and portability across multiple language platforms by addressing dependencies [27].

IV. CONCLUSION

The basic purpose of conducting this research is to provide state of the art research available for Multilingual Source Code Analysis (MLSCA). This research provides

TABLE 20. Finalized list of primary studies.

Paper ID	Ref #	The title of Selected Research Publications and Proceedings.	Topics /domain of the research study	Publication Venue
1	1	Ackermann, C., Lindvall, M., & Cleaveland, R. (2009, October). Recovering views of inter-system interaction behaviors. In Reverse Engineering, 2009. WCRE'09. 16th Working Conference on (pp. 53-61). IEEE.	Dynamic Analysis of HAs	Conference
2	2	Albertson, L. (2006, September). Holistic Debugging--Enabling Instruction Set Simulation for Software Quality Assurance. In Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on (pp. 96-103). IEEE.	Molding, Analysis & Simulation of Distributed MLA's	Conference
3	3	Alexandro, C. V. and H. C. Gall (2015). Rapid multi-purpose, multi-commit code analysis. Proceedings of the 37th International Conference on Software Engineering-Volume 2, IEEE Press.	Pattern Analysis, Machine Intelligence	Journal
4	4	Alnusair, A., Zhao, T., & Bodden, E. (2010, August). Effective API navigation and reuse. In Information Reuse and Integration (IRI), 2010 IEEE International Conference on (pp. 7-12). IEEE.	Information Reuse and Integration (IRI)	Conference
5	5	Angerer, F. (2014, September). Variability-aware change impact analysis of multi-language product lines. In Proceedings of the 29th ACM/IEEE international conference on automated software engineering (pp. 903-906). ACM.	Change Impact Analysis (CIA) Techniques, Source Code Analysis	Conference
6	8	Aryani, A., Peake, I. D., & Hamilton, M. (2010, September). Domain-based change propagation analysis: An enterprise system case study. In ICSM (pp. 1-9).	SCAM, Change Propagation Analysis, SW Comprehension.	Conference
7	12	Binkley, D. (2007, May). Source code analysis: A road map. In 2007 Future of Software Engineering (pp. 104-119). IEEE Computer Society.	SCAM Trends	Conference
8	13	Caracciolo, A., Chis, A., Spasojevic, B., & Lungu, M. (2014, September). Pangea: A Workbench for Statically Analyzing Multi-Language Software Corpora. In Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on (pp. 71-76). IEEE.	Static Analysis of MLA's	Conference
9	17	D. Giffhorn, & Hammer, C. (2008, September). Precise analysis of java programs using JAONA. In Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on (pp. 267-268). IEEE.	Dependency graph, slicing, and chopping	Conference
10	18	Ebert, J., Kullbach, B., Riediger, V., & Winter, A. (2002). Gupro-generic understanding of programs an overview. Theoretical Computer Science, 72(2), 47-56.	Re engineering, reverse engineering, program understanding	Journal
11	19	Flores, E., Barrón-Cedeño, A., Rosso, P., & Moreno, L. (2011). Towards the detection of cross-language source code reuse. In Natural Language Processing and Information Systems (pp. 250-253). Springer Berlin Heidelberg.	Source Code Reuse, Cross-language Source Code Analysis, Plagiarism Detection	Conference
12	20	Flores, E., et al. (2015). "Cross-Language Source Code Re-Use Detection Using Latent Semantic Analysis." Journal of Universal Computer Science 21(13): 1708-1725.	Cross language source code reuse detection,	Journal
13	22	G. A. Di Lucca, Fasolino, A. R., Pace, F., Tramontana, P., & De Carlini, U. (2002). WARE: a tool for the Reverse Engineering of Web Applications. In Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on (pp. 241-250). IEEE.	Program Comprehension, Maintenance, Reverse Engineering.	Conference
14	23	Harman, M. (2010, September). Why Source Code Analysis and Manipulation Will Always be Important. In SCAM (pp. 7-19).	Program Understanding, Maintenance & Evolution	Journal.
15	24	Hashimoto, M., & Mori, A. (2008, October). Diff/TS: A tool for fine-grained structural change analysis. In Reverse Engineering, 2008. WCRE'08. 15th Working Conference on (pp. 279-288). IEEE.	Reverse Engineering, Code Clone Detection, Web Applications	Conference
16	25	Hugo, B., Cabot, J., Jouault, F., & Madiot, F. (2010, September). MoDisco: a generic and extensible framework for model driven reverse engineering. In Proceedings of the IEEE/ACM international conference on automated software engineering (pp. 173-174). ACM.	Reverse Engineering, Program Understanding, Comprehension.	Journal
17	26	Janardan, M., Annervaz, K. M., Kaulgud, V., Sengupta, S., & Titus, G. (2012, October). Software clustering: Unifying syntactic and semantic features. In Reverse Engineering (WCRE), 2012 19th Working Conference on (pp. 113-122). IEEE.	Content Analysis, Authority Graphs Calculation	Conference
18	27	Janes, A., Pfatov, D., Sillitti, A., & Succi, G. (2013). How to calculate software metrics for multiple languages using open source parsers. In Open Source Software: Quality Verification (pp. 264-270). Springer Berlin Heidelberg.	AST of Multiple Source Code Applications.	Conference
19	28	Jovanovic, N., Kruegel, C., & Kirda, E. (2006, May). Pixy: A static analysis tool for detecting web application vulnerabilities. In Security and Privacy, 2006 IEEE Symposium on (pp. 6-pp). IEEE.	Cross Script vulnerability detection	Conference
20	29	Kats, L. C., & Visser, E. (2010, September). Encapsulating software platform logic by aspect-oriented programming: A case study in using aspects for language portability. In Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on (pp. 147-156). IEEE.	SCAM, Interoperability of Heterogeneous Platform.	Conference
21	31	Lehnert, S., Farooq, Q., & Riebisch, M. (2013, March). Rule-based impact analysis for heterogeneous software artifacts. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on (pp. 209-218). IEEE.	Multiperspective Modeling; Horizontal Traceability, SW CIA.	Conference
22	32	Linos, P., Lucas, W., Myers, S., & Maier, E. (2007, November). A metrics tool for multi-language software. In Proceedings of the 11th IASTED International Conference on Software Engineering and Applications (pp. 324-329). ACTA Press.	SW understanding, maintenance, Reuse, Reverse/Re-engineering	Conference

TABLE 20. ***(Continued.)*** **Finalized list of primary studies.**

23	33	Lu Jiang, Zhang, Z., & Zhao, Z. (2013, November). AST Based JAVA Software Evolution Analysis. In Web Information System and Application Conference (WISA), 2013 10th (pp. 180-183). IEEE.	AST, Tree Matching Algorithm, SW Evolution Analysis	Conference
24	34	Marchetto, A., Tonella, P., & Ricca, F. (2012). Rejax: a reverse engineering tool for ajax web applications. IET software, 6(1), 33-49.	Reverse Engineering, Web applications, finite state machine	Journal
25	35	Marinescu, C., & Jurca, I. (2006, September). A meta-model for enterprise applications. In Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC'06. Eighth International Symposium on (pp. 187-194). IEEE.	Reverse Engineering of EAs, Meta Model	Conference
26	36	Mayer, P., & Schroeder, A. (2012, September). Cross-language code analysis and refactoring. In Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on (pp. 94-103). IEEE.	Multi Language Analysis and Refactoring	Conference
27	37	Mayer, P., & Schroeder, A. (2014). Automated multi-language artifact binding and rename refactoring between Java and DSLs used by Java frameworks. In ECOOP 2014–Object-Oriented Programming (pp. 437-462). Springer Berlin Heidelberg.	Multi-Language Artifact Binding and Rename Refactoring	Conference
28	38	Mayer, P., & Schroeder, A. (2013, October). Towards automated cross-language refactoring between Java and DSLs used by Java frameworks. In Proceedings of the 2013 ACM workshop on Workshop on refactoring tools (pp. 5-8). ACM.	Cross-language Linking & Refactoring	Workshop
29	39	Misra, J., Annervaz, K. M., Kaulgud, V., Sengupta, S., & Titus, G. (2012, October). Software clustering: Unifying syntactic and semantic features. InReverse Engineering (WCRE), 2012 19th Working Conference on (pp. 113-122). IEEE.	Code Clustering; Architectural Recovery; Component Discovery,	Conference
30	41	Muhammad, T., Zibran, M. F., Yamamoto, Y., & Roy, C. K. (2013, May). Near-miss clone patterns in web applications: An empirical study of industrial systems. In Electrical and Computer Engineering (CCECE), 2013 26th Annual IEEE Canadian Conference on (pp. 1-6). IEEE.	Management of CLAs. Code Clone Detection,	Conference
31	42	Klint, P., Van Der Storm, T., & Vinju, J. (2009, September). Rascal: A domain specific language for source code analysis and manipulation. In Source Code Analysis and Manipulation, 2009. SCAM'09. Ninth IEEE International Working Conference on (pp. 168-177). IEEE.	Domain Specific Languages, automated software engineering,	Conference
32	44	Perin, F., Girba, T., & Nierstrasz, O. (2010, September). Recovery and analysis of transaction scope from scattered information in Java enterprise applications. In Software Maintenance (ICSM), 2010 IEEE International Conference on (pp. 1-10). IEEE.	SW Maintenance	Conference
33	45	Perin, F. (2010, September). MooseJEE: A moose extension to enable the assessment of JEAs. In ICSM (pp. 1-4).	Software Visualization, Object Oriented Modeling	Conference
34	46	Pfeiffer, R. H., & Wąsowski, A. (2011). Taming the confusion of languages. InModelling Foundations and Applications (pp. 312-328). Springer Berlin Heidelberg.	Inter-Component Dependency Analysis.	Conference
35	47	Pfeiffer, R. H., & Wąsowski, A. (2014). The design space of multi-language development environments. Software & Systems Modeling, 1-29.	Language Integration of MLDE's	Journal
36	48	Pfeiffer, R. H., & Wąsowski, A. (2012). Texmo: A multi-language development environment. In Modelling Foundations and Applications (pp. 178-193). Springer Berlin Heidelberg.	Multilanguage Developing Environments (MLDE)	Conference
37	49	Pfeiffer, R. H., Reimann, J., & Wąsowski, A. (2014). Language-Independent Traceability with Lässig. In Modelling Foundations and Applications (pp. 148-163). Springer International Publishing.	Traceability Detection in MLA's.	Conference
38	50	Philip, M. & Bauer, A. (2015, April). An empirical analysis of the utilization of multiple programming languages in open source projects. In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (p. 4). ACM.	Evaluation of Multiple source code application	Conference
39	51	Polychniatis, T., Hage, J., Jansen, S., Bouwers, E., & Visser, J. (2013, March). Detecting Cross-language Dependencies Generically. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on (pp. 349-352). IEEE.	RE, Cross languages parsing & dependencies, SW maintenance.	Conference
40	52	Roubtsov, S., Serebrenik, A., Mazoyer, A., van den Brand, M. G., & Roubtsova, E. (2013). I2SD: reverse engineering Sequence Diagrams from Enterprise Java Beans with interceptors. IET software, 7(3), 150-166.	SCAM, Aspect Oriented Programming, RE, SW Maintenance	Conference
41	53	Schink, H., Kuhlemann, M., Saake, G., & Lämmel, R. (2011). Hurdles in Multi-Language Refactoring of Hibernate Applications. In ICSOFT (2) (pp. 129-134).	OOP/ SW & DB Technologies	Conference
42	54	Schink, H. (2013, September). SQL-schema-compare: Support of multi-language refactoring with relational databases. In Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on (pp. 173-178). IEEE.	Multi-Language analysis & Refactoring	Conference
43	55	Sebastian, E., Rendel, T., Kästner, C., & Ostermann, K. (2011, October). Sugarj: library-based syntactic language extensibility. In ACM SIGPLAN Notices (Vol. 46, No. 10, pp. 391-406). ACM.	Sugar libraries, SW Modeling, Reuse, DSLs	Conference
44	56	Strein, D., Kratz, H., & Lowe, W. (2006, September). Cross-language program analysis and refactoring. In Source Code Analysis and Manipulation, 2006. SCAM'06. Sixth IEEE International Workshop on (pp. 207-216). IEEE.	Multi-Language Analysis & Refactoring, Program Understanding	Conference
45	57	Synytskyy, N., Cordy, J. R., & Dean, T. R. (2003, October). Robust multilingual parsing using island grammars. In Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research (pp. 266-278). IBM Press.	Automated source code analysis, comprehension & transformation	Conference
46	56	Terceiro, A., Costa, J., Miranda, J., Meirelles, P., Rios, L. R., Almeida, L., & Kon, F. (2010, September). Analyze an extensible multi-language source code analysis and visualization toolkit. In Brazilian Conference on Software: Theory and Practice (Tools Session).	Source Code Analysis & Visualization	Conference
47	59	Tichelaar, S., Ducasse, S., Demeyer, S., & Nierstrasz, O. (2000). A meta-model for language-independent refactoring. In Principles of Software Evolution, 2000. Proceedings. International Symposium on (pp. 154-164). IEEE.	OOPLs, Refactoring, Type related Analysis	Conference
48	60	Mayer, P., & Schroeder, A. (2013, May). Patterns of cross-language linking in java frameworks. In Program Comprehension (ICPC), 2013 IEEE 21st International Conference on (pp. 113-122). IEEE.	Understanding, Analysis & Refactoring. Cross-Language Patterns	Conference
49	61	Tomassetti, F., Vetrò, A., Torchiano, M., Voelter, M., & Kolb, B. (2013, May). A model-based approach to language integration. In Proceedings of the 5th International Workshop on Modeling in Software Engineering (pp. 76-81). IEEE Press.	Cross-language Integration; Multilanguage Refactoring	Conference
50	62	Tonella, P., & Ricca, F. (2008, October). Dynamic model extraction and statistical analysis of Web applications: Follow-up after 6 years. In Web Site Evolution, 2008. WSE 2008. 10th International Symposium on (pp. 3-10). IEEE.	SW Evolution, Reverse Engineering;	Conference
51	63	Van der Storm, T., & Vinju, J. J. (2015). Towards multilingual programming environments. The science of Computer Programming, 97, 143-149.	Language Multiplicity/Interoperability; Meta Programming	Journal
52	64	Van Dam, J. K. and V. Zaytsev (2016). Software Language Identification with Natural Language Classifiers. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE.	Source code identification using NLP Classifiers	Conference
53	65	Van Dam, J. K. (2016). "Identifying source code programming languages through natural language processing." Master Thesis, University of Amsterdam.	Source code detection using NLP techniques	Thesis
54	66	Xia, X., Lo, D., Wang, X., Zhang, C., & Wang, X. (2014, June). Cross-language bug localization. In Proceedings of the 22nd International Conference on Program Comprehension (pp. 275-278). ACM.	Program Comprehension	Conference
55	67	Vazdanshenas, A. R., & Moonen, l. (2011, September). Crossing the boundaries while analyzing heterogeneous component-based software systems. In Software Maintenance (icsm), 2011 27th IEEE International Conference on (pp. 193-202). IEEE.	SW Maintenance; Knowledge Engineering, Program Slicing	Conference
56	68	Yohann, C. & Candea, G. (2008, December). Deprogramming Large Software Systems. In HotDep.	SW Dependency, Deprogramming Large Software applications.	Workshop

understanding about MLSCA, its applicability, and prospects of software engineering domains. In this research, we presented a review of the research contributions presented in the form of models, tools, domains and techniques to analyze multilingual applications. In order to extract precise pragmatic evidence, we devised review methodology, focused domains, research questions, sources of information from renowned publishers, inclusion & exclusion criteria, basic search string, type of information criteria and study assessment criteria. We searched 3820 research papers

TABLE 21. Programming languages.

Sr. #	Languages	2	3	12	16	17	18	20	22	23	24	25	26	27	29	30	32	33	34	35	36	39	40	42	43	44	45	46	47	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	65	66	67	68
1	.Net															√									√										√													
2	Ada					√																																										
3	Android App																		√																														
4	Apache Meetings																																						√								
5	C/C++/C#			√		√	√		√			√		√																					√		√							√	√	√
6	Cobol					√																																									
7	Coral																									√																					
8	CORBA																													√																
9	DSLs		√											√			√			√	√		√				√		√	√				√					√								
10	Groovy																										√		√																	
11	HBM																			√	√																										
12	Hibernate																			√	√											√														
13	HQL																			√	√													√					√							
14	HTML/Applets							√		√			√				√			√	√	√		√	√		√	√			√					√					√			√	√
15	Intermediate/ Byte Code	√														√																			√								√				
16	Java/J#			√	√	√	√		√	√	√	√		√	√			√		√	√		√			√	√	√	√	√		√		√	√		√	√	√	√			√	√	√	
17	JEA/EJB																							√	√				√		√									√	√					
18	Jscript							√				√					√					√			√		√	√						√		√								√	√
19	J Servlet																								√																					
20	JSP								√															√	√					√	√														
21	J-Unit														√																															
22	MPLs																																						√		√					
23	OOPLs			√	√	√	√		√	√	√	√		√	√	√		√		√	√		√		√	√	√	√	√		√		√	√	√	√	√	√	√			√	√	√	√
24	OWL/RDF		√												√								√																							
25	PHP												√				√					√																							√	√
26	Prolog																																	√												
27	Python						√		√																																			√		
28	Ruby																		√																									√			
29	SCRO		√																																											
30	Small Talk			√																																		√								
31	Spring-																			√										√								√							
32	SQL/RDBMS					√					√							√											√		√	√									√					
33	Stratego													√																				√												
34	UML														√															√											√	√	√				
35	URN														√																																
36	VB																																		√	√											
37	Wicket																			√																											
38	XML							√		√							√										√	√						√						√						
39	XSS/CSS								√		√		√				√					√																									
40	Xtend																												√																	

from selected research venues during January 2001 to January 2016. By applying 3 staged rigorous selection criteria, we finalized 56 publications for further evaluation. The detailed information of these selected publications is presented in the form of publisher, proceedings, year and country of publication along with the research topic in Table 6. It is found that most of the software applications are multilingual in nature and this trend is creeping day by day. It is observed that MLSCA domain is quite widespread and most recent for the research community. We found that maximum research started from 2010 (i.e. 75%) and this tendency is inclining with the passage of time as 23.2% of the whole publications during the last two years. The quality assessment criteria determine relevance, adequacy, quality and validity of the research domain appropriate to the selected publications. The overall assessment with respect to prescribed criteria is 66% which is quite aligned with the research topic. The results of assessment criteria are presented in Table 18 (Appendix).

A. RESEARCH PROBLEMS

Identification of research requirements and issues is the first step to building the research foundation of literature review findings. In this research, we recognized 46 different research problems and requirements to analyze multilingual applications. Each issue is marked with PR #). The issues and requirements are divided into 13 different software

TABLE 22. SW engineering domains.

Domains/References	ABV	1	2	3	4	7	11	12	16	17	18	20	21	22	23	24	25	26	27	29	30	31	32	33	34	35	36	37	39	40	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68
Aspect Oriented Programming	AOP																																						√																		
Change Impact Analysis	CIA				√	√	√							√						√																																						
Code Clone/ Clustering	CCD										√			√														√	√																												
Code/ Graph Parsing	CGP								√	√											√											√													√												
Component Discovery	CMD																										√																												√				
Content Analysis	CAN															√																																											
Data Flow Analysis	DFA																	√																																									
De-Programming	DEP																																																								√				
Dynamic Analysis	DYA	√										√											√						√	√																					√		√				√				
Intermediate/Bytecode Analysis	BCA																				√																												√								√				
Knowledge Engineering	KEG														√																																										√				
Language Integration/ Interoperability	LIN/ LIO	√	√				√		√										√													√	√																				√		√						
Lexical Analysis	LXA																																											√		√												
Machine Intelligence	MIN																																																											
Meta-Model	MM											√			√									√						√														√	√										√				
Multilanguage IDEs	MLDE						√					√																		√				√	√							√	√	√									√				
Natural Language Processing	NLP																																																									√	√	√
Pattern Analysis	PTA																				√																																				√		√			
Program Slicing/ Chopping	PSC								√																																																	√				
Program Understanding	PRU						√								√				√						√																							√				√								
Re-Engineering	REg						√													√																			√																		√		
Refactoring	RFT						√																		√	√	√														√	√		√				√	√	√				√				
Reuse	REU			√			√				√													√																				√												√			
Reverse Engineering	REV	√											√	√	√									√				√				√				√					√									√						√	√	
Software Visualization	SVS																													√	√															√											
Statistical Analysis	STA							√				√		√	√			√						√						√	√												√	√		√			√		√			√	√	√	√
SW Comprehension	SWC				√	√							√		√													√																	√			√				√						
SW Debugging	SWD		√																																																								
SW Evolution	SWE		√			√	√													√		√																											√		√			√			
SW Maintenance	SMN					√	√						√							√											√									√	√												√				
SW Modeling	SWM		√				√																										√	√	√																					
Traceability Analysis Dependency Analysis	TRA DPA				√		√		√							√				√			√									√			√		√															√				
Tree matching algorithm AST/Call Graph	TMA													√			√					√									√																								√				
Vulnerability Detection	VLD																	√																																								
Cross Lang Links XLL	XLL						√																			√	√										√				√		√						√	√			√				

engineering domains. The majority of the issues include Static Analysis 26%, Program Understanding & Reverse Engineering 15%, Cross Language Link Detection 13%, Dynamic Analysis 11%, Analysis of Enterprise Applications 11% and Multiple Language Integrated Development Environments 11%. The remaining portion of issues is less than 10%. The summary of reported problems and requirements of multilingual source code analysis is presented in Fig 5 and the complete detail is mentioned in Table 7.

B. RESEARCH CONTRIBUTIONS

In this research, we identified the support of multilingual source code applications, languages, and domains in the form of research models, platforms, frameworks, prototype models or case studies. The features of selected primary studies with their strengths and limitations are highlighted in the respective Tables. Each research contribution is marked with CN # (contribution number) as a solution of particular problem or requirement which is mapped by PR # (problem/requirement number). The contributions are separated into 38 tools & approaches, 14 techniques, and 4 surveys. They focus general purpose languages, Domain Specific Languages/ Meta-languages, and intermediate representations. Each research offers separate analysis mechanism and supports to accomplish their requirements. They focus multiple domains of software engineering. The analysis tools are further divided into static analysis tools (26), dynamic analysis tools (8) and semantic/hybrid analysis tools (4). Each tool represents the model, analysis mechanism, source code languages and experimental case studies. These tools are evaluated through frameworks, development environments, case studies, prototypes, plug-ins, and simulators. The results show that proposed model and analysis mechanism of each contribution support different languages and are

TABLE 23. Tools for multilingual source code analysis.

Sr. #	Tool	Model/Representation	Mechanism	Source Code Languages	Experimental Evaluation
1	Normir [2]	Holistic Debugging: Analysis Modeling and Simulation.	Dynamic Analysis	Intermediate/ Byte Code representation	Simics: Instruction set Simulator
2	RECOS [3]	Semantic knowledge base & point to analysis technique. Explicit ontological code representations.	Semantic Analysis	SCRO, OWL, RDF, RDFS	Eclipse tool
3	Pangea [12]	Language independent meta-model Repository OMS, analyze MLAs.	Static Analysis	VerveineJ: Java, Smalltalk, C/C++, C#,	Famix (Java, C, C++) Analyzer (Smalltalk, C)
4	JOANA [16]	Dependence graph calculation Language security algorithm. Java Object Sensitive Analysis	Programming Slicing, Program Chopping	Java based languages	Eclipse Plug-in
5	GUPRO [17, 28].	Use graph querying & graph algorithms extract source code into a graph repository.	Source Code Parsing	COBOL, CSP, Ada, C MVS/JCL, PSB, SQL.	GEOS, XFIG, COBOL etc
6	WARE [20]	Reverse engineering MDWE, Meta-Model	Static and Dynamic Analysis	HTML JavaScript, XML	General Examples
7	Diff/TS [22]	Analyze structural changes Extend string differentiating algorithms.	Structural Analysis, Fine-grained Analysis.	Python, Java, C and C++	Case Study: Emacs editor
9	TARENTe [24]	Adhoc modular framework. Extract, explore, analyze web docs	Content Analysis. Authority Graph Analysis.	Java, my SQL	Open-source code WAs.
10	Metrics Tool for MLAs [25]	Open Source Parser, extract AST of MLAs. Extend Eclipse CDT Parser.	Syntactic Analysis, Semantic Analysis	C, C++, Java and JavaScript	Eclipse IDE
11	Pixy [26]	Detects cross-site & taint-style scripting vulnerabilities in WAs.	Static Analysis, Context /Flow Sensitive Analysis	PHP, XSS, HTML	Open source Java tool, evaluated on PHP scripts
12	AOP for XL Portability [28]	Program transformation tool, provide aspects of managing interoperability in HAs	Dynamic Analysis, Aspect Weaving Mechanism.	Java, DSL, C, JVM	Tool: Stratego (Eclipse), Case Study
13	EMFTrace [29]	Multi-perspective impact analysis to analyze CIA in artifacts of MLAs.	Change Impact Analysis. Dependency Analysis	Java, JUnit test cases, UML, URN, OWL.	Case study EMFTrace, Eclipse Framework
14	MMT [30]	Parse MSIL (Microsoft Intermediate Language)	Complexity analysis of MLAs.	.Net based languages	NHibernate, MMT, Timecard CS Client
15	REAJAX [32]	DOM GUI-based, Build Finite State Models (FSM)	Dynamic Analysis, Trace Link Analysis.	HTML, CSS, JScript, PHP, XML	http://pafm.source.net, http://tudu.source.net
16	DATES [33]	A meta-model for EAs, Third party API. Recover relational/ object-oriented entities.	Static Analysis	Java, SQL	Evaluated on 3 case studies KITTA, TRS, SALARY
17	XLL & Refactoring [36]	Generalized approach: Cross lang. binding & refactoring	XLL Analysis, Code Refactoring	Java, HTML, DSLs(HQL, HBM)	Jtrac support Spring, Hibernate, and Wicket
18	Code Clone Detection [39]	Use patterns to exact near-miss code clones in WAs.	Dynamic Analysis, Code Clone Detection	HTML, PHP, JavaScript, CSS, MVC	Clone detectors VisCad, NiCad. Industrial Was
19	Rascal [40]	Meta Model. Analyze HAs at a high level of integration.	Syntax Analysis. Dynamic Analysis, Semantic Links.	DSLs ASF, SDF, RScript, Java	Eclipse IDE
20	FAMIX [42]	Expose and analyze transaction scope in EAs.	Statistical Analysis, SW Visualization, Maintenance.	EJB, JSP, HTML, Applets	FAMIX, Moose, Eclipse: Plugin.
21	MooseJEE [43]	Code browser, Visualizations. Analyze architectural variants.	Static Analysis, Maintenance & Modeling, Transactional analysis	Java Beans, JSP, JavaScript, HTML, Servlet	FAMIX platform
22	GenDeMoG [44]	Language independent high level model. Inter component dependency patterns.	Dynamic Analysis. Explicit Dependencies Analysis.	Java & DSLs	Eclipse plug-in, Case Study: OFBiz.
23	MLDE's Design Space [45]	Integration of MLDEs. Search based relation. Track XLLs	Static Analysis & Multi-modeling of MLDEs.	Java, JScript, HTML, XML. Groovy, Coral	Tool + Case study (TexMo, Coral)
24	TexMo [46]	Analyze MLDEs using explicit relation model.	Static Analysis, Visualization	Java, JavaScript, HTML and XML	Case study (JTrac)
25	Lässig [47]	Language independent Integrate traceability to modeling frameworks.	Dependency Analysis, SW Modeling	Extend, Java, and Groovy	Autonomous trace links modeling framework.
26	I2SD [50]	EJB interceptors to sequence diagrams. Modular pipe filter architecture.	Dynamic Analysis. Reverse Engineering. AOP,	EJB, JSP, HTML, CORBA, UML	NetBeans IDE, DataPortal, WasabiBeans,
27	Refactoring MLAs [51]	ORM bw Java & DB Schema. Rename Method and Push Down Method for MLAs.	Multi-Language Structural Analysis & Refactoring	Java, Hibernate, and SQL	Hibernate Application (HRM)
28	SQL Schema Comparer [52]	Schema compare library detect changes & validate SQL schemes	Database Analysis & Refactoring MLAs.	SQL DBMS	Eclipse plug-in
29	SugarJ [53]	Java-based extensible language. Unique parsing mechanism	Syntax Analysis Random Context Free & Layout Sensitive.	DSLs, JavaScript, Prolog and Haskell	Spoofax-based IDE, five case studies
30	X-Develop [54]	Language Independent meta model to analyze & refactor MLAs. Intermediate representation as AST.	Static low-level analysis,	C#, J#, VB	IText, .Net
31	Island Grammar [55]	Concurrent & robust parsing of MLAs in form of a parse tree.	Lexical Analysis, CFG.	VB, HTML, Jscript, Rascal	McCabe-complexity
32	Analizo [56]	Layered Style, Doxygen Parser. Generates source code metrics.	SW Evolution, Visualization Dependency Analysis	C, C++, Java	VLC project, Analyze
33	MOOSE [57]	Language independent Meta model & Refactoring OOPLs.	Refactoring, Visualization, Type related Analysis	Java, Smalltalk	FAMIX model, Prototype tool
34	Dynamic Analysis & Model Extr[60]	Extract UML model of a Web application model through its execution	Dynamic Analysis. Static Analysis. SW Evolution.	Web Applications	ReWeb: Spider (Explicit-state model of WAs)
35	CrosLocator [62]	Bug localization algorithm to rank source code files in MLAs.	Program Comprehension	Ruby	Ruby-China
36	LISA [63]	Language independent parsing. Translate AST into a graph structure.	Static Analysis, SW Evolution	Java based languages	JGit repositories, AspectJ's
37	KDM [64]	Knowledge discovery Metamodel, RE homogeneous model from heterogeneous artifacts.	Static Analysis, Program Slicing, Maintenance, Knowledge eng.	C/C++, Java	Code Surfer Prototype plugin. Industrial app code.
38	DeP [65]	Deprogramming. Analyze code patterns. Patterns recognition.	Static analysis, Pattern Analysis, Refactoring,	Java	DeP tool

evaluated through their respective case studies. Moreover, 14 different techniques and 4 surveys are presented to analyze multilingual applications. The analysis techniques are presented in the form of a models, frameworks or algorithms. The analysis models include mathematical, graphical, UML, semantic or high-level models. The techniques are

TABLE 24. Conference & workshop proceedings.

S. #	Acronyms	Conference/Workshop Name
1	CBSoft	Brazilian Conference on Software: Theory and Practice.
2	CCECE	Canadian Conference on Electrical and Computer Engineering
3	CSMR	Conference on Software Maintenance and Reengineering
4	ECMFA	European Conference on Modeling Foundations and Applications
5	ECOOP	European Conference on Object-Oriented Programming
6	FOSE	Future of Software Engineering Conference
7	ICANLI	International Conference on Applications of Natural Language to Information Systems
8	ICASE	International Conference on Automated Software Engineering
9	ICICT	International Conference on Information and Communication Technologies
10	ICPC	International Conference on Program Comprehension
11	ICSM	International Conference on Software Maintenance
12	ICSOFT	International Conference on Software and Database Technologies
13	ICSR	International Conference on Software Reuse
14	IRI	International Conference on Information Reuse and Integration
15	MiSE	International Workshop on Modeling in Software Engineering
16	MMAS	International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems
17	SCAM	International Conference on Source Code Analysis and Manipulation
18	WCRE	Working Conference on Reverse Engineering
19	WISA	International Conference on Web Information System and Application
20	WRT	Workshop on Refactoring Tools
21	WWSE	Workshop on Website Evaluation
S. #	Journal Publications	
1	IEEE Transactions on Pattern Analysis and Machine Intelligence	
2	Journal of Science of Computer Programming	
3	Journal of Software & Systems Modeling	

evaluated mathematically, through case studies or by using some framework/platform. The surveys or reviews highlight the importance of source code analysis, multi-language IDEs, cross-language analysis, refactoring and empirical evaluation of multi-language applications. The detail of research contributions is mentioned in Tables 9-17.

C. SOURCE CODE LANGUAGES

A number of multidiscipline languages are discussed in this literature review in the form of General Purpose Languages (GPLs) containing C/C++, Java,Net, HTML etc, Domain Specific Languages (DSLs) /Meta Programming Languages (MPLs) comprising XML, Spring, Xtend, RDF etc and Intermediate Representations (Byte Code). These languages target web based applications, enterprise applications, heterogeneous applications, integrated development environments, cross language applications, embedded applications and legacy applications. The collection of selected research supports 40 different kinds of programming languages. Maximum numbers of languages discussed are from 9 to 2 languages per publication. The research trends are more likely towards analysis of OOP 57%, Java 53%, HTML/Applets 26%, DSLs 21% and C/C++/C# 17%. The remaining languages have less contribution than 17 %. The summary of source code languages is mentioned in Fig 6, whereas the detail is provided in Table 21 (appendix).

D. DOMAINS FOR SOURCE CODE ANALYSIS

The research trends identified to analyze multilingual applications are grouped into 34 domains, recognize almost from every field of software engineering. The detail is provided in Table 22 (appendix). During analysis, it is found that the research community is inclined towards Static Analysis 25%, Program Understanding & Reverse Engineering 22%, XLL Detection 16%, Dynamic Analysis 15%, Source Code Refactoring 11%, Change Impact Analysis 9% and Multi Language IDEs 9%, whereas the contribution of remaining domains is less than 9% (summarized in Fig 7).

E. ISSUES AND FUTURE RESEARCH

During the review, it is observed that this research area is widely spread across various software engineering domains, continuously growing and has an opportunity for further improvements and enhancements. Various MLSCA areas are still open for further research as mentioned in the assessment of Q3. There are 13 domains that highlight research prospects for future research. Table 17 describes 56 limitations and future work associated with their respective domains. These domains include Static Analysis 25%, Program Understanding & RE 22%, XLL Detection 16%, Dynamic Analysis 15%, Source Code Refactoring 11%, Change Impact Analysis 9% and Multiple Language IDEs 9%. Whereas, the rest of the domains are less than 9% (summarized in Figure 8). The highlighted domains require advancement in performance

TABLE 25. Abbreviations.

Acronym	Descriptions	Acronym	Descriptions
AST	Abstract Syntax Tree	MLR	Multi Language Refactoring
CIA	Change Impact Analysis	MLSCA	Multilingual Source Code Analysis
CLA	Cross-Language Applications	MPLs	Multiple Programming Languages
CLR	Cross-Language Refactoring	NM	Not Mentioned
CNF	Conference	OOP	Object Oriented Programming
CSDG	conditional system dependence graphs	OMS	Object Model Snapshot
DB	Database	ORM	Object Relation Model
DSLs	Domain Specific languages	RE	Reverse Engineering
FSM	Finite State Model	SCAM	Source Code Analysis & Manipulation
GPLs	General Purpose Programming Languages	SCRO	Source code reference ontology
HAs	Heterogeneous Applications	SPLs	Software Product Lines
ICT	Information & Communication Technologies	SW	Software
IDE	Integrated Development Environment	SWAs	Software Applications
IFC	Information Flow Control	WAs	Web Applications
IRI	Information Reuse and Integration	WKS	Workshop
IS	Information Systems	XLAs	Cross-Language Applications
JEAs	Java Enterprise Applications	XLL	Cross Language Link
Jrnls	Journals	XSS	Cross Site Scripting
MLAs	Multi-Language Software Applications		
SLI	Source Code Language Identification		

and accuracy of analysis mechanism, scalability of parsing techniques, extraction and visualization of cross-language artifacts, their relationships and dependencies. Completeness in source code security, coverage, clone detection, similarity analysis, change impact, classification and evolution analysis are required in multilingual applications. Moreover, managing source code repositories, portability across multiple language platforms and document generation from multilingual source code are the areas that need attention for future research.

APPENDIX

See Tables 18–25.

REFERENCES

[1] C. Ackermann, M. Lindvall, and R. Cleaveland, "Recovering views of inter-system interaction behaviors," in *Proc. Reverse Eng. WCRE*, Oct. 2009, pp. 53–61.

[2] L. Albertsson, "Holistic debugging–enabling instruction set simulation for software quality assurance," in *Proc. 14th IEEE Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2006, pp. 96–103.

[3] C. V. Alexandro and H. C. Gall, "Rapid multi-purpose, multi-commit code analysis," in *Proc. 37th Int. Conf. Softw. Eng.*, vol. 2. May 2015, pp. 635–638.

[4] A. Alnusair, T. Zhao, and E. Bodden, "Effective API navigation and reuse," in *Proc. IEEE Int. Conf. Inf. Reuse Integr. (IRI)*, Aug. 2010, pp. 7–12.

[5] F. Angerer, "Variability-aware change impact analysis of multi-language product lines," in *Proc. 29th ACM/IEEE Int. Conf. Autom. Softw. Eng.*, Sep. 2004, pp. 903–906.

[6] T. B. C. Arias, P. Avgeriou, and P. America, "Analyzing the actual execution of a large software-intensive system for determining dependencies," in *Proc. Reverse Eng. WCRE*, Oct. 2008, pp. 49–58.

[7] A. Aryani, F. Perin, M. Lungu, A. N. Mahmood, and O. Nierstrasz, "Can we predict dependencies using domain information?" in *Proc. Reverse Eng. WCRE*, 2011, pp. 55–64.

[8] A. Aryani, I. D. Peake, and M. Hamilton, "Domain-based change propagation analysis: An enterprise system case study," in *Proc. ICSM*, 2010, pp. 1–9.

[9] B. Kitchenham *et al.*, "Systematic literature reviews in software engineering - A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, 2009.

[10] B. Kitchenham, "Procedures for undertaking systematic reviews: Joint technical report," Dept. Comput. Sci., Keele Univ., Keele, U.K., Tech. Rep. TR/SE-0401, 2004.

[11] B. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proc. 26th Int. Conf. Softw. Eng.*, May 2004, pp. 273–281.

[12] D. Binkley, "Source code analysis: A road map," in *Proc. Future Softw. Eng.*, May 2007, pp. 104–119.

[13] A. Caracciolo, A. Chis, B. Spasojevic, and M. Lungu, "Pangea: A workbench for statically analyzing multi-language software corpora," in *Proc. IEEE 14th Int. Working Conf. Source Code Anal. Manipulation (SCAM)*, Sep. 2014, pp. 71–76.

[14] (2007). *Centre for Reviews and Dissemination, what are the Criteria for the Inclusion of Reviews on DARE?*. [Online]. Available: http://www.york.ac.uk/inst/crd/faq4.htm

[15] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, Jan. 1990.

[16] A. Cleve, T. Mens, and J. L. Hainaut, "Data-intensive system evolution," *Computer*, vol. 43, no. 8, pp. 110–112, 2010.

[17] D. Giffhorn and C. Hammer, "Precise analysis of java programs using joana," in *Proc. 8th IEEE Int. Working Conf. Sour. Code Anal. Manipulation*, Sep. 2008, pp. 267–268.

[18] J. Ebert, B. Kullbach, V. Riediger, and A. Winter, "Gupro-generic understanding of programs an overview," *Electron. Notes Theor. Comput. Sci.*, vol. 72, no. 2, pp. 47–56, 2002.

[19] E. Flores, A. Barrón-Cedeño, P. Rosso, and L. Moreno, "Towards the detection of cross-language source code reuse," in *Natural Language Processing and Information Systems*. Berlin, Germany: Springer, 2011, pp. 250–253.

[20] E. Flores, A. Barrón-Cedeño, P. Rosso, and L. Moreno, "Cross-Language Source Code Re-Use Detection Using Latent Semantic Analysis," *J. Univ. Comput. Sci.*, vol. 21, no. 13, pp. 1708–1725, 2015.

[21] G. CanforaHarman and M. Di Penta, "New frontiers of reverse engineering," in *Proc. Future Softw. Eng.*, May 2007, pp. 326–341.

[22] G. A. Di Lucca *et al.*, "WARE: A tool for the reverse engineering of Web applications," in *Proc. 6th Eur. Conf. Softw. Maintenance Reeng.*, 2002, pp. 241–250.

[23] M. Harman, "Why source code analysis and manipulation will always be important," in *Proc. SCAM*, 2010, pp. 7–19.

[24] M. Hashimoto and A. Mori, "Diff/TS: A tool for fine-grained structural change analysis," in *Proc. 15th Working Conf. Reverse Eng. (WCRE)*, Oct. 2008, pp. 279–288.

[25] B. Hugo, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: A generic and extensible framework for model driven reverse engineering," in *Proc. IEEE/ACM Int. Conf. Autom. Softw. Eng.*, Sep. 2010, pp. 173–174.

[26] M. Janardan, K. M. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software clustering: Unifying syntactic and semantic features," in *Proc. 19th Working Conf. Reverse Eng. (WCRE)*, Oct. 2012, pp. 113–122.

[27] A. Janes, D. Piatov, A. Sillitti, and G. Succi, "How to calculate software metrics for multiple languages using open source parsers," in *Open Source Software: Quality Verification*. Berlin, Germany: Springer, 2013, pp. 264–270.

[28] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting Web application vulnerabilities," in *Proc. IEEE Symp. Secur. Privacy*, May 2006, p. 6.

[29] L. C. Kats and E. Visser, "Encapsulating software platform logic by aspect-oriented programming: A case study in using aspects for language portability," in *Proc. 10th IEEE Working Conf. Sour. Code Anal. Manipulation (SCAM)*, Sep. 2010, pp. 147–156.

[30] B. Kullbach, A. Winter, P. Dahm, and J. Ebert, "Program comprehension in multi-language systems," in *Proc. 15th Working Conf. Reverse Eng.*, Oct. 1998, pp. 135–143.

[31] S. Lehnert, Q. Farooq, and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng. (CSMR)*, Mar. 2013, pp. 209–218.

[32] P. Linos, W. Lucas, S. Myers, and E. Maier, "A metrics tool for multi-language software," in *Proc. 11th IASTED Int. Conf. Softw. Eng. Appl.*, Nov. 2007, pp. 324–329.

[33] L. Jiang, Z. Zhang, and Z. Zhao, "AST based JAVA software evolution analysis," in *Proc. Web Inf. Syst. Appl. Conf. (WISA)*, Nov. 2013, pp. 180–183.

[34] A. Marchetto, P. Tonella, and F. Ricca, "ReAjax: A reverse engineering tool for Ajax Web applications," *IET Softw.*, vol. 6, no. 1, pp. 33–49, 2012.

[35] C. Marinescu and I. Jurca, "A meta-model for enterprise applications," in *Proc. 8th Int. Symp. Symbolic Numer. Algorithms Sci. Comput. (SYNASC)*, Sep. 2006, pp. 187–194.

[36] P. Mayer and A. Schroeder, "Cross-language code analysis and refactoring," in *Proc. IEEE 12th Int. Working Conf. Sour. Code Anal. Manipulation (SCAM)*, Sep. 2012, pp. 94–103.

[37] P. Mayer and A. Schroeder, "Automated multi-language artifact binding and rename refactoring between Java and DSLs used by Java frameworks," in *ECOOP Object-Oriented Programming*. Berlin, Germany: Springer, 2014, pp. 437–462.

[38] P. Mayer and A. Schroeder, "Towards automated cross-language refactoring between Java and DSLs used by Java frameworks," in *Proc. ACM Workshop Refactoring Tools*, Oct. 2013, pp. 5–8.

[39] J. Misra, K. M. Annervaz, V. Kaulgud, S. Sengupta, and G. Titus, "Software clustering: Unifying syntactic and semantic features," in *Proc. 19th Working Conf. Reverse Eng. (WCRE)*, 2012, pp. 113–122.

[40] *MoDISCO Eclipse*. [Online]. Available: http://www.eclipse.org/MoDisco/

[41] T. Muhammad, M. F. Zibran, Y. Yamamoto, and C. K. Roy, "Near-miss clone patterns in Web applications: An empirical study with industrial systems," in *Proc. 26th Annu. IEEE Can. Conf. Elect. Comput. Eng. (CCECE)*, May 2013, pp. 1–6.

[42] P. Klint, T. van Der Storm, and J. Vinju, "Rascal: A domain specific language for source code analysis and manipulation," in *Proc. 9th IEEE Int. Working Conf. Sour. Code Anal. Manipulation (SCAM)*, Sep. 2009, pp. 168–177.

[43] M. Pautasso, "Ten simple rules for writing a literature review," *PLoS Comput. Biol.*, vol. 9, no. 7, p. e1003149, 2013.

[44] F. Perin, T. Girba, and O. Nierstrasz, "Recovery and analysis of transaction scope from scattered information in Java enterprise applications," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2010, pp. 1–10.

[45] F. Perin, "MooseJEE: A moose extension to enable the assessment of JEAs," in *Proc. ICSM*, Sep. 2010, pp. 1–4.

[46] R. H. Pfeiffer and A. Wasowski, "Taming the confusion of languages," in *Modelling Foundations and Applications*. Berlin, Germany: Springer, 2011, pp. 312–328.

[47] R. H. Pfeiffer and A. Wasowski, "The design space of multi-language development environments," in *Proc. Softw. Syst. Modeling*, 2014, pp. 1–29.

[48] R. H. Pfeiffer and A. Wasowski, "Texmo: A multi-language development environment," in *Modelling Foundations and Applications*. Berlin, Germany: Springer, 2012, pp. 178–193.

[49] R. H. Pfeiffer, J. Reimann, and A. Wasowski, "Language-Independent Traceability with Lässig," in *Modelling Foundations and Applications*. Berlin, Germany: Springer, 2014, pp. 148–163.

[50] M. Philip and A. Bauer, "An empirical analysis of the utilization of multiple programming languages in open source projects," in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, Apr. 2015, p. 4.

[51] T. Polychniatis, J. Hage, S. Jansen, E. Bouwers, and J. Visser, "Detecting cross-language dependencies generically," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng. (CSMR)*, 2013, pp. 349–352.

[52] S. Roubtsov, A. Serebrenik, A. Mazoyer, M. G. van den Brand, and E. Roubtsova, "I2SD: Reverse engineering sequence diagrams from enterprise Java beans with interceptors," *IET Softw.*, vol. 7, no. 3, pp. 150–166, 2013.

[53] H. Schink, M. Kuhlemann, G. Saake, and R. Lämmel, "Hurdles in multi-language refactoring of hibernate applications," in *Proc. ICSOFT*, 2011, pp. 129–134.

[54] H. Schink, "SQL-schema-comparer: Support of multi-language refactoring with relational databases," in *Proc. IEEE 13th Int. Working Conf. Sour. Code Anal. Manipulation (SCAM)*, Sep. 2013, pp. 173–178.

[55] E. Sebastian, T. Rendel, C. Kästner, and K. Ostermann, "SugarJ: Library-based syntactic language extensibility," *ACM SIGPLAN Notices,*, vol. 46, no. 10, pp. 391–406, Oct. 2011.

[56] D. Strein, H. Kratz, and W. Lowe, "Cross-language program analysis and refactoring," in *Proc. 6th IEEE Int. Workshop Sour. Code Anal. Manipulation (SCAM)*, Sep. 2006, pp. 207–216.

[57] N. Synytskyy, J. R. Cordy, and T. R. Dean, "Robust multilingual parsing using island grammars," in *Proc. Conf. Centre Adv. Stud. Collaborative Res.*, 2003, pp. 266–278.

[58] A. Terceiro *et al.*, "Analizo: An extensible multi-language source code analysis and visualization toolkit," in *Proc. Brazilian Conf. Softw., Theory Practice, Tools Session)*, Jan. 2010, p. 107.

[59] S. Tichelaar, S. Ducasse, S. Demeyer, and O. Nierstrasz, "A meta-model for language-independent refactoring," in *Proc. Int. Symp. Principles Softw. Evol.*, 2000, pp. 154–164.

[60] P. Mayer and A. Schroeder, "Patterns of cross-language linking in Java frameworks," in *Proc. IEEE 21st Int. Conf. Program Comprehension (ICPC)*, May 2013, pp. 113–122.

[61] F. Tomassetti, A. Vetró, M. Torchiano, M. Voelter, and B. Kolb, "A model-based approach to language integration," in *Proc. 5th Int. Workshop Modeling Softw. Eng.*, May 2013, pp. 76–81.

[62] P. Tonella and F. Ricca, "Dynamic model extraction and statistical analysis of Web applications: Follow-up after 6 years," in *Proc. 10th Int. Symp. Web Site Evol. (WSE)*, Oct. 2008, pp. 3–10.

[63] T. van der Storm and J. J. Vinju, "Towards multilingual programming environments," *Sci. Comput. Program.*, vol. 97, pp. 143–149, Jan. 2015.

[64] J. K. van Dam and V. Zaytsev, "Software language identification with natural language classifiers," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2016, pp. 624–628.

[65] J. K. van Dam, "Identifying source code programming languages through natural language processing," M.S. thesis, Faculty Sci., Math. Inform., Univ. Amsterdam, Amsterdam, The Netherlands, 2016.

[66] X. Xia, D. Lo, X. Wang, C. Zhang, and X. Wang, "Cross-language bug localization," in *Proc. 22nd Int. Conf. Program Comprehension*, Jun. 2014, pp. 275–278.

[67] A. R. Yazdanshenas and L. Moonen, "Crossing the boundaries while analyzing heterogeneous component-based software systems," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2011, pp. 193–202.

[68] C. Yohann and G. Candea, "Deprogramming large software systems," in *Proc. Workshop Hot Topics Syst. Dependability (HotDep)*, Dec. 2008, paper DSLAB-CONF-2008-005.

ZAIGHAM MUSHTAQ received the M.S. degree in computer science from the COMSATS Institute of Information Technology, Lahore, in 2010, where he is currently pursuing the Ph.D. degree in computer science. He was involved in software process improvement and semantic Web-based SQL statements. He is involved in design recovery of multilingual applications through recognition of J2EE Pattern. His active research areas include source code analysis especially cross language dependence analysis, program comprehension, and source code documentation.

GHULAM RASOOL received the M.Sc. degree in computer science from BZU, Multan, Pakistan, in 1998, the M.S.C.S. degree from the University of Lahore, Pakistan, in 2008, and the Ph.D. degree in reverse engineering from the Technical University of Ilmenau, Germany, in 2011. In 2006, he joined the University of Lahore. He has teaching and research experience of 15 years at national and international levels. He is currently an Associate Professor with the COMSATS Institute of Information Technology, Lahore, Pakistan. His research interests include reverse engineering, design pattern recovery, program comprehension, and source code analysis.

BALAWAL SHEHZAD is currently pursuing the M.S. degree with the COMSATS Institute of Information Technology, Lahore. His research interests are source code analysis, program comprehension, and design pattern detection.