IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 September 2022, accepted 29 September 2022, date of publication 4 October 2022, date of current version 11 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3211949

RESEARCH ARTICLE

Evolution of Software Testing Strategies and Trends: Semantic Content Analysis of Software Research Corpus of the Last 40 Years

FATIH GURCAN[1], GONCA GOKCE MENEKSE DALVEREN[2], NERGIZ ERCIL CAGILTAY[2], DUMITRU ROMAN[3], AND AHMET SOYLU[4]

[1]Department of Computer Engineering, Faculty of Engineering, Karadeniz Technical University, 61080 Trabzon, Turkey
[2]Department of Software Engineering, Faculty of Engineering, Atilim University, 06830 Ankara, Turkey
[3]Department of Sustainable Communication Technologies, SINTEF AS, 0373 Oslo, Norway
[4]Department of Computer Science, Faculty of Information Technology and Electrical Engineering, Norwegian University of Science and Technology, 2815 Gjøvik, Norway

Corresponding author: Ahmet Soylu (ahmet.soylu@ntnu.no)

ABSTRACT From the early days of computer systems to the present, software testing has been considered as a crucial process that directly affects the quality and reliability of software-oriented products and services. Accordingly, there is a huge amount of literature regarding the improvement of software testing approaches. However, there are limited reviews that show the whole picture of the software testing studies covering the topics and trends of the field. This study aims to provide a general figure reflecting topics and trends of software testing by analyzing the majority of software testing articles published in the last 40 years. A semi-automated methodology is developed for the analysis of software testing corpus created from core publication sources. The methodology of the study is based on the implementation of probabilistic topic modeling approach to discover hidden semantic patterns in the 14,684 published articles addressing software testing issues between 1980 and 2019. The results revealed 42 topics of the field, highlighting five software development ages, namely specification, detection, generation, evaluation, and prediction. The recent accelerations of the topics also showed a trend toward prediction-based software testing actions. Additionally, a higher trend on the topics concerning ''Security Vulnerability'', ''Open Source'' and ''Mobile Application'' was identified. This study showed that the current trend of software testing is towards prediction-based testing strategies. Therefore, the findings of this study may provide valuable insights for the industry and software communities to be prepared for the possible changes in the software testing procedures using prediction-based approaches.

INDEX TERMS Software testing, topic modeling, trend analysis, test strategies.

I. INTRODUCTION

Today, software is an indispensable component of the majority of systems and integrated into the daily life of the society. With the advancements of technologies, such as open systems and highly automated or networked devices, software systems are becoming very complex [1]. Additionally, several people from different areas of expertise are usually required to be involved in a software project, which also increases its complexity level. Since software is developed by human beings, it is usual that people make mistakes; thus, in every commercial software some errors always occur [2], and as the level of complexity increases, then these error ratios become even higher [3]. Therefore, the errors that occur need to be detected and removed as soon as possible in the development process. In order to improve the quality of the software, various activities are performed under the heading of software testing. This process is an economically and technically vital component for a high-quality software product [2] and an integral part of the software development life cycle intended

The associate editor coordinating the review of this manuscript and approving it for publication was Mahmoud Elish.

to produce more reliable and higher quality software products [4]. For systems in which there is zero tolerance of error, such as in medical systems and space missions that are directly related with human safety, as well as banking systems, the importance of ensuring a higher quality software development process becomes even more critical.

In the literature, there are a very high volume of studies that have been conducted on software testing from different perspectives. However, only a few early systematic review studies have analyzed research studies showing the trends, developmental stages and topics related to software testing. This information is extremely critical in creating a big picture of the software testing studies, which can guide decision makers, practitioners and educators in the field of software testing to improve their current systems, and thus significantly improve the quality of the software product [5]. These earlier systematic review studies were also conducted with a limited perspective. Currently, there is no study that aimed to analyze all major research articles conducted in the domain of software testing. Accordingly, this study aims to fill this gap by analyzing the articles addressing software testing to create a big picture of the domain.Considering this background, the methodology of the study was designed to investigate the following research questions (RQ):

RQ 1: What are the bibliometric characteristics of software testing studies?

RQ 2: What are the software testing strategies and themes?

RQ 3: How do the trends of software testing strategies and themes change over time?

II. BACKGROUND OF THE STUDY

Testing is defined as ''an activity in which a system is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system'' (ISO/IEC 24765, 2006) [6]. In parallel to this definition of testing, a major task of the software development process, software testing is defined as the process of observing and demonstrating the behavior of a software system for compliance with its specifications [7]. As it requires several strategies and techniques with the involvement of several tools and resources, software testing is also considered as a complex task [8]. The background for this study is given below, summarizing the important role of software testing in the software development life cycle, potential impact of software testing strategies, and review studies conducted on software testing.

A. IMPORTANCE OF SOFTWARE TESTING IN SOFTWARE DEVELOPMENT

Software testing covers several activities of the software development processes starting from the validation of initial requirements through to the acceptance of the end product by the customer [9]. Starting from the requirement specifications, the software testing tasks need to be planned and implemented in different stages of the software development process. Furthermore, software testing needs to be performed during different stages of the software development process for different purposes, such as the testing of the software product lines [10] and the graphical user interface [11].

Software testing is usually conducted in the three stages of creating, executing and evaluating the test cases [12], [13]; thus, the creation of appropriate test cases is critical [14], [15]. In other words, the appropriateness of test cases with software features, such as the technology used, the domain in which the software will be used, and the end-user skills is a critical factor in a successful testing process. Matalonga et al. defined the following seven elements to compose a test case: item (product/functionality under test), input (input variables that will stimulate the test item, output (response returned by the test item after receiving a test input), oracle (expected result, predicted behavior under specified conditions based on its specification or another source), result (comparison between the test output and the test oracle), environment (facilities, hardware, software, firmware, procedures, and documentation intended for or used to perform the software testing), and script (procedure specification for manual or automated testing) [16].

An analysis of the whole software development process reveals that the testing stage has the longest duration and is the most expensive phase [17] involving labor-intensive tasks [18]. As the software testing process is usually performed with limited resources under time constraints, currently, several research studies are being conducted to improve software testing techniques in order to obtain higher-quality and more reliable software products [19].

B. POTENTIAL IMPACT OF SOFTWARE TESTING STRATEGIES

Different software technologies require various testing methodologies and strategies. For instance, testing approaches on context-aware software systems [16], semantic web-enabled software testing [19], testing embedded software systems [20], mobile systems [21], or testing in service oriented architectures [22], [23] may require different strategies. Accordingly, several research studies have been conducted to improve the software testing methods and approaches specific to the technologies being used.

There is also a need for a better estimation of testing effort which may be related to the software technology and is important for completing its processes appropriately. For instance, as a result of a systematic literature review study, Kaur and Kaur reported that it was possible to improve the existing testing effort estimation techniques of mobile applications by weighting the specific characteristics and considering suggestions from experienced developers and testers [24].

Other studies and strategies are also needed for improving the testing process itself. For instance, test case prioritization approaches in regression testing [25], [26], designing the software testing processes [27], improving the regression testing costs [28] and using genetic algorithms compared to pure random testing [1], [2]. Deciding on the appropriate testing strategy for the testing process is another challenge [29].

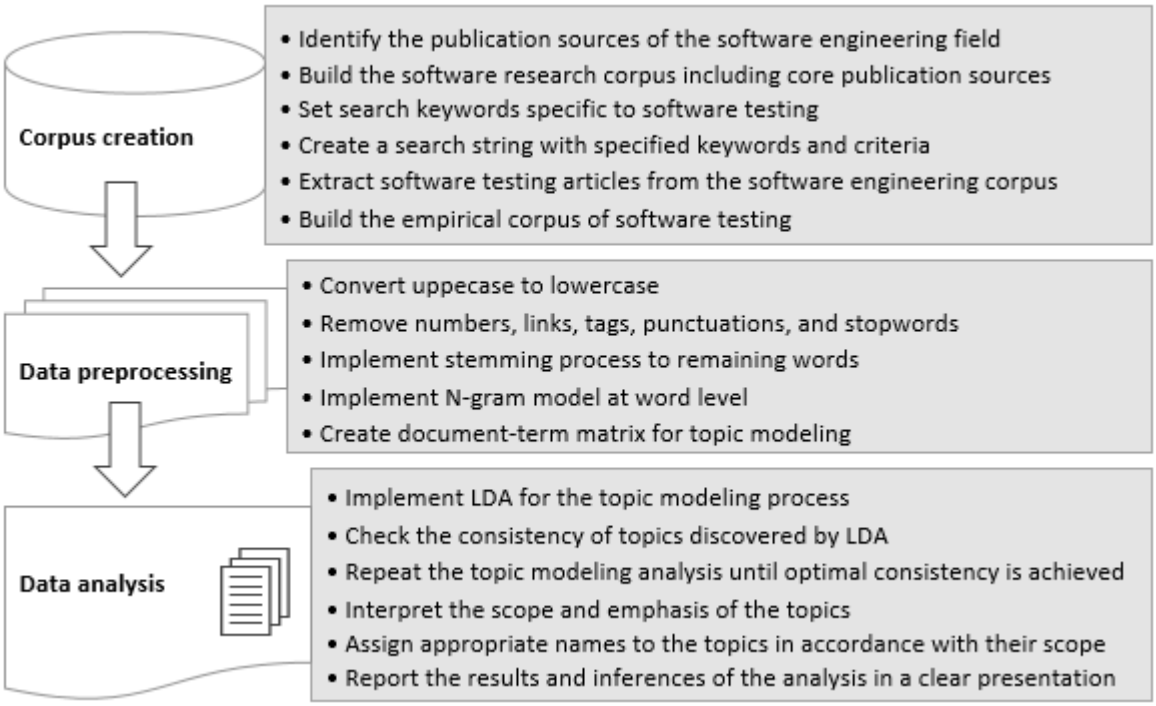

FIGURE 1. An overview of the methodology of the study.

To summarize, some heterogeneity and ambiguity exist among the different concepts dealing with testing methods and processes; therefore, Tebes et al. analyzed software testing ontologies to conceptualize software testing concepts, concluding that there was a lack of addressing non-functional software requirements and static testing terminological coverage where none of the ontologies directly linked functional and non-functional software requirements [30]. Accordingly, understanding the trends and development stages of software testing is critical for the development of conceptual models for software testing methods and processes. The literature on software testing provides a large number of studies, regarding both general and specific issues; however, among these studies, there are only a few reviews evaluating the trends and topics related to software testing, which are summarized in the next section.

C. REVIEW STUDIES ON SOFTWARE TESTING

Barmi et al. conducted a systematic review to better understand the connections between the specifications and testing requirements and reported that ''Model-based testing'' was the most commonly studied topic (26%), followed by ''Formal Approaches'' (24%), and ''Traceability'' (18%) and concluded that there was a significant gap between the specification and testing requirements [31]. Their study is considering the relationship between the specifications and testing requirements, not the whole process of software testing. In this context, Garousi and Mäntylä reported that over 101 secondary research studies (as a study of studies) had been published in the area of software testing since 1994, with model-based software testing being the most popular method, web-services the most popular system, and regression testing the most popular testing phase [32]. Since this was a ternary study, it has limitations in showing the whole picture of the software testing studies. Zein et al. performed a systematic mapping study in order to reveal testing techniques for mobile application and mapped 79 empirical studies to a taxonomy [33].

There are also several topic modeling studies conducted in the field of software engineering [34], [35], [36]. However, to the best of the authors' knowledge, there is no topic modeling study conducted on the software testing area by using text-mining analysis and considering whole software testing processes though its literature from its early years to today. Using the methodology described below, this study aims to fill this gap of the literature and provide a larger picture of the software testing field.

III. RESEARCH METHODOLOGY

In this study, a semi-automated methodology was developed in order to analyze the empirical corpus consisting of the software testing articles. The methodology of the study was based on the implementation of Latent Dirichlet Allocation (LDA) [37], a probabilistic topic modeling algorithm used to discover hidden semantic patterns on the software testing corpus created in two consecutive stages.

In this context, the research methodology designed in accordance with the purpose of the study consisted of the

following stages (see Figure 1). Initially, the experimental corpus of this study was prepared. Afterwards, the data preprocessing was applied to the corpus, which was followed by LDA implementation. Finally, interpretation and visualization procedures were conducted. This methodology is described in detail below.

A. CREATION OF SOFTWARE TESTING CORPUS

Testing is a comprehensive concept related to the development of each system. In the software engineering discipline, testing is a crucial task of the software development life cycle. In contrast, software testing in any field other than software engineering can be considered as an end-user testing focused on the suitability of a software developed for a specific purpose in this field. For this reason, the multidisciplinary use of software testing makes it difficult to create a specific corpus of software testing studies in the scope of software engineering. In this context, to create a specific corpus of software testing within the scope of software engineering, a methodology including two sequential stages was followed for corpus creation, which included identifying core publication sources for the software engineering field and extracting articles specific to software testing.

From this perspective, firstly, publication sources (core conferences and journals) within the scope of the software engineering field were tried to be identified. As a result, the 44 core publication sources (28 conferences and 16 journals) specific to the software engineering field, which we identified in our previous study [38], were used as the data source for the creation of the software research corpus of this study.

After creating the software research corpus, the process of obtaining articles specific to software testing was carried out on this corpus. To extract the articles in this context, firstly, the keywords related to software testing were selected using an iterative process and keywords related to software testing including ''test*'', ''fault*'', ''bug*'', ''debug*'', and ''defect*'' were identified.

During this iterative process, we initially searched for articles with the term ''test*'' in the software research corpus and filtered them. Then, we examined the keywords in the filtered articles and found that the term ''fault*'' is frequently seen in them. Therefore, we added the term ''fault*'' as a second keyword to the search string. Then we searched for articles with the terms ''test*'' or ''fault*''. We examined the keywords in these articles, and this time we added the frequently seen word ''bug*'' to the search string. We re-examined the keywords that appear frequently in these articles, and this time we added ''bug*'', an another high-frequency term, to the search string. Finally, we added the terms ''debug*'' and ''defect*'' to the search string, repeating these sequential steps each time.

Then, the articles containing these five keywords (''test*'', ''fault*'', ''bug*'', ''debug*'', ''defect*'') in the title, abstract, and author keywords were searched. In conclusion, the search string was finalized by adding the time period (1980-2019) and language (only English) criteria of the articles. As a result, the final version of the search query was created as follows:

((EXACTSRCTITLE (''Empirical Software Engineering'' OR ''Information and Software Technology'' OR ''Journal of Systems and Software'' OR ''IEEE Transactions on Software Engineering'' OR. . . . . ''*other publication sources in Table 2*'') AND (PUBYEAR < 2020) AND (PUBYEAR > 1979)) AND (TITLE-ABS (test* OR fault* OR bug* OR debug* OR defect*) OR AUTHKEY (test* OR fault* OR bug* OR debug* OR defect*)) AND (LIMIT-TO (DOCTYPE, ''cp'') OR LIMIT-TO (DOCTYPE, ''ar'') OR LIMIT-TO (DOCTYPE, ''re'')) AND (LIMIT-TO (LANGUAGE, ''English''))

This search string created with these criteria was employed on SCOPUS, a bibliometric database that indexes all journals and conferences selected for the corpus [38], [39], [40]. As a result of this search carried out on July 28, 2020, a software testing corpus was created containing 14,684 articles (9,205 conference proceedings, 5,349 research articles, and 130 review articles) published in English over the last 40 years. This empirical corpus contains only the title, abstract, and author keywords of each article because these sections best describe the characteristics of an article, such as purpose, method, conclusion, and scope [35]. The distribution of the numbers of these articles in the corpus by publication sources and years is given in the results section.

B. DATA PREPROCESSING

Data preprocessing is an important task and critical step for the success of analysis based on text mining and natural language processing [41]. It renovates textual data into a form that can be predicted and analyzed more effectively so that machine learning algorithms can perform better [42]. In this regard, with the aim of preparing the software testing corpus for probabilistic topic modeling, a series of necessary textual data processing steps were respectively implemented on the corpus. As a first step, the word tokenization procedure was performed on texts in the corpus to separate the texts into single tokens (words). This was followed by the process of converting all text to lowercase. Subsequently, publication source titles, links, misleading words, special characters, and punctuations were removed. The stop words (is, and, a, an, the, of, for, etc.), which have a high frequency in English and do not make any sense alone, were also deleted [39]. The Snowball stemming algorithm [43] was applied to the remaining words to combine different variations of the words derived from the same root into a single root form. Moreover, with the intention of investigating the word phrases having high frequency in the software testing corpus, the N-gram based text categorization approach at word level was performed on the texts, and thus high-frequency phrases were identified as unigrams, bigrams, and trigrams [41]. Subsequently, each article in the empirical corpus was demonstrated as a word vector making available the numerical representation of the texts in the corpus. To conclude, a document-term matrix, which is the numerical

matrix form necessary for topic modeling implementation, was created by combining these vectors [44].

C. LDA IMPLEMENTATION

Topic modeling is an approach that provides for semantic analysis and understanding of the themes in large collections that contain unstructured textual content [37], [44]. In this way, it offers perspectives for the analysis, modeling, understanding, and summarizing of huge collections, which include a large number of text documents. LDA is one of the widely used topic modeling algorithms and an unsupervised method for probabilistic topic modeling to discover groups of words called ''topics'' in a text document [37], [44]. In the LDA model, each document is assumed to consist of a collection of topics and each word in the document corresponds to one of these topics. These topics can be defined as a set of words that are frequently used together and often reveal a common theme. The topics discovered by LDA, represented by predefined word sets, are considered as a tool to best describe the entire document semantically [37], [44], [45]. In this study, the fitting and implementation of the LDA [37] topic modeling technique with Gibbs sampling [46] to the empirical corpus of software testing was achieved using the tmtoolkit package [47], an effective toolkit developed in Python that includes a wide spectrum of tools for text mining and topic modeling approaches. In order to fit the LDA model to the software testing corpus, the values of the prior parameters (α, β and K) that provide the optimization of the model were used with $\alpha = 0.1$ and $\beta = 0.01$, which are the values suggested for the topic modeling of short texts in previous studies [39], [42]. Subsequently, the LDA model was implemented on the corpus for different values between 15 and 75 of parameter K, which indicates the number of topics. With the intention of empirically identifying the optimal number of topics, coherence measure C_V was calculated for each topic number from 1 to 75 using semantic coherence model [48]. Maximum coherence score was achieved with a topic number of $K = 42$. As a consequence, these 42 topics discovered in line with the coherence measure were used in all subsequent analysis.

D. INTERPRETATION AND VISUALIZATION

The scope and consistency of the 42 topics and their temporal trends discovered by LDA were evaluated and interpreted at this stage, taking into account the background and dimensions of software testing, which was the context of the study. Each of these 42 topics contained top 15 descriptive keywords reflecting the characterization of the topics. Taking into consideration the first five of these keywords with the highest frequency, the topic labelling process was performed manually for each topic [35], [39]. Furthermore, the distribution percentage of each topic per document and the distribution percentage of the topics in the entire corpus were calculated, and the annual changes of these percentages were interpreted and visualized for each topic, and a taxonomy that reflects the evolution of software testing from past to present from a panoramic perspective was proposed.

IV. RESULTS

First, the results of the study are given descriptively to provide the general figure, followed by the topic modeling analysis, and temporal analysis.

A. DESCRIPTIVE ANALYSIS (RQ1)

Table 1 shows a total of 14,684 articles related to software testing that were analyzed in this study. The volume of articles published in each five-year period can be seen to continually increase.

TABLE 1. Distribution of articles in five-year periods.

Periods	n	%
1980-1984	204	1.39
1985-1989	357	2.43
1990-1994	703	4.79
1995-1999	940	6.40
2000-2004	1,294	8.81
2005-2009	2,510	17.09
2010-2014	3,689	25.12
2015-2019	4,987	33.96
Total	**14,684**	**100**

Table 2 reveals the publication sources [38], their type as conference (C) or journal (J), number of articles selected from these sources (N), their percentages to the total number of articles considered in the corpus of software testing articles.

The data created using the procedures described in the research methodology section was analyzed first to understand the keywords' unigram, bigram and trigram distributions. As seen from Table 3, the keyword ''test'' had the highest unigram ratio for all studied articles (67.64%) whereas ''software develop'' had the highest bigram ratio (12.12%) and ''open source project'' had the highest trigram ratio (2.65%).

B. TOPIC MODELING ANALYSIS (RQ2)

Implementing the LDA-based topic modeling analysis, 42 topics describing the software testing strategies were found. The top 15 keywords of each topic and their ratio in the corpus are given in Table 4. The topic names are given by considering the first four keywords classified under each topic. The topics in Table 4 also illustrate software testing strategies, so the terms ''topic'' and ''software testing strategies'' are used interchangeably throughout this paper. These topics are listed in Table 4 according to their ratio among all corpus, with ''Test Generation'' having the highest ratio (5.85%) considering the number of articles published under this topic, and the lowest ratio (1.21%) belonged to ''Security Vulnerability''.

C. TEMPORAL TRENDS OF THE TOPICS (RQ3)

In order to better understand the temporal trends of the discovered topics and their temporal developmental ages, the

TABLE 2. Publication sources included in the corpus.

Type	Publication Name	N	%
C	ACM/IEEE International Conference on Software Engineering	1,734	11.81
C	IEEE International Conference on Software Testing, Verification and Validation (ICST)	1,288	8.77
C	International Symposium on Software Reliability Engineering (ISSRE)	1,075	7.32
J	IEEE Transactions on Software Engineering	1,046	7.12
J	Journal of Systems and Software	1,020	6.95
J	Information and Software Technology	763	5.20
C	IEEE/ACM International Conference on Automated Software Engineering (ASE)	719	4.90
C	International Symposium on Software Testing and Analysis	558	3.80
J	Software Practice and Experience	532	3.62
C	ACM SIGSOFT International Symposium on Foundations of Software Engineering	480	3.27
J	IEEE Software	419	2.85
J	Empirical Software Engineering	347	2.36
J	Software Testing Verification And Reliability	323	2.20
C	ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)	285	1.94
C	ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Lang… (OOPSLA)	284	1.93
J	Software Quality Journal	273	1.86
C	International Symposium on Empirical Software Engineering and Measurement, ESEM	263	1.79
J	Science of Computer Programming	248	1.69
C	International Colloquium on Automata, Languages and Programming (ICALP)	219	1.49
J	International Journal on Software Tools for Technology Transfer	208	1.42
C	Mining Software Repositories	207	1.41
C	IEEE International Conference on Software Maintenance and Evolution	176	1.20
J	ACM Transactions on Software Engineering and Methodology (TOSEM)	169	1.15
C	IEEE Annual Computer Software and Applications Conference (COMPSAC)	158	1.08
C	IEEE International Requirements Engineering Conference	145	0.99
C	International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)	138	0.94
C	ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)	134	0.91
C	European Conference on Object-oriented Programming (ECOOP)	129	0.88
J	Journal of Software Evolution and Process	120	0.82
C	ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)	117	0.80
C	International Conference on Principles and Practice of Constraint Programming	114	0.78
J	International Journal of Parallel Programming	112	0.76
C	IEEE International Conference on Web Services	112	0.76
C	International Conference on Software Analysis, Evolution, and Reengineering (SANER)	112	0.76
C	IEEE International Conference on Program Comprehension	112	0.76
J	Software and Systems Modeling	101	0.69
C	International Conference on Evaluation and Assessment in Software Engineering	90	0.61
C	ACM SIGPLAN International Conference on Functional Programming (ICFP)	76	0.52
C	International Conference on Agile Processes in Software Engineering and Extreme Programming (XP)	66	0.45
J	Theory and Practice of Logic Programming	65	0.44
C	IEEE International Symposium on Performance Analysis of Systems and Software	45	0.31
C	European Symposium on Programming (ESOP)	40	0.27
J	Requirements Engineering	39	0.27
C	International Symposium on Software Engineering for Adaptive and Self-Managing Systems	23	0.16
	Total	14,684	100

percentage of each topic was analyzed in five-year periods. As a result, the percentage of the topics in the corpus (C%), and percentage of the topics in the same yearly period (Y%) are given in the table in Appendix-A. In addition, the average acceleration (AC) value for each year was calculated by subtracting the Y% of the previous year (percentage of the topics in the same year period) from that of the current year. Considering these yearly AC values, the five-year average acceleration values for each topic were then calculated and presented in Appendix-A. Furthermore, the overall AC values of each topic over the last 40 years are given in the last column of the table in Appendix A.

Considering these overall AC values (see Appendix A – AVG), we identified whether the acceleration values of the topics were positive (increasing) or negative (decreasing) from 1980 to 2019. Following, we illustrated the top ten topics with positive AC values in Figure 2 and the top ten topics with negative AC values in Figure 3. As seen in Figure 2, the topic ''Prediction'' had the highest acceleration (0.11), followed by ''Empirical Evaluation'' (0.10), ''Source Code'' (0.09), and ''Bug Reporting'' (0.09). On the other hand, Figure 3 revealed that ''Programming Tools'' (−0.21), ''Language Specification'' (−0.19), ''Graph Algorithms'' (−0.18) and ''Database'' (−0.18) were the top topics with negative AC values.

In order to visualize our findings given in Appendix-A and to provide a better understanding of the temporal changes in the trends of the topics, we presented acceleration graphs of the top ten topics with positive accelerations (see Figure 4) and the top ten topics with negative accelerations (see Figure 5). In Figures 4 and 5, the blue lines show the acceleration (AC) values calculated for each five-year period for

TABLE 3. Top 25 unigrams, bigrams, and trigrams in the corpus.

Unigram	%	Bigram	%	Trigram	%
test	67.64	softwar develop	12.12	open sourc project	2.65
softwar	57.67	softwar test	11.56	open sourc softwar	1.66
develop	44.37	softwar engin	10.46	softwar develop process	1.65
system	43.39	softwar system	9.20	automat generat test	1.41
use	41.07	open sourc	8.85	test data generat	1.37
approach	38.82	test generat	8.26	autom test generat	1.30
program	36.32	test suit	7.77	mine softwar repositori	1.11
techniqu	33.74	generat test	7.03	empir softwar engin	0.97
model	33.30	sourc code	6.83	automat test generat	0.95
base	32.04	autom test	5.07	graphic user interfac	0.93
analysi	30.33	softwar qualiti	4.75	softwar develop project	0.88
applic	29.03	test softwar	4.50	softwar product line	0.86
tool	28.37	system test	4.43	reliabl growth model	0.79
code	27.90	static analysi	4.13	generat test suit	0.78
provid	27.39	program languag	3.95	generat test data	0.76
effect	27.38	automat generat	3.85	softwar reliabl model	0.75
evalu	26.91	test techniqu	3.82	softwar qualiti assur	0.72
perform	26.91	test data	3.81	static analysi tool	0.72
method	26.14	regress test	3.47	improv softwar qualiti	0.69
data	25.88	develop process	3.41	softwar reliabl growth	0.69
implement	25.68	softwar reliabl	3.39	generat test input	0.67
generat	25.45	softwar project	3.21	softwar defect predict	0.67
process	25.31	fault detect	3.20	test generat techniqu	0.66
requir	24.20	test tool	3.06	fault local techniqu	0.65
execut	23.41	test execut	3.05	test generat tool	0.63

that topic and the red line shows the linear trend-line that enables predictions of the near-future trend of that topic. Taking into account Figures 4 and 5, a number of implications can be drawn as to which software testing strategies will dominate and which will withdraw in the near-future. Temporal changes in volume and acceleration of other topics can be seen in Appendix-A.

D. THE LATEST TRENDS IN THE TOPICS (RQ3)

Due to the rapid paradigmatic transformations in software technologies, we specifically analyzed the trends in software testing strategies in the last 5 years from 2015 to 2019. In this way, the recent acceleration values of the topics during the last five-year period were also calculated and presented in Figures 6 and 7. Specifically, Figure 6 shows the top ten topics with positive acceleration values from 2015 to 2019. On the other hand, the top ten topics with negative acceleration values from 2015 to 2019 are given in Figure 7. As seen in Figure 6, interestingly, the topic ''Prediction'' had a significantly higher recent acceleration (0.75) compared to the other topics. Here, it should be noted that even the average volume of the topic ''Security Vulnerability'' was the lowest (see Appendix-A, 0.15) while its recent acceleration was one of the highest (see Appendix-A, 2015-2019, 0.30). A similar trend was observed for the topics ''Open Source'' and ''Mobile Applications'' where their average volume was lower compared to the other topics (see Appendix-A, 0.20 and 0.16 respectively) but their recent accelerations were the highest (see Appendix-A, 2015-2019, 0.29 and 0.24, respectively). On the other hand, as emphasized in Figure 7, the topics ''Web Applications'' (−0.23), ''Fault Detection'' (−0.22), and ''Project Management'' (−0.22) had the lowest recent accelerations.

E. DEVELOPMENTAL AGES OF SOFTWARE TESTING (RQ3)

With the aim of providing a better understanding of the developmental ages of software testing strategies over the last 40 years from 1980 to 2019, the top ten topics of each five-year period were identified and presented in Table 5. The newly included ones in the top ten topics in each period are highlighted in bold (see Table 5). In order to more clearly demonstrate the changes in software testing over the timeline of the last 40 years and to define its developmental ages, we visualized the top five topics in each five-year period and presented Figure 8. As shown in Figure 8, from 1985 to 1995, topics such as ''Model Reliability'', ''Programming Tools'' and ''Fault Detection'' were in the top five list of all topics. The testing processes during this period can be considered as more programming environment-oriented and fault detection-based. Accordingly, this period was labelled as ''Detection Age'', which can be considered as programming-oriented. After 1995, topics such as ''Empirical Evaluation'' and ''Test Generation'' became dominant; thus, the period from 1995 to 2005 was referred to as the ''Generation Age'' of software testing. ''Testing Practices'' then became one of the dominating topics, with the period from 2005 to 2015 being called the ''Evaluation Age''. Interestingly, after 2015, the topic ''Prediction'' became one of the dominating topics, indicating a change in the field and was named the ''Prediction Age''.

TABLE 4. Discovered topics and top keywords by LDA.

Topic Name	Keywords	%
Test Generation	test*; generat*; autom*; tool*; automat*; unit*; techniqu*; manual*; execut*; function*; framework*; suit*; tester*; applic*; experi*	5.85
Empirical Evaluation	empir*; perform*; experi*; effect*; investig*; impact*; evalu*; factor*; signific*; compar*; analysi*; understand*; find*; conduct*; suggest*	5.00
Testing Practices	industri*; practic*; challeng*; test*; discuss*; techniqu*; tool*; address*; applic*; identifi*; field*; current*; solut*; futur*; issu*	4.61
Project Management	process*; project*; qualiti*; manag*; product*; improv*; cost*; plan*; effort*; risk*; organ*; mainten*; activ*; busi*; assur*	3.40
Program Analysis	analysi*; program*; dynam*; static*; depend*; slice*; techniqu*; flow*; inform*; analys*; graph*; execut*; comput*; call*; precis*	3.12
Component Design	design*; compon*; architectur*; framework*; integr*; applic*; implement*; reus*; pattern*; support*; structur*; level*; object*; aspect*; interfac*	3.03
Model Reliability	model*; reliabl*; estim*; statist*; data*; test*; profil*; predict*; time*; failur*; probabl*; growth*; analysi*; process*; distribut*	2.95
Programming Tools	program*; tool*; debug*; user*; environ*; visual*; interact*; programm*; support*; debugg*; interfac*; execut*; spreadsheet*; task*; describ*	2.88
Bug Detection	detect*; tool*; bug*; error*; code*; pattern*; analysi*; static*; fals*; api*; mine*; posit*; automat*; rule*; sourc	2.79
Source Code	code*; chang*; sourc*; refactor*; clone*; evolut*; file*; smell*; mainten*; identifi*; open*; project*; impact*; histori*; evolv*	2.66
Parallel Computing	perform*; memori*; parallel*; comput*; applic*; virtual*; time*; load*; execut*; machin*; benchmark*; overhead*; effici*; processor*; alloc*	2.57
Regression Testing	test*; regress*; techniqu*; suit*; select*; priorit*; reduct*; reduc*; cost*; effect*; detect*; fault*; execut*; time*; evalu*	2.57
Fault Detection	fault*; failur*; detect*; error*; toler*; caus*; inject*; techniqu*; reliabl*; effect*; analysi*; depend*; redund*; recoveri*; tree*	2.54
Programming Languages	program*; type*; java*; languag*; compil*; code*; object*; implement*; class*; c*; object-ori*; function*; contract*; optim*; support*	2.50
Verification	verif*; check*; properti*; formal*; model*; specif*; verifi*; safeti*; correct*; valid*; invari*; techniqu*; assert*; tempor*; proof*	2.48
Algorithm Optimization	algorithm*; search*; optim*; configur*; combinatori*; search-bas*; space*; set*; solut*; genet*; paramet*; generat*; interact*; combin*; heurist*	2.45
Requirements	requir*; user*; knowledg*; inform*; usabl*; specif*; document*; process*; analysi*; support*; evalu*; traceabl*; link*; artifact*; valid*	2.43
Bug Reporting	bug*; report*; fix*; inform*; issu*; sourc*; retriev*; repositori*; task*; duplic*; file*; project*; assign*; topic*; track	2.43
Prediction	predict*; learn*; model*; defect*; data*; classif*; perform*; machin*; featur*; project*; classifi*; dataset*; train*; techniqu*; accuraci*	2.34
Model Transformation	model*; transform*; model-bas*; uml*; diagram*; tool*; generat*; net*; support*; languag*; model-driven*; valid*; simul*; design*; process*	2.32
Language Specification	languag*; specif*; semant*; formal*; express*; generat*; implement*; program*; descript*; describ*; grammar*; rule*; specifi*; natur*; string*	2.31
Constraint Programming	constraint*; program*; logic*; solv*; theori*; express*; variabl*; solver*; set*; predic*; condit*; comput*; satisfi*; reason*; formula*	2.30
Distributed Networks	distribut*; network*; protocol*; communic*; implement*; messag*; process*; fault-toler*; node*; comput*; applic*; agent*; checkpoint*; mechan*; recoveri*	2.28
Embedded Control	control*; embed*; real-tim*; time*; simul*; environ*; adapt*; schedul*; requir*; resourc*; automot*; behavior*; monitor*; oper*; task*	2.06
Program Execution	execut*; input*; program*; symbol*; generat*; path*; techniqu*; loop*; explor*; synthesi*; fuzz*; automat*; complex*; dynam*; acm*	2.02
Fault Localization	local*; fault*; program*; repair*; techniqu*; debug*; patch*; locat*; cluster*; effect*; fail*; correct*; autom*; statement*; faulti*	2.01
Test Strategies	test*; strategi*; random*; oracl*; input*; effect*; sampl*; partit*; output*; select*; metamorph*; domain*; adapt*; program*; detect*	2.00
Metrics	metric*; measur*; qualiti*; modul*; class*; complex*; predict*; fault-pron*; testabl*; correl*; attribut*; coupl*; size*; valid*; empir*	1.97
Test Coverage	test*; coverag*; criteria*; branch*; program*; suit*; criterion*; effect*; structur*; code*; set*; cover*; adequaci*; measur*; achiev*	1.93
Test Sequence	test*; sequenc*; specif*; behavior*; machin*; transit*; conform*; finit*; implement*; generat*; model*; automata*; behaviour*; time*; formal*	1.92
Agile Practices	agil*; team*; student*; practic*; project*; test-driven*; program*; game*; tdd*; test*; experi*; qualiti*; learn*; cours*; survey*	1.75
Concurrent Programming	concurr*; program*; race*; thread*; execut*; synchron*; bug*; atom*; memori*; parallel*; detect*; multithread*; sequenti*; schedul*; interleav*	1.72

TABLE 4. *(Continued.)* **Discovered topics and top keywords by LDA.**

Web Applications	web*; applic*; servic*; javascript*; user*; server*; composit*; page*; xml*; dynam*; browser*; client*; qos*; busi*; service-ori*	1.71
Graph Algorithms	algorithm*; graph*; tree*; bound*; time*; set*; comput*; test*; function*; approxim*; polynomi*; block*; effici*; complex*; size*	1.70
Product Line Inspection	defect*; product*; inspect*; line*; detect*; qualiti*; featur*; techniqu*; effect*; process*; variabl*; read*; experi*; improv*; code*	1.68
Database	data*; databas*; structur*; queri*; applic*; collect*; inform*; process*; scheme*; set*; sql*; integr*; store*; schema*; consist*	1.67
Open Source	project*; sourc*; build*; open*; continu*; releas*; integr*; commit*; repositori*; code*; ci*; time*; communiti*; github*; mine*	1.58
Event Tracing	event*; trace*; gui*; log*; execut*; user*; crash*; applic*; replay*; record*; reproduc*; interact*; interfac*; imag*; sequenc*	1.47
Mutation Testing	mutat*; test*; mutant*; oper*; program*; analysi*; equival*; suit*; fault*; effect*; set*; evalu*; generat*; kill*; techniqu*	1.33
Mobile Application	app*; mobil*; android*; applic*; cloud*; except*; devic*; user*; platform*; energi*; handl*; consumpt*; comput*; resourc*; power*	1.25
Version Update	librari*; version*; updat*; oper*; kernel*; function*; robust*; increment*; linux*; packag*; conflict*; driver*; migrat*; api*; depend*	1.22
Security Vulnerability	secur*; vulner*; attack*; polici*; access*; detect*; smart*; certif*; contract*; inject*; exploit*; implement*; protect*; applic*; inform*	1.21

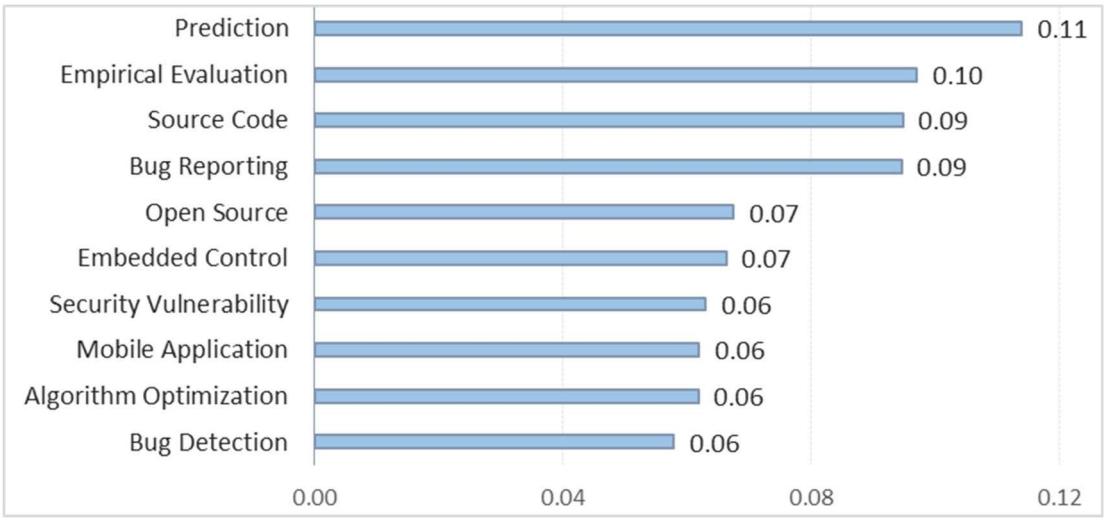

FIGURE 2. Top ten topics with positive acceleration values.

V. DISCUSSION

This study analyzed the last 40 years of the software testing studies and provided several contributions to the software engineering field. These contributions are summarized below under six headings, namely systematic methodology for corpus-based topic modeling, the wide spectrum of software testing topics, insights into the methods and strategies, developmental ages of software testing, and the future outlook for software testing. Finally, the limitations and suggestions of the study are presented.

A. SYSTEMATIC METHODOLOGY FOR CORPUS-BASED TOPIC MODELING

The first contribution of this study is the proposed two-stage corpus creation methodology, which was used due to the challenges in creating an appropriate search term for selecting articles related to the software testing domain. This two-stage corpus creation approach has not been reported in earlier studies, and accordingly this is a significant contribution to the corpus creation method of mapping studies. For certain specific domains such as software testing, the proposed methodology improves the existing approaches. As the corpus is very important for mapping studies, our corpus creation methodology is expected to improve future studies significantly.

B. INSIGHTS INTO THE METHODS AND STRATEGIES

In the software testing stages, the aim is to develop software-oriented products and services in a systematic and efficient manner, in which a wide range of tasks, methods, and strategies are used. Depending on the type, scope and context of the software designed and developed, the methods and strategies

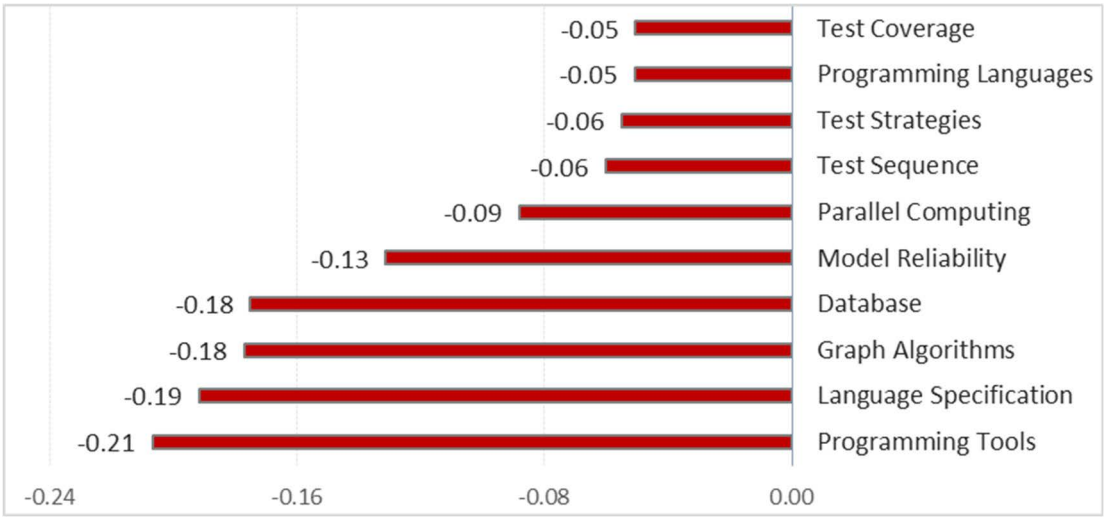

FIGURE 3. Top ten topics with negative acceleration values.

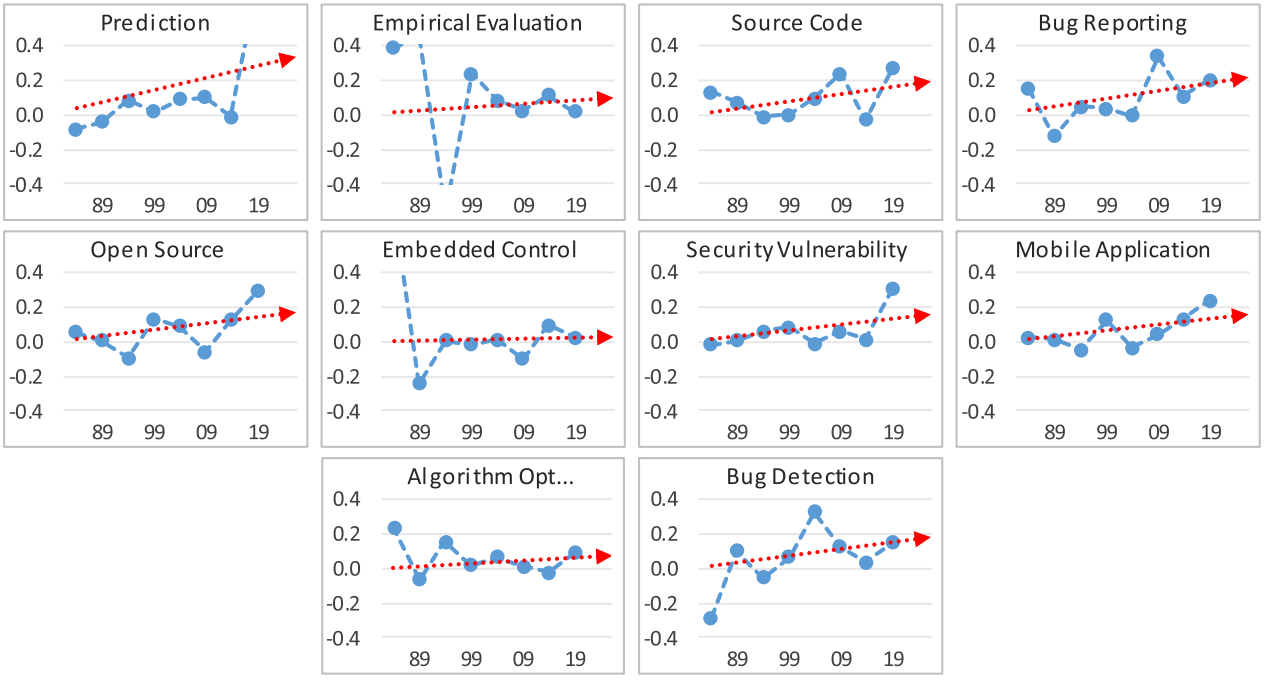

FIGURE 4. The acceleration graphs of the top ten topics with positive trends.

chosen during the software testing stages vary considerably. The findings of this study offer a wide-ranging insight into not only the themes and trends in focus but also the tools, tasks, methods, and strategies specific to software testing. Specifically, the discovered topics reveal that the most focused tasks in software testing are specification, transformation, detection, localization, generation, evaluation, optimization, verification, and prediction. The important background provided by the core tasks highlighted in this study for software testing has also been addressed by previous studies [49]. Likewise, the findings draw clear attention to methods and strategies, such as "Test Generation", "Empirical Evaluation", "Fault Localization", "Regression Testing", "Mutation Testing", "Program Analysis", "Bug Reporting", "Algorithm Optimization", "Event Tracing", and "Product Line Inspection", which are revealed as discrete topics. Among these topics, which also emphasize methods and strategies, attention is drawn to "Test Generation", "Empirical Evaluation", and "Testing Practices" as the top three topics having the highest percentages. Hence, results of these earlier studies and this current study validates each other.

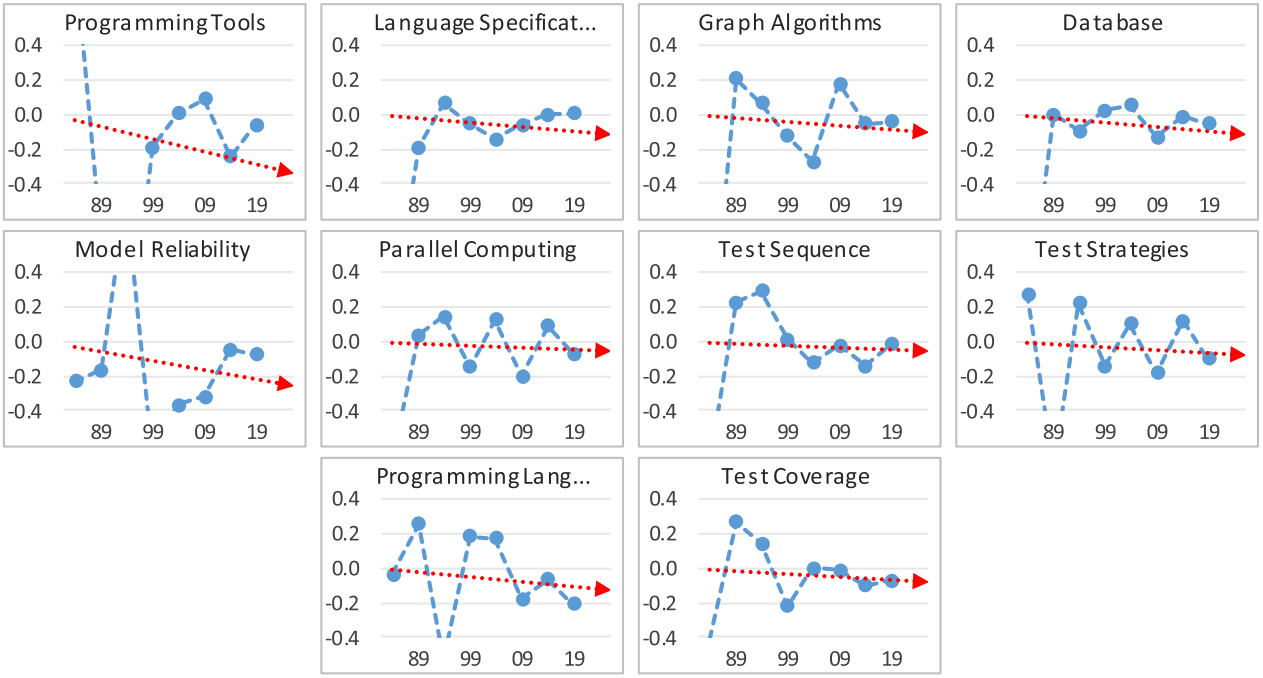

FIGURE 5. The acceleration graphs of the top ten topics with negative trends.

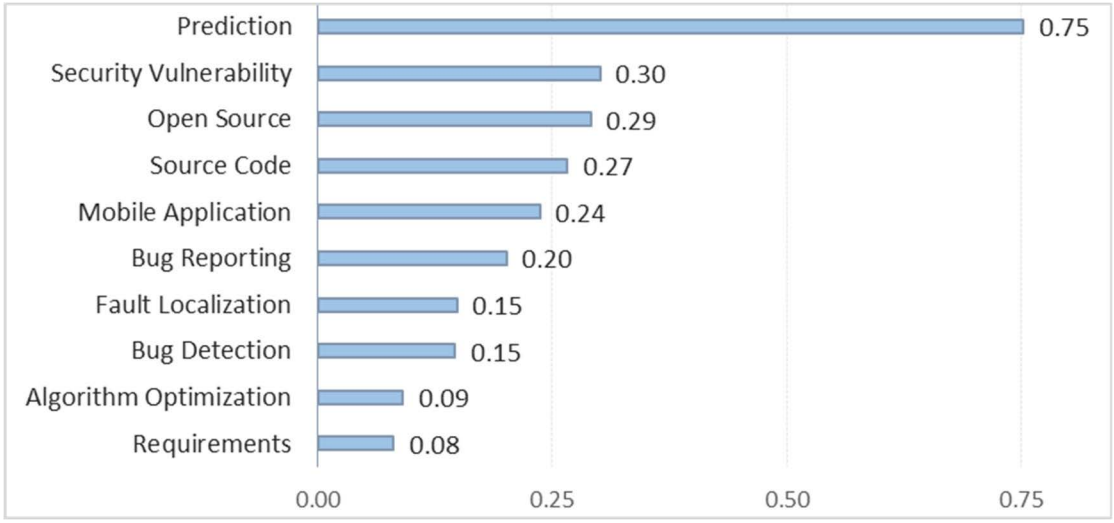

FIGURE 6. Top ten topics with positive acceleration values from 2015 to 2019.

C. FIVE DEVELOPMENTAL AGES FROM SPECIFICATION TO PREDICTION

The results indicate that the formation of the topics in software testing started in 1980, which marks the year when IBM released the first personal computer on the mass market. From this date, since many users started to use software applications, testing became more critical. In the current study, starting from 1980, the developmental periods of software testing were classified under five developmental ages, namely specification, detection, generation, evaluation, and prediction. As indicated by Boehm, during the early 1980s, the testing process was mainly conducted on fixing bugs in the codes [50]. This is also confirmed by our results, and we named the period between 1980 and 1985 as the specification age of software testing. After 1985, defect detection and understanding the distribution of defects as well as building connections between defects and requirements were the concepts that influenced testing [51], [52]. Researchers started to develop methods to classify and mathematically model defects [53]. This situation is also supported by our results and indicates the impact of fault detection on testing from 1985 to1995, which we called the detection age of software

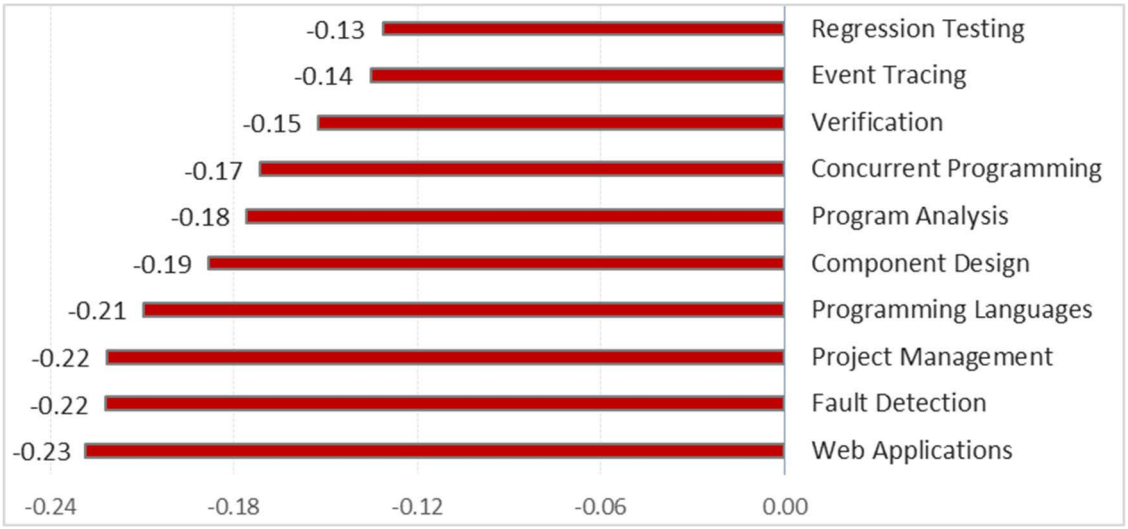

FIGURE 7. Top ten topics with negative acceleration values from 2015 to 2019.

TABLE 5. Discovered topics and top keywords by LDA.

1980-1984	1985-1989	1990-1994	1995-1999
• Programming Tools	• Programming Tools	• Model Reliability	• Model Reliability
• Language Specification	• **Distributed Networks**	• Programming Tools	• Project Management
• Database	• Model Reliability	• Fault Detection	• Distributed Networks
• Model Reliability	• Language Specification	• **Test Generation**	• Test Generation
• Project Management	• Fault Detection	• Distributed Networks	• **Empirical Evaluation**
• Test Generation	• Empirical Evaluation	• **Program Analysis**	• Component Design
• Parallel Computing	• Component Design	• Project Management	• Testing Practices
• Empirical Evaluation	• Project Management	• Language Specification	• Fault Detection
• Program Analysis	• Database	• Component Design	• Program Analysis
• Fault Detection	• **Testing Practices**	• Testing Practices	• Programming Tools
2000-2004	**2005-2009**	**2010-2014**	**2015-2019**
• Test Generation	• Test Generation	• Test Generation	• Test Generation
• Component Design	• Empirical Evaluation	• Empirical Evaluation	• Empirical Evaluation
• Model Reliability	• Testing Practices	• Testing Practices	• Testing Practices
• Empirical Evaluation	• Component Design	• Bug Detection	• **Prediction**
• Project Management	• Project Management	• Project Management	• Bug Reporting
• Testing Practices	• Program Analysis	• Program Analysis	• Source Code
• **Programming Languages**	• Programming Languages	• **Regression Testing**	• Bug Detection
• **Verification**	• **Bug Detection**	• **Source Code**	• **Fault Localization**
• Program Analysis	• Verification	• **Bug Reporting**	• Regression Testing
• Distributed Networks	• Model Reliability	• Algorithm Optimization	• Algorithm Optimization

testing. Starting from 1995, test generation concepts began to be introduced by researchers such as [54], and this was also supported by our results, indicating the generation age between 1995 and 2005. As also reported by van Dam, the steps for software testing were taken after 1998, indicating the beginning of the software development age in the software testing process [55]. However, a systematic implementation on testing procedures started after 1995. Then, after 2008, testing started to be considered as part of the software development processes [55], which may support the current study's findings on the evaluation age. Currently, since time is critical for software projects, automation in software testing procedures was reported as very important, for which prediction is an inevitable part. Hence, studies on software testing indicate a trend toward better defect prediction and removal of bugs, and thus improving software quality [55].

D. FUTURE OUTLOOKS FOR SOFTWARE TESTING

In this study, considering the volume percentages of the topics, even ''Prediction'', a younger topic that started to be

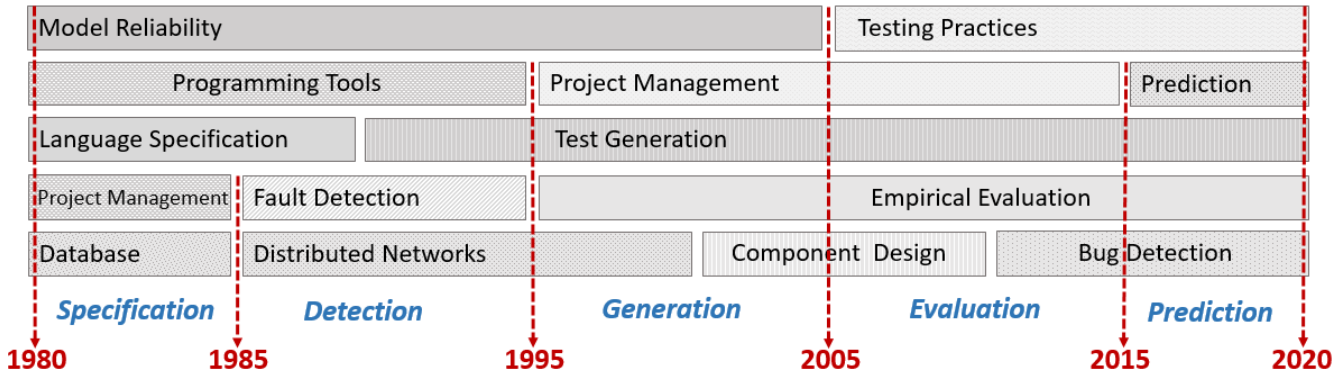

FIGURE 8. Five developmental ages of software testing.

studied more often, after 1995 (see Appendix-A), was one of the top five topics from 2015 to 2020 (see Figure 8). The average acceleration of ''Prediction'' (see Figure 2, 0.11) was also the highest among all topics, and its recent acceleration value was significantly higher (see Figure 6, 0.75) compared to the other topics where its trend-line also indicates a steady increase (see Appendix-A). Additionally, although their average volumes were lower (see Appendix-A, 0.15, 0.20 and 0.16 respectively), the topics of ''Security Vulnerability'', ''Open Source'' and ''Mobile Application'' are also showing higher recent accelerations (see Figure 6) with an increasing trend-line (see Figure 4). These results indicate that in the next decade, the topics of ''Prediction'', ''Security Vulnerability'', ''Open Source'' and ''Mobile Application'' will dominate the testing studies. With the improvements in artificial intelligence studies, prediction in the testing process and improvements through the automation of the testing process can be expected to take place in the next decade [52], [56]. This trend shown in the current study is also supported by van Dam [55], referring to the possible impact of artificial intelligence on testing automation studies. However, this does not mean that there will be no need for software test engineers or software developers; rather, it is an indicator of their changing roles and how they work in a software life cycle.

E. LIMITATIONS AND SUGGESTIONS

This corpus-based topic modeling study, revealing the emerging themes and trends in the field of software testing from a panoramic perspective, provides a starting point and a methodological understanding for more in-depth research into the software testing phenomenon. In addition to the findings that reveal this background, this study has some limitations. The empirical corpus created for this study contained only articles published in core publication sources. In this respect, further research is recommended to expand the findings of this study using a comprehensive content analysis that includes a wider range of publication sources. In addition, the proposed semi-automated methodology can be applied to different sub-contexts of software testing processes, such as parallel computing, mobile applications, web applications, and open-source software systems, and more specific inferences can be obtained. As a result, deeper studies, which include specific content analysis on software testing and other sub-contexts of the software engineering field based on automated text mining and topic modeling, should be encouraged. The methodology of this study, which focuses on corpus-based LDA topic modeling, can be supported by approaches with different backgrounds, such as Non-Negative Matrix Factorization, Probabilistic Latent Semantic Analysis, and Hierarchical Dirichlet Process.

VI. CONCLUSION

The results of this study offer insights into software testing through the analysis of a rich corpus. This methodology can be applied regularly to analyze the trends and developments in the field of software engineering. Presenting regular feedback for the decision makers, educators, researchers and industry is critical to future-proof software testing and to take appropriate strategic decisions. The results of this study show a current trend through prediction, which is an indicator of the signals of a change in testing procedures and in the roles of software test engineers. Additionally, the findings of this study indicate an increasing trend for the topics ''Security Vulnerability'', ''Open Source'', and ''Mobile Application''. The results of the study may provide valuable insights for the industry and software communities in order to be better prepared for the possible changes in the software testing procedures using prediction-based approaches. From this perspective, new research can be conducted to better understand this change and develop strategies for educators to better prepare future test engineers with the necessary skills, thus enabling the industry to adapt and develop their testing strategies by considering these signals of change, and for decision makers to consider this information for their future decisions.

APPENDIX A

Details of the discovered topics.

TABLE 6. Five-year percentages and acceleration values of the topics.

Topic		1980-1984	1985-1989	1990-1994	1995-1999	2000-2004	2005-2009	2010-2014	2015-2019	AVG
Test Generation	C%	0.06	0.07	0.24	0.30	0.49	1.07	1.62	1.99	0.73
	Y%	4.02	2.94	5.09	4.76	5.58	6.24	6.48	5.86	5.12
	AC	-0.28	-0.08	0.48	-0.10	0.23	0.02	0.00	-0.12	0.02
Empirical Evaluation	C%	0.05	0.10	0.16	0.30	0.42	0.79	1.28	1.89	0.63
	Y%	3.72	4.01	3.42	4.70	4.76	4.51	5.09	5.57	4.47
	AC	0.38	0.47	-0.56	0.24	0.09	0.02	0.11	0.02	0.10
Testing Practices	C%	0.04	0.08	0.17	0.25	0.34	0.72	1.17	1.83	0.58
	Y%	2.85	3.31	3.53	4.06	3.92	4.19	4.64	5.37	3.98
	AC	-0.06	0.10	0.10	0.57	-0.63	0.14	0.15	0.07	0.06
Project Management	C%	0.06	0.09	0.19	0.35	0.37	0.65	0.79	0.90	0.42
	Y%	4.06	3.76	4.03	5.39	4.16	3.80	3.13	2.67	3.87
	AC	1.00	-0.58	0.12	0.27	-0.23	-0.14	-0.02	-0.22	0.02
Program Analysis	C%	0.05	0.07	0.21	0.24	0.34	0.58	0.74	0.90	0.39
	Y%	3.65	2.63	4.39	3.72	3.78	3.38	2.94	2.66	3.39
	AC	0.16	-0.29	0.56	-0.48	0.29	-0.20	0.03	-0.18	-0.01
Component Design	C%	0.04	0.10	0.19	0.27	0.46	0.65	0.64	0.68	0.38
	Y%	2.86	4.00	3.92	4.33	5.27	3.98	2.54	2.01	3.62
	AC	0.29	0.18	-0.09	0.32	-0.17	-0.29	-0.08	-0.19	0.00
Model Reliability	C%	0.06	0.13	0.38	0.44	0.42	0.48	0.49	0.55	0.37
	Y%	4.62	5.37	7.70	6.79	4.82	2.83	1.97	1.62	4.46
	AC	-0.23	-0.16	0.83	-0.67	-0.37	-0.32	-0.05	-0.07	-0.13
Programming Tools	C%	0.19	0.29	0.25	0.23	0.24	0.47	0.54	0.69	0.36
	Y%	13.68	11.49	5.40	3.57	2.69	2.67	2.13	2.02	5.46
	AC	0.93	-0.74	-1.45	-0.19	0.01	0.09	-0.25	-0.06	-0.21
Bug Detection	C%	0.02	0.02	0.04	0.06	0.15	0.51	0.81	1.19	0.35
	Y%	1.08	0.84	0.75	0.91	1.62	2.93	3.23	3.50	1.86
	AC	-0.29	0.10	-0.06	0.07	0.33	0.12	0.04	0.15	0.06
Source Code	C%	0.01	0.02	0.04	0.06	0.10	0.47	0.72	1.24	0.33
	Y%	0.75	0.89	0.89	1.01	1.10	2.77	2.87	3.64	1.74
	AC	0.13	0.07	-0.01	0.00	0.09	0.24	-0.03	0.27	0.09
Parallel Computing	C%	0.06	0.08	0.16	0.22	0.23	0.40	0.62	0.81	0.32
	Y%	3.91	3.25	3.40	3.34	2.66	2.47	2.47	2.40	2.99
	AC	-0.68	0.04	0.14	-0.15	0.13	-0.20	0.09	-0.07	-0.09
Regression Testing	C%	0.01	0.03	0.08	0.14	0.23	0.41	0.73	0.95	0.32
	Y%	0.89	1.10	1.51	2.19	2.65	2.35	2.90	2.78	2.05
	AC	0.29	0.01	0.09	0.02	0.11	-0.01	0.07	-0.13	0.06
Fault Detection	C%	0.05	0.09	0.25	0.25	0.27	0.41	0.59	0.63	0.32
	Y%	3.50	4.04	5.26	3.90	3.10	2.40	2.32	1.85	3.30
	AC	0.26	-0.19	0.22	-0.27	-0.16	-0.03	0.07	-0.22	-0.04
Programming Languages	C%	0.04	0.07	0.12	0.18	0.34	0.55	0.58	0.61	0.31
	Y%	3.02	3.14	2.66	2.82	3.89	3.31	2.33	1.80	2.87
	AC	-0.03	0.26	-0.53	0.18	0.17	-0.18	-0.06	-0.21	-0.05
Verification	C%	0.04	0.04	0.11	0.19	0.34	0.48	0.63	0.66	0.31
	Y%	2.55	1.57	2.28	2.93	3.81	2.89	2.52	1.96	2.56
	AC	0.08	-0.34	0.27	0.05	0.18	-0.31	0.03	-0.15	-0.02
Algorithm Optimization	C%	0.01	0.02	0.06	0.14	0.17	0.40	0.70	0.95	0.31
	Y%	0.88	0.98	1.28	2.12	1.92	2.29	2.76	2.78	1.88
	AC	0.24	-0.07	0.16	0.02	0.06	0.01	-0.02	0.09	0.06
Requirements	C%	0.02	0.07	0.11	0.14	0.20	0.43	0.56	0.90	0.30
	Y%	1.57	2.72	2.19	2.23	2.29	2.53	2.22	2.66	2.30
	AC	0.34	-0.13	0.03	0.07	-0.01	0.12	-0.18	0.08	0.04
Bug Reporting	C%	0.01	0.01	0.02	0.04	0.05	0.29	0.73	1.29	0.30
	Y%	0.39	0.47	0.47	0.55	0.51	1.66	2.87	3.80	1.34
	AC	0.15	-0.12	0.05	0.04	0.00	0.34	0.11	0.20	0.09
Prediction	C%	0.01	0.01	0.04	0.07	0.10	0.31	0.49	1.32	0.29
	Y%	0.46	0.23	0.84	1.05	1.16	1.70	1.91	3.84	1.40
	AC	-0.08	-0.04	0.08	0.02	0.09	0.10	-0.01	0.75	0.11
Model Transformation	C%	0.02	0.04	0.07	0.09	0.22	0.46	0.65	0.78	0.29
	Y%	1.39	1.74	1.36	1.34	2.44	2.67	2.57	2.29	1.98
	AC	0.04	0.48	-0.25	-0.04	0.25	-0.08	0.01	-0.06	0.04
Language Specification	C%	0.10	0.10	0.18	0.19	0.23	0.38	0.48	0.64	0.29
	Y%	7.10	4.29	4.03	2.96	2.68	2.26	1.90	1.88	3.39
	AC	-1.16	-0.19	0.07	-0.05	-0.14	-0.06	0.00	0.01	-0.19
Constraint Programming	C%	0.04	0.08	0.16	0.21	0.27	0.40	0.59	0.55	0.29
	Y%	3.22	3.23	3.31	3.32	3.03	2.48	2.36	1.63	2.82
	AC	-0.32	0.37	-0.29	-0.06	0.15	-0.08	-0.13	-0.01	-0.05

TABLE 6. *(Continued.)* **Five-year percentages and acceleration values of the topics.**

Distributed Networks	C%	0.03	0.18	0.21	0.31	0.30	0.41	0.38	0.44	0.28
	Y%	2.08	7.72	4.41	4.85	3.49	2.52	1.54	1.31	3.49
	AC	1.06	0.85	-0.76	0.10	-0.49	-0.13	-0.19	0.00	0.06
Embedded Control	C%	0.03	0.06	0.10	0.11	0.19	0.34	0.51	0.73	0.26
	Y%	1.96	2.26	2.10	1.68	2.09	2.05	2.03	2.15	2.04
	AC	0.75	-0.24	0.00	-0.02	0.02	-0.10	0.10	0.02	0.07
Program Execution	C%	0.02	0.03	0.07	0.07	0.10	0.34	0.59	0.80	0.25
	Y%	1.51	1.21	1.46	1.09	1.11	2.00	2.37	2.35	1.64
	AC	-0.14	-0.14	0.15	0.02	-0.03	0.20	0.09	0.05	0.02
Fault Localization	C%	0.01	0.02	0.03	0.04	0.08	0.28	0.59	0.95	0.25
	Y%	0.97	0.60	0.63	0.61	0.91	1.70	2.34	2.79	1.32
	AC	-0.18	-0.01	0.04	-0.03	0.06	0.24	0.10	0.15	0.05
Test Strategies	C%	0.04	0.04	0.14	0.15	0.20	0.30	0.51	0.61	0.25
	Y%	3.35	1.67	2.84	2.37	2.28	1.71	2.02	1.80	2.26
	AC	0.28	-0.73	0.22	-0.14	0.10	-0.18	0.12	-0.10	-0.06
Metrics	C%	0.03	0.04	0.15	0.22	0.19	0.37	0.46	0.50	0.25
	Y%	1.99	1.67	3.25	3.48	2.20	2.16	1.87	1.47	2.26
	AC	0.26	-0.06	0.36	-0.20	-0.27	0.10	-0.12	-0.04	0.00
Test Coverage	C%	0.03	0.06	0.13	0.17	0.18	0.37	0.47	0.52	0.24
	Y%	2.37	2.39	2.52	2.75	1.97	2.12	1.86	1.52	2.19
	AC	-0.42	0.27	0.14	-0.22	0.00	-0.01	-0.10	-0.07	-0.05
Test Sequence	C%	0.03	0.04	0.12	0.14	0.26	0.36	0.52	0.47	0.24
	Y%	1.93	1.77	2.45	2.13	3.00	2.18	2.09	1.37	2.11
	AC	-0.70	0.22	0.29	0.02	-0.13	-0.02	-0.14	-0.02	-0.06
Agile Practices	C%	0.01	0.03	0.04	0.06	0.14	0.39	0.47	0.61	0.22
	Y%	0.98	1.17	0.92	0.89	1.54	2.14	1.89	1.79	1.42
	AC	-0.32	0.17	-0.10	0.09	0.03	0.22	-0.11	-0.06	-0.01
Concurrent Programming	C%	0.01	0.04	0.06	0.07	0.09	0.27	0.60	0.59	0.22
	Y%	0.79	1.58	1.38	1.07	0.98	1.61	2.38	1.74	1.44
	AC	0.43	-0.17	-0.04	-0.02	0.02	0.08	0.14	-0.17	0.03
Web Applications	C%	0.00	0.00	0.01	0.04	0.13	0.42	0.54	0.56	0.21
	Y%	0.26	0.20	0.18	0.55	1.43	2.44	2.15	1.65	1.11
	AC	-0.01	-0.07	0.03	0.08	0.44	0.05	-0.11	-0.23	0.02
Graph Algorithms	C%	0.05	0.07	0.12	0.13	0.16	0.29	0.42	0.48	0.21
	Y%	3.31	2.86	2.49	2.07	1.81	1.61	1.69	1.40	2.15
	AC	-1.38	0.21	0.07	-0.13	-0.27	0.18	-0.06	-0.04	-0.18
Product Line Inspection	C%	0.01	0.03	0.08	0.17	0.24	0.29	0.41	0.45	0.21
	Y%	0.51	1.40	1.75	2.59	2.74	1.66	1.63	1.32	1.70
	AC	0.09	0.17	-0.03	0.11	0.23	-0.28	-0.02	-0.02	0.03
Database	C%	0.07	0.09	0.11	0.12	0.16	0.28	0.38	0.46	0.21
	Y%	4.94	3.73	2.27	1.95	1.80	1.67	1.49	1.36	2.40
	AC	-1.19	0.00	-0.10	0.03	0.05	-0.13	-0.01	-0.05	-0.18
Open Source	C%	0.01	0.01	0.02	0.03	0.07	0.17	0.38	0.91	0.20
	Y%	0.51	0.36	0.35	0.48	0.72	1.03	1.49	2.65	0.95
	AC	0.05	0.01	-0.09	0.13	0.09	-0.06	0.12	0.29	0.07
Event Tracing	C%	0.01	0.02	0.03	0.05	0.11	0.24	0.44	0.58	0.18
	Y%	0.50	0.90	0.68	0.80	1.20	1.37	1.71	1.70	1.11
	AC	-0.06	0.23	-0.14	0.06	0.05	0.05	0.12	-0.14	0.02
Mutation Testing	C%	0.01	0.01	0.06	0.05	0.06	0.17	0.42	0.56	0.17
	Y%	0.44	0.38	1.17	0.78	0.71	0.98	1.65	1.64	0.97
	AC	-0.05	0.08	0.16	-0.01	-0.24	0.30	0.04	-0.10	0.02
Mobile Application	C%	0.00	0.01	0.01	0.02	0.04	0.10	0.26	0.81	0.16
	Y%	0.10	0.30	0.20	0.29	0.51	0.56	1.00	2.39	0.67
	AC	0.03	0.01	-0.04	0.12	-0.04	0.04	0.13	0.24	0.06
Version Update	C%	0.01	0.03	0.05	0.07	0.09	0.19	0.31	0.46	0.15
	Y%	0.98	1.28	1.01	1.13	1.00	1.09	1.25	1.34	1.14
	AC	0.35	0.05	-0.19	0.08	-0.08	0.08	-0.01	0.04	0.04
Security Vulnerability	C%	0.01	0.01	0.01	0.03	0.06	0.19	0.35	0.56	0.15
	Y%	0.35	0.56	0.23	0.47	0.66	1.08	1.40	1.63	0.80
	AC	-0.01	0.01	0.06	0.09	-0.01	0.06	0.01	0.30	0.06

C%: Percentage of the topics in the corpus
Y%: Percentage of the topics in the same yearly period
AC: Average acceleration calculated according to the consecutive yearly changes of Y%

REFERENCES

[1] F. Pudlitz, F. Brokhausen, and A. Vogelsang, "What am i testing and where? Comparing testing procedures based on lightweight requirements annotations," *Empirical Softw. Eng.*, vol. 25, no. 4, pp. 2809–2843, Jul. 2020.

[2] T. Mantere and J. T. Alander, "Evolutionary software engineering, a review," *Appl. Soft Comput.*, vol. 5, no. 3, pp. 315–331, Mar. 2005.

[3] M. Kuhrmann, P. Diebold, and J. Münch, "Software process improvement: A systematic mapping study on the state of the art," *PeerJ Comput. Sci.*, vol. 2, p. e62, May 2016.

[4] S. K. Swain, D. P. Mohapatra, and R. Mall, "Test case generation based on use case and sequence diagram," *Int. J. Softw. Eng.*, vol. 3, no. 2, pp. 21–52, Jul. 2010.
[5] A. A. Salatino, F. Osborne, and E. Motta, "How are topics born? Understanding the research dynamics preceding the emergence of new areas," *PeerJ Comput. Sci.*, vol. 3, p. e119, Jun. 2017.
[6] S. Ahmed, "Overview of software testing standard ISO/IEC/IEEE 29119," *Int. J. Comput. Sci. Netw. Secur.*, vol. 18, no. 2, pp. 112–116, 2018.
[7] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, U.K.: Cambridge Univ. Press, 2016.
[8] É. F. de Souza, R. D. A. Falbo, and N. L. Vijaykumar, "Knowledge management initiatives in software testing: A mapping study," *Inf. Softw. Technol.*, vol. 57, pp. 378–391, Jan. 2015.
[9] P. Tripathy and K. Naik, *Software Testing and Quality Assurance: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2011.
[10] P. A. Da Mota Silveira Neto, I. D. C. Machado, J. D. McGregor, E. S. De Almeida, and S. R. De Lemos Meira, "A systematic mapping study of software product lines testing," *Inf. Softw. Technol.*, vol. 53, no. 5, pp. 407–423, May 2011.
[11] I. Banerjee, B. Nguyen, V. Garousi, and A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository," *Inf. Softw. Technol.*, vol. 55, no. 10, pp. 1679–1694, Oct. 2013.
[12] L. Wang, J. Yuan, X. Yu, J. Hu, X. Li, and G. Zheng, "Generating test cases from UML activity diagram based on gray-box method," in *Proc. 11th Asia–Pacific Softw. Eng. Conf.*, 2004, pp. 284–291.
[13] C. Catal and D. Mishra, "Test case prioritization: A systematic mapping study," *Softw. Quality J.*, vol. 21, no. 3, pp. 445–478, 2013.
[14] A. K. Jena, S. K. Swain, and D. P. Mohapatra, "Model based test case generation from UML sequence and interaction overview diagrams," in *Computational Intelligence in Data Mining* (Smart Innovation, Systems and Technologies), vol. 32. New Delhi, India: Springer, 2015, pp. 247–257.
[15] P. Samuel, R. Mall, and P. Kanth, "Automatic test case generation from UML communication diagrams," *Inf. Softw. Technol.*, vol. 49, no. 2, pp. 158–171, Feb. 2007.
[16] S. Matalonga, F. Rodrigues, and G. H. Travassos, "Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review," *J. Syst. Softw.*, vol. 131, pp. 1–21, Sep. 2017.
[17] A. A. Pomeransky and I. B. Khriplovich, "Equations of motion of spinning relativistic particle in external fields," *J. Exp. Theor. Phys.* vol. 14, no. 1, pp. 839–849, 1999.
[18] N. M. Minhas, S. Masood, K. Petersen, and A. Nadeem, "A systematic mapping of test case generation techniques using UML interaction diagrams," *J. Softw., Evol. Process*, vol. 32, no. 6, p. e2235, Jun. 2020.
[19] M. Dadkhah, S. Araban, and S. Paydar, "A systematic literature review on semantic web enabled software testing," *J. Syst. Softw.*, vol. 162, Apr. 2020, Art. no. 110485.
[20] V. Garousi, M. Felderer, Ç. M. Karapýçak, and U. Yýlmaz, "Testing embedded software: A survey of the literature," *Inf. Softw. Technol.*, vol. 104, pp. 14–45, Dec. 2018.
[21] R. Suman and S. Sahibuddin, "User acceptance testing in mobile health applications: An overview and the challenges," in *Proc. 2nd Int. Conf. Inf. Sci. Syst.*, Mar. 2019, pp. 145–149.
[22] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing and verification in service-oriented architecture: A survey," *Softw. Test., Verification Rel.*, vol. 23, no. 4, pp. 261–313, Jun. 2013.
[23] M. Palacios, J. García-Fanjul, and J. Tuya, "Testing in service oriented architectures with dynamic binding: A mapping study," *Inf. Softw. Technol.*, vol. 53, no. 3, pp. 171–189, Mar. 2011.
[24] A. Kaur and K. Kaur, "Investigation on test effort estimation of mobile applications: Systematic literature review and survey," *Inf. Softw. Technol.*, vol. 110, pp. 56–77, Jun. 2019.
[25] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 93, pp. 74–93, Jan. 2018.
[26] B. Jiang, T. H. Tse, W. Grieskamp, N. Kicillof, Y. Cao, X. Li, and W. K. Chan, "Assuring the model evolution of protocol software specifications by regression testing process improvement," *Softw., Pract. Exper.*, vol. 4, no. 10, pp. 1073–1103, 2011.
[27] J. A. Whittaker, *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. London, U.K.: Pearson Education, 2010.
[28] S. U. R. Khan, S. P. Lee, N. Javaid, and W. Abdul, "A systematic review on test suite reduction: Approaches, experiment's quality evaluation, and guidelines," *IEEE Access*, vol. 6, pp. 11816–11841, 2018.
[29] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: A survey," *Softw. Test. Verification Rel.*, vol. 15, no. 3, pp. 167–199, Sep. 2005.
[30] G. Tebes, D. Peppino, P. Becker, G. Matturro, M. Solari, and L. Olsina, "Analyzing and documenting the systematic review results of software testing ontologies," *Inf. Softw. Technol.*, vol. 123, Jul. 2020, Art. no. 106298.
[31] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in *Proc. IEEE 4th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2011, pp. 476–485.
[32] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing," *Inf. Softw. Technol.*, vol. 80, pp. 195–216, Dec. 2016.
[33] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *J. Syst. Softw.*, vol. 117, pp. 334–356, Jul. 2016.
[34] K. Cosh, S. Ramingwong, N. Eiamkanitchat, and L. Ramingwong, "Automatically identifying themes and trends in software engineering research," in *Proc. 10th Int. Conf. Knowl. Smart Technol. (KST)*, Jan. 2018, pp. 106–111.
[35] G. Mathew, A. Agrawal, and T. Menzies, "Finding trends in software research," *IEEE Trans. Softw. Eng.*, early access, Sep. 14, 2019, doi: 10.1109/TSE.2018.2870388.
[36] H. Nabli, R. Ben Djemaa, and I. A. Ben Amor, "Efficient cloud service discovery approach based on LDA topic modeling," *J. Syst. Softw.*, vol. 146, pp. 233–248, Dec. 2018.
[37] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, nos. 4–5, pp. 993–1022, 2003.
[38] F. Gurcan, G. G. M. Dalveren, N. E. Cagiltay, and A. Soylu, "Detecting latent topics and trends in software engineering research since 1980 using probabilistic topic modeling," *IEEE Access*, vol. 10, pp. 74638–74654, 2022.
[39] F. Gurcan, N. E. Cagiltay, and K. Cagiltay, "Mapping human–computer interaction research themes and trends from its existence to today: A topic modeling-based review of past 60 years," *Int. J. Hum. Comput. Interact.*, vol. 37, no. 3, pp. 267–280, 2021.
[40] P. Mongeon and A. Paul-Hus, "The journal coverage of web of science and scopus: A comparative analysis," *Scientometrics*, vol. 106, no. 1, pp. 213–228, Jan. 2016.
[41] F. Gurcan and N. E. Cagiltay, "Research trends on distance learning: A text mining-based literature review from 2008 to 2018," *Interact. Learn. Environ.*, early access, pp. 1–22, Sep. 2020.
[42] F. Gurcan and N. E. Cagiltay, "Exploratory analysis of topic interests and their evolution in bioinformatics research using semantic text mining and probabilistic topic modeling," *IEEE Access*, vol. 10, pp. 31480–31493, 2022.
[43] M. F. Porter, "Snowball: A language for stemming algorithms," 2001.
[44] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, no. 4, pp. 77–84, Apr. 2012.
[45] F. Gurcan, O. Ozyurt, and N. E. Cagitay, "Investigation of emerging trends in the e-learning field using latent Dirichlet allocation," *Int. Rev. Res. Open Distrib. Learn.*, vol. 22, no. 2, pp. 1–18, Jan. 2021.
[46] A. E. Gelfand, "Gibbs sampling," *J. Amer. Stat. Assoc.*, vol. 95, no. 452, pp. 1300–1304, Dec. 2000.
[47] M. Konrad. (2017). *Text Mining and Topic Modeling Toolkit*. Accessed: Jan. 21, 2022. [Online]. Available: https://pypi.org/project/tmtoolkit/
[48] D. Mimno, H. M. Wallach, E. Talley, M. Leenders, and A. McCallum, "Optimizing semantic coherence in topic models," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2011, pp. 262–272.
[49] A. A. Sawant, P. H. Bari, and P. Chawan, "Software testing techniques and strategies," *Int. J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 980–986, 2012.
[50] B. W. Boehm, "Software engineering economics," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 1, pp. 4–21, Jan. 1984.
[51] D. Gelperin and B. Hetzel, "The growth of software testing," *Commun. ACM*, vol. 31, no. 6, pp. 687–695, 1988.
[52] J. W. Cangussu, S. W. Haider, K. Cooper, and M. Baron, "On the selection of software defect estimation techniques," *Softw. Test., Verification Rel.*, vol. 21, no. 2, pp. 125–152, Jun. 2011.
[53] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal defect classification—A concept for in-process measurements," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, Nov. 1992.

[54] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG system: An approach to testing based on combinatorial design," *IEEE Trans. Softw. Eng.*, vol. 23, no. 7, pp. 437–444, Jul. 1997.
[55] K. Van Dam, "The future of testing," in *The Future of Software Quality Assurance*. Cham, Switzerland: Springer, 2020, pp. 197–205.
[56] B. Caglayan, A. T. Misirli, A. B. Bener, and A. Miranskyy, "Predicting defective modules in different test phases," *Softw. Quality J.*, vol. 23, no. 2, pp. 205–227, Jun. 2015.

FATIH GURCAN received the Ph.D. degree in computer engineering from Karadeniz Technical University. He was an Instructor at the Department of Informatics, Karadeniz Technical University, from 2001 to 2014, where he has been an Instructor with the Center for Research and Application in Distance Education, since 2015. His research interests include trend analysis, sentiment analysis, statistical topic modeling, engineering education, data mining, machine learning, big data analytics, and text mining.

GONCA GOKCE MENEKSE DALVEREN received the Ph.D. degree in software engineering from Atilim University. She was a Post-Doctoral Researcher at the Department of Computer Science, Norwegian University of Science and Technology. Currently, she is with the Software Engineering Department, Atilim University. Her research interests include software engineering, eye tracking, medical informatics, and human–computer interaction.

NERGIZ ERCIL CAGILTAY received the Ph.D. degree in instructional technologies from Middle East Technical University. She worked for commercial and government organizations as a project manager for more than eight years in Turkey. She also worked for the Indiana University Digital Library Program as a System Analysis and a Programmer for four years. She has been a Professor with the Software Engineering Department, Atilim University, Turkey, since 2003. Her main research interests include information systems, medical information systems, engineering education, instructional systems technologies, distance education, e-learning, and medical education.

DUMITRU ROMAN is currently working as a Senior Research Scientist with SINTEF, Norway. He has wide experience with initiating, leading, and carrying out (research-intensive) projects on data management and service-oriented topics. He is also active in the data management field, particularly in the emerging Data-as-a-Service (DaaS) domain. He holds an Adjunct Associate Professorship with the University of Oslo, Norway.

AHMET SOYLU received the Ph.D. degree in computer science from University of Leuven, in 2012. He is currently an Associate Professor at the Norwegian University of Science and Technology. His main research interests include ontology-driven information systems and ontology-driven design of information systems from a human–computer interaction perspective.