# Recovering Traceability Links Between Code and Documentation: A Retrospective

Giulio Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, *Senior Member, IEEE*, and Ettore Merlo

***Abstract*—Software system documentation is almost always expressed informally in natural language and free text. Examples include requirement specifications, design documents, manual pages, system development journals, error logs, and related maintenance reports. In our 2002 seminal paper we proposed a method based on information retrieval to recover traceability links between source code and free text documents. A premise of our work was that programmers use meaningful names for program items, such as functions, variables, types, classes, and methods. The paper paved the way to the adoption of IR in software engineering opening a new perspective. Reflecting on the past twenty years we briefly overview the many results that have been achieved, however, the emergence of new technologies, such as AI, pose unprecedented challenges.**

***Index Terms*—Redocumentation, information retrieval, object orientation, program comprehension, traceability.**

## I. Introduction

At the turn of the 21st century, software engineering research was increasingly concerned with addressing the complexities introduced by the rapid growth of large-scale, legacy, and distributed software systems. Among the most critical challenges was the need to establish and maintain traceability—the ability to track and link disparate software artifacts such as requirements, design documents, source code, and test cases. Traceability links were recognized as essential for program comprehension, maintenance, impact analysis, and regression testing, yet they were rarely explicitly established or adequately managed throughout the software development lifecycle [1], [2].

Two major research trajectories dominated the field at the time. On the one hand, reverse engineering research largely focused on the recovery of specific views and documents, such as high-level architectural representations, detailed UML diagrams, and domain models derived from source code or runtime behaviour [3], [4]. These efforts emphasized reconstructing individual artifacts in isolation but paid limited attention to establishing interconnections or "networks of relations" among software entities. On the other hand, traceability research was predominantly concerned with making explicit and managing traceability links during development and leveraging these links for tasks such as impact analysis and regression testing [5], [6]. This work assumed the presence of traceability links from the outset, rather than focusing on their recovery in cases where such links were missing or incomplete.

Our paper, published in 2002 [7], introduced a novel approach that bridged this gap by leveraging reverse engineering techniques to recover traceability links between source code and free-text documentation. This innovation marked a departure from prior approaches by moving beyond the recovery of isolated artifacts and establishing a framework to create networks of relations among software entities. Specifically, we demonstrated the feasibility and utility of using information retrieval (IR) models [8] to automatically recover traceability links from natural language documents and source code identifiers. By doing so, we enabled the semi-automatic recovery of traceability links that could support various software engineering tasks, including program comprehension, maintenance, and design recovery.

At the time, the use of IR models in software engineering was still in its infancy, with only applications to software reuse (e.g., [9]). Our contribution extended this line of investigation by applying IR models to the problem of traceability recovery between source code and higher level software artifacts and rigorously evaluating their performance in two real-world case studies. These case studies provided empirical evidence that IR models could achieve high recall and precision in recovering traceability links, even when significant differences existed between the vocabularies of source code and documentation. Indeed, the main intuition of our work was that all software artifacts contains text and that the text similarity in different components are indicators of traceability relations between these components.

Besides boosting the field of traceability recovery, our paper provided a contribution to the application of IR, text mining, and natural language processing to different software engineering tasks involving any kind of software artifacts. Indeed, as observed later by Hindle et al. [10], similarly to natural

Received 20 January 2025; accepted 21 January 2025. Date of publication 27 January 2025; date of current version 17 March 2025. Recommended for acceptance by S. Uchitel. *(Corresponding author: Gerardo Canfora.)*

Giulia Antoniol and Ettore Merlo are with the Department of Computer and Software Engineering, Polytechnique Montreal, Montreal, QC H3C 3A7, Canada.

Gerardo Canfora is with the University of Sannio, 82100 Benevento, Italy (e-mail: canfora@unisannio.it).

Gerardo Casazza is with DENSO International Europe B.V., World Trade Centre, 1077 XX Amsterdam, The Netherlands.

Andrea De Lucia is with the Department of Computer Science, University of Salerno, 84084 Fisciano SA, Italy.

Digital Object Identifier 10.1109/TSE.2025.3534027

language most software is likely to be repetitive and predictable and can be usefully modeled by statistical language models that can support software engineering tasks.

In Section II we briefly review literature to identify and discuss developments in the field and the influence and impact of our work. For sake of space limitations our literature review does not intend to be systematic or exhaustive. A systematic literature review on IR-based traceability recovery can be found in [11]. In Section III we then highlight open problems and future directions for research, while section IV provides concluding remarks.

## II. Retrospective

In [7] we defined an IR-based traceability recovery process that starts from a set of source code components and higher level documents to be traced (i.e., the corpus of the IR method), where software artifacts of one type are used as queries and the artifacts of the other type as document base. The process consists of several steps organized in a pipeline, where the output of each step constitutes the input for the next step. For example, for the Vector Space IR Model (VSM) the process can be defined as follows:

1) parsing code components and higher level documents, extracting and pre-processing terms, by removing non textual tokens (i.e., operators, special symbols, some numbers, etc.), splitting compound terms, removing stop words (i.e., articles, adverbs, etc), and stemming (reducing inflected words to the root), to build a bag of words for each software artifact;
2) indexing the corpus and building a *term-by-document* matrix, where each entry of the matrix represents the weight of a term (row) for a document[1] (column) and is usually given by the product of a local weight representing the importance of the term for that document and a global weight representing the extent to which the term is specific for the document; for example, in case a *TF-IDF* scheme is used for the weight, the local weight is represented by the frequency of the term in the document (*TF*), while the global weight is represented by the inverse document frequency (*IDF*), i.e. the ratio between the number of total documents and the number of documents containing that term;
3) computing the similarity between source code components and higher level documents, for example by computing the cosine of the angle between the vectors (columns of the matrix) representing the software artifacts in the space of the terms;
4) producing a ranked list of candidate traceability links ordered by similarity values.

The process is semi-automatic and requires the software developer to analyze and classify the traceability links suggested by the IR-based tool, by marking correct links and discarding false positives. Other IR models might use a variant of this pipeline. Indeed, different IR models can be used to index the corpus and compute the similarity between software artifacts. In our paper [7] we experimented and compared probabilistic and vector space models in two case studies for the detection of traceability links between source code and functional requirements/user manual pages, respectively. We measured the results achieved by these models in terms of typical information retrieval metrics, namely *recall* (which measures the percentage of correct links that are retrieved) and *precision* (which measures the percentage of retrieved links that are correct). The results, however, did not show the superiority of an IR model with respect to the other.

To measure the performances of IR models we need to cut the ranked list of candidate links at some point and consider only traceability links above the cut point. In our paper [7] we used a fixed cut point (a fixed number of traceability links in the ranked list) as well as a scale threshold (a percentage of the maximum similarity value). Other approaches include a variable cut point (a percentage of traceability links in the ranked list), a constant threshold [12], and a variable threshold, where the constant threshold is projected from the interval [0, 1] into the interval [min similarity, max similarity] [13].

With the goal of improving IR-based traceability, other authors used different IR models, including Latent Semantic Indexing (LSI) [12], [13], [14], an extension of VSM that helps to deal automatically with the problem of synonymy by projecting the documents from the space of the terms to a reduced space of concepts [15], as well as other probabilistic models, such as probabilistic LSI and the Jensen-Shannon information theory model [16], Latent Latent Dirichlet Allocation (LDA) [17], a technique used to infer semantic topics from a text corpus [18], and Relational Topic Model (RTM) [19], a hierarchical probabilistic model of links and document attributes [20]. Capobianco et al. [21] used a numerical analysis approach mapping the frequency of terms contained in artifacts on particular interpolation curves, called B-Splines [22]. Oliveto et al. [23] conducted experiments that demonstrated the equivalence of different IR methods in terms of performances. However, a finer grained analysis of the links retrieved by the different models revealed that topic modeling methods are able to retrieve links missed by the other methods, paving the way to improve the results of IR-based traceability through their integration [19].

Besides tracing source code to higher level requirements, IR based approaches have been used (possibly in combination with other techniques) to recover traceability links between different types of artifacts, including higher level to lower level requirements [14], change requests to changed components [24], emails to source code artifacts [25], textual domain elements to source code [26], test to code components [27], [28], [29], bug reports and committed changes [30], [31], service specifications and source code components [32], requirements to design and tests [33], bug reports to test cases [34], [35], business rules expressed as data constraints to source code [36], regulatory codes and requirements [37], and even between elements of informal hand-drawn sketches on large interactive displays [38].

The application of IR, and in general text mining and natural language processing moved very quickly from traceability recovery to other software engineering tasks. The closest

[1]The term document is used to generically refer to any kind of low level (e.g. source code component) or high level software artifact.

task to traceability recovery is *feature location*, which consists of identifying the code components where some given high level concept is located. Feature or concept location was not a new topic, as it has been recognized as an essential task for impact analysis and program comprehension during software maintenance [39]. However this type of research received a deep boost with the intuition that a feature expressed in natural language could be used as a query to search code components implementing that concept [40], [41], [42], [43], [44]. Similar software engineering tasks include code search [45], bug localization [46], [47], [48], [49], duplicated bug report detection [50], software retrieval [51], API discussion mining [52], [53], and analysis of user reviews on app marketplaces for software maintenance [54], [55], [56]. Other tasks include the definition and application of software metrics [57], [58], source code readability [59], semantic clustering and topic extraction [60], [61], [62], program comprehension and source code summarization [63], [64], source code labelling [65], assertion generation [66], software testing [67], and requirement analysis and classification [68].

Industrial experiences and case studies have also been conducted to prove the effectiveness of the use of IR methods for traceability recovery and other software engineering tasks [33], [67], [69], [70]. One of the authors of the paper also brought and adapted IR-based techniques into the software processes of his company, DENSO International Europe, to automatically dispatch maintenance requests to the proper maintenance queue.

The main problem of IR based traceability recovery is that these techniques are not able to raise all correct links in the upper part of the ranked list and some correct links still remain in the lower part of the list. Therefore, while most correct links are in the upper part of the ranked list, in the lower part of the list the correct links are very sparse and this makes very hard for the software engineer to classify and recover all of them. One of the causes that researchers have identified for this problem is the mismatch between the vocabularies of higher level documents and source code components. Similar problems are identified for other IR-based software engineering tasks involving source code. Attempts to improve the performances of IR-based methods through *enhancement strategies* [11] have then been proposed. Several approaches tried to reduce the mismatch between high level documents and source code by improving and normalizing the vocabulary of the source code through expanding abbreviations or splitting identifiers [71], [72], [73]. De Lucia et al. [74] proposed to solve this problem by inducing software engineers to improve the quality of the source code lexicon, by augmenting the Integrated Development Environment (IDE) used by the software engineer with a plug-in showing the textual similarity between the source code component being developed and the related high-level artifacts.

Other enhancement strategies act on the indexing process, for example by using a *Part of Speech (POS)* tagger, to automatically assign part-of-speech tags to words in a sentence [75]. For example, Capobianco et al. [76] found that better results are achieved when indexing only nouns contained in software artifacts. Ali et al. [77] proposed an approach that exploits POS tagging after the application of IR method by keeping only recovered links where the same verb appears both in the requirement and source code component and filtering out the other links. Ali et al. [78] also proposed an improved term weighting scheme, named *Developers Preferred Term Frequency/Inverse Document Frequency*, that gives more importance to developers' preferred types of Source Code Entities (SCEs) identified through empirical studies, e.g., domain vs. implementation-level source code terms and class names vs. method names. Other approaches aims at enhancing the corpus through the use of thesauri or ontologies [37], [79], or partition the textual information extracted from different entities of the source code [80], or try to remove the noise of the corpus (for example common terms contained in software artifact templates) by using smoothing filters on the term-by-document matrix [81].

Some researchers advocated improving IR based traceability recovery and other software engineering tasks by combining the textual information with other types of information extracted by static/dynamic analysis of source code [82] or information mined from software repositories [83], [84]. Relevance feedback provided by the software engineer when classifying links have also been used to adjust the weights of the term by document matrix, with the expectation that the tool will retrieve more correct links and less false positives in future search sessions [14], [85], [86]. Query quality prediction and reformulation has also been proposed as a way to improve the results of IR based software engineering tasks [87], [88]. Other enhancement strategies include using transitive relations [89], clustering [90], swarm intelligence [91], machine learning/deep learning [92], [93], or a combination of approaches [94]. It is worth noting that machine learning/deep learning approaches have also been proposed as an alternative to information retrieval, but still exploiting textual features extracted from source code [95], [96]. Some researchers also attempted to improve traceability recovery and other software engineering tasks by using genetic algorithms to find the best parameter configuration of IR methods like LSI or LDA [97], [98], [99].

The different enhancement strategies for IR based traceability recovery achieve higher precision levels at the same level of recall, but this still happens only in the upper part of the ranked list. A number of correct traceability links remain in the lower part of the list, where the density of false positives is much higher and the ranking provided by the IR based model does not help anymore. Some authors have performed empirical studies and have proposed processes, tools, and best practices to make the traceability recovery process cost effective for software engineers [100], [101], [102]. De Lucia et al. [13] proposed to integrate IR-based traceability recovery methods in artifact management systems and to use an incremental process with decreasing thresholds to analyze the candidate links. Experiments with developers demonstrated the effectiveness of this approach [103]. The basic idea of the incremental approach is to stop the traceability recovery process when the effort to discard false positive becomes much higher than the benefit of finding more correct links. Bavota et al. [104] analyzed the benefits of providing coverage analysis and link count information, thus highlighting software artifacts that are scarcely traced and guiding the software engineer to the identification of missing

links. Falessi et al. [105] suggested to provide information to the software engineer about the number of links still missing through the use of machine learning techniques.

Finally, traceability management encompasses other tasks besides traceability link creation or recovery [106]. Some researchers investigated the problem of generating and visualizing the rationale for trace links [107] and of maintaining and evolving trace links [13], [108], [109] within a traceability management process.

## III. Future Perspectives

More than two decades have passed since our study pioneered the use of information retrieval techniques to recover traceability links between software code and documentation. While the approach showed promising potential, numerous challenges and unresolved questions persist, especially in light of advancements in software engineering and of the growing complexity of modern systems. Key issues include:

- Scalability: Modern software systems are significantly larger and more complex compared to those analyzed in the original study.
- Semantics: Traditional IR models are constrained by their inherent lack of semantic understanding, which reduces their overall effectiveness.
- Dynamic Systems: Modern development practices, such as microservices architectures, DevOps, and continuous integration, require traceability solutions that work dynamically, providing real-time insights as code and documentation evolve.

Advances in Machine Learning (ML) and Artificial Intelligence (AI) could sensibly improve traceability management by supporting semantic understanding. As discussed in the previous section some work has already been done in this direction and recently the use of Large Language Models is also being investigated [110]. They could also ease automated feedback loops, where traceability recovery improves as systems learn from developer corrections.

However, AI applications also introduce new traceability challenges by themselves. A new research frontier is represented by investigating the traceability problem of AI-based systems. AI traceability is the capability of tracing back AI decisions to system components and life-cycle processes. System components include data such as training data, training software, and the trained models used to process information by recognition and classification programs. Life-cycle processes include the development of the training programs and libraries and the development, test and quality assurance of the classification and recognition programs. AI traceability is an important dimension of trustworthy AI [111].

AI traceability is also important for continuous auditing against AI requirements, many of which are non-functional such as explainability, bias and fairness. AI traceability also depends on AI reproducibility, that addresses the issue of replicating experiments in AI learning and decision algorithms and of obtaining the same results.

The Tailor European project for Trustworthy AI, describes AI traceability [112] and suggests two main perspectives such as: (1) provenance tracking of data, processes and artifacts involved in the development of the AI model, and (2) continuous performance monitoring of the AI model after deployment in production.

AI data traceability and auditing are parts of data quality assurance which is needed in many domains such as finance and accounting. Traceability of data supply chain has been presented in [113]. Data provenance issues are particularly relevant for AI traceability [114], [115].

Protection of data privacy [116], [117] is an issue in AI systems. Furthermore, attacks to intelligent systems can be performed by data poisoning [118], [119], so that data provenance and traceability are more and more relevant. Models themselves should also be traceable and protected [120], [121].

Since bugs have been documented and reported in intelligent systems [122], [123], the development of AI software and libraries can benefit from standard software engineering practices including the traceability approaches and techniques previously mentioned in this paper.

ML recognition and classification programs indirectly contain random aspects, because their computation is determined by the programmed instructions and also by the training data patterns, distributions, and algorithms. This is radically different from conventional systems for which the behavior can be somehow more deterministically checked against specifications and requirements.

Therefore, the traceability of AI systems also includes tracing back the decisions, obtained from an inferred model, to input data. It is related to the AI interpretability and explanability problems, including accessible explanations and justifications to humans. Explainable AI (XAI) [124], [125], [126] aims at supporting humans in understanding the AI processes and results and in building trust in AI systems.

AI explainability and auditability are also properties demanded by AI certification and regulatory compliance that are emerging domains. AI auditing aims at determining whether intelligent systems follow and comply - if required - to legal, security, and ethical constraints. Important related issues include AI privacy and security.

In many domains, training data may also incorporate personal data associated to outputs that present historically biased human decisions or social values. This can lead to biases and discrimination against groups in the population who are characterized by ethically sensitive attributes such as gender, race, age, and more. There is abundant literature about algorithmic discrimination and fairness [127], [128].

AI systems should be free of negative and undesired biases otherwise their decisions and classifications may reproduce existing biases or lead to unethical decisions in human sensitive domains, and they should also be aligned with socially and ethically acceptable values.

Traceability of AI decisions with respect to ethically sensitive input features is related to fairness analysis [129], [130]. In this direction, research objectives are fairness assessment, bias removal, and bias mitigation approaches. Tracing ethically

sensitive input features is also linked to the provenance analysis of data.

## IV. Concluding Remarks

This retrospective paper highlighted the enduring impact of IR-based methods for recovering traceability links between source code and documentation. We have briefly discussed over two decades of research that evolved from initial challenges, like vocabulary mismatches and scalability, to leveraging advancements in machine learning and artificial intelligence. Our IR framework, back in 2002, has catalysed a wealth of research and industrial applications, confirming its important role in modern software engineering.

Nowadays, the continued growth of software systems complexity calls for more robust and dynamic traceability approaches. AI technologies promise to overcome semantic limitations and scalability constraints, offering enhanced automation and real-time traceability solutions. However, these advancements also introduce challenges, particularly in maintaining transparency, fairness, and auditability within AI-driven systems.

Looking ahead, the integration of traceability into broader software engineering tasks and lifecycle processes remains a priority. Strategies such as adaptive learning, incremental approaches, and semantic enrichment are critical to addressing persistent limitations. Furthermore, the rise of AI traceability, with its focus on explainability and ethical compliance, broadens the scope of traceability to encompass not only technical accuracy but also societal values.

By reflecting on past achievements and outlining future directions, this paper has reaffirmed the importance of traceability as a dynamic and interdisciplinary field. Continued innovation and collaboration are essential to fully implement its potential in supporting the design, maintenance, and evolution of complex, large-scale software systems.

## Acknowledgment

As a final thought, we would like to express our deepest gratitude to Sebastian Uchitel and the editorial board of IEEE Transactions on Software Engineering for inviting us to contribute to this commemorative issue with our retrospective. It has been a wonderful opportunity to reconnect with friends, more than just colleagues, now that the passage of time has grayed our hair. For a few weeks, we found ourselves discussing and exchanging ideas as we used to do when young researchers, as if time had stood still, and it was truly delightful. It also provided the motivation to delve deeper into the impacts and evolutions of our original 2002 work, impacts whose significance we were perhaps not fully aware of at the time.

## References

[1] O. C. Z. Gotel and A. C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. 1st Int. Conf. Requirements Eng.*, 1994, pp. 94–101.

[2] R. S. Arnold and S. A. Bohner (eds.), *Software Change Impact Analysis.* Hoboken, NJ, USA: Wiley, 1996.

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

[4] R. Kazman, G. Abowd, L. Bass, and P. Clements, "Scenario-based analysis of software architecture," *IEEE Softw.*, vol. 13, no. 6, pp. 47–55, Nov. 1996.

[5] L. Mikael and S. Kristian, "Practical implications of traceability," *Softw. Pract. Experience*, vol. 26, no. 10, pp. 1161–1180, 1996.

[6] B. Ramesh and M. Jarke, "Towards references models for requirements traceability," *IEEE Trans. Softw. Eng.*, vol. 27, no. 1, pp. 58–93, Jan. 2001.

[7] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 970–983, Oct. 2002.

[8] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Reading, MA, USA: Addison-Wesley, 1999.

[9] Y. Maarek, D. Berry, and G. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Trans. Softw. Eng.*, vol. 17, no. 8, pp. 800–813, Aug. 1991.

[10] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. T. Devanbu, "On the naturalness of software," *Communun. ACM*, vol. 59, no. 5, pp. 122–131, 2016.

[11] M. Borg, P. Runeson, and A. Ardö, "Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability," *Empirical Softw. Eng.*, vol. 19, no. 6, pp. 1565–1616, 2014.

[12] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proc. 25th ACM/IEEE Int. Conf. Softw. Eng.*, 2003, pp. 125–135.

[13] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artefact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007, Art. no. 13.

[14] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Trans. Softw. Eng.*, vol. 32, no. 1, pp. 4–19, 2006.

[15] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *J. Amer. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990.

[16] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, 2008, pp. 103–112.

[17] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 95–104.

[18] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[19] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance*, 2011, pp. 133–142.

[20] J. Chang and D. M. Blei, "Hierarchical relational models for document networks," *Ann. Appl. Statist.*, vol. 4, no. 1, pp. 124–150, 2010.

[21] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Traceability recovery using numerical analysis," in *Proc. 16th Work. Conf. Reverse Eng.*, 2009, pp. 195–204.

[22] C. de Boor, "On calculating with b-splines," *J. Approximation Theory*, vol. 6, no. 1, pp. 50–62, 1972.

[23] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *Proc. 18th IEEE Int. Conf. Program Comprehension*, 2010, pp. 68–71.

[24] G. Canfora and L. Cerulo, "Fine grained indexing of software repositories to support impact analysis," in *Proc. Int. Workshop Mining Softw. Repositories*, 2006, pp. 105–111.

[25] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, 2010, pp. 375–384.

[26] R. Komondoor, I. Bhattacharya, D. D'Souza, and S. Kale, "Using relationships for matching textual domain models with existing code," in *Proc. 20th Work. Conf. Reverse Eng.*, 2013, pp. 371–380.

[27] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *Proc. 13th Eur. Conf. Softw. Maintenance Reeng.*, 2009, pp. 209–218.

[28] A. Qusef, G. Bavota, R. Oliveto, A. D. Lucia, and D. W. Binkley, "Recovering test-to-code traceability using slicing and textual analysis," *J. Syst. Softw.*, vol. 88, pp. 147–168, Feb. 2014.

[29] R. White, J. Krinke, and R. Tan, "Establishing multilevel test-to-code traceability links," in *Proc. 42nd ACM/IEEE Int. Conf. Softw. Eng.*, 2020, pp. 861–872.

[30] R. Wu, H. Zhang, S. Kim, and S. Cheung, "ReLink: Recovering links between bugs and changes," in *Proc. Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2011, pp. 15–25.

[31] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proc. 20th ACM SIGSOFT Symp. Found. Softw. Eng.*, 2012, pp. 63:1–63:11.

[32] R. Sindhgatta and K. Ponnalagu, "Locating components realizing services in existing systems," in *Proc. IEEE Int. Conf. Services Comput.*, 2008, pp. 127–134.

[33] M. Lormans, A. van Deursen, and H. Groß, "An industrial case study in reconstructing requirements views," *Empirical Softw. Eng.*, vol. 13, no. 6, pp. 727–760, 2008.

[34] N. Kaushik, L. Tahvildari, and M. Moore, "Reconstructing traceability between bugs and test cases: An experimental study," in *Proc. 18th Work. Conf. Reverse Eng.*, 2011, pp. 411–414.

[35] G. Gadelha, F. Ramalho1, and T. Massoni, "Traceability recovery between bug reports and test cases - a Mozilla Firefox case study," *Automated Softw. Eng.*, vol. 28, no. 8, 2021, Art. no. 8.

[36] J. M. Florez, J. Perry, S. Wei, and A. Marcus, "Retrieving data constraint implementations using fine-grained code patterns," in *Proc. 44th IEEE/ACM Int. Conf. Softw. Eng.*, 2022, pp. 1893–1905.

[37] J. Guo, M. Gibiec, and J. Cleland-Huang, "Tackling the term-mismatch problem in automated trace retrieval," *Empirical Softw. Eng.*, vol. 22, no. 3, pp. 1103–1142, 2017.

[38] M. Kleffmann, S. Rohl, M. Book, and V. Gruhn, "Evaluation of a traceability approach for informal freehand sketches," *Automated Softw. Eng.*, vol. 25, no. 1, pp. 1–43, 2018.

[39] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster, "Program understanding and the concept assigment problem," *Commun. ACM*, vol. 37, no. 5, pp. 72–82, 1994.

[40] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "SNIAFL: Towards a static noninteractive approach to feature location," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 2, pp. 195–226, 2006.

[41] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proc. 22nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2007, pp. 234–243.

[42] S. Zamani, S. P. Lee, R. Shokripour, and J. Anvik, "A noun-based approach to feature location using time-aware term-weighting," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 991–1011, 2014.

[43] D. W. Binkley, D. J. Lawrie, C. Uehlinger, and D. Heinz, "Enabling improved ir-based feature location," *J. Syst. Softw.*, vol. 101, pp. 30–42, 2015.

[44] A. Razzaq, A. Ventresque, R. Koschke, A. D. Lucia, and J. Buckley, "The effect of feature characteristics on the performance of feature location techniques," *IEEE Trans. Softw. Eng.*, vol. 48, no. 6, pp. 2066–2085, Jun. 2022.

[45] S. Wang, D. Lo, and L. Jiang, "AutoQuery: Automatic construction of dependency queries for code search," *Automated Softw. Eng.*, vol. 23, no. 3, pp. 393–425, 2016.

[46] S. Wang and D. Lo, "Amalgam+: Composing rich information sources for accurate bug localization," *J. Softw. Evol. Process*, vol. 28, no. 10, pp. 921–942, 2016.

[47] B. Sisman, S. A. Akbar, and A. C. Kak, "Exploiting spatial code proximity and order for improved source code retrieval for bug localization," *J. Softw. Evol. Process*, vol. 29, no. 1, 2017, Art. no. e1805.

[48] S. Khatiwada, M. Tushev, and A. Mahmoud, "Just enough semantics: An information theoretic approach for ir-based software bug localization," *Inf. Softw. Technol.*, vol. 93, pp. 45–57, Jan. 2018.

[49] T. Dilshener, M. Wermelinger, and Y. Yu, "Locating bugs without looking back," *Automated Softw. Eng.*, vol. 25, no. 3, pp. 383–434, 2018.

[50] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proc. 29th ACM/IEEE Int. Conf. Softw. Eng.*, 2007, pp. 499–510.

[51] M. D. Dikaiakos, A. Katsifodimos, and G. Pallis, "Minersoft: Software retrieval in grid and cloud computing infrastructures," *ACM Trans. Internet Technol.*, vol. 12, no. 1, pp. 2:1–2:34, 2012.

[52] G. Petrosyan, M. P. Robillard, and R. D. Mori, "Discovering information explaining API types using text classification," in *Proc. 37th IEEE/ACM Int. Conf. Softw. Eng.*, 2015, pp. 869–879.

[53] G. Uddin and F. Khomh, "Automatic mining of opinions expressed about APIs in stack overflow," *IEEE Trans. Softw. Eng.*, vol. 47, no. 3, pp. 522–559, Mar. 2021.

[54] F. Palomba et al., "Recommending and localizing change requests for mobile apps based on user reviews," in *Proc. 39th ACM/IEEE Int. Conf. Softw. Eng.*, 2017, pp. 106–117.

[55] F. Palomba et al., "Crowdsourcing user reviews to support the evolution of mobile apps," *J. Syst. Softw.*, vol. 137, pp. 143–162, Mar. 2018.

[56] Y. Zhou, Y. Su, T. Chen, Z. Huang, H. C. Gall, and S. Panichella, "User review-based change file localization for mobile applications," *IEEE Trans. Softw. Eng.*, vol. 47, no. 12, pp. 2755–2770, Dec. 2021.

[57] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Trans. Softw. Eng.*, vol. 34, no. 2, pp. 287–300, Mar./Apr. 2008.

[58] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical Softw. Eng.*, vol. 14, no. 1, pp. 5–32, 2009.

[59] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *J. Softw. Evol. Process*, vol. 30, no. 6, 2018, Art. no. e1958.

[60] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Inf. Softw. Technol.*, vol. 49, no. 3, pp. 230–243, 2007.

[61] P. van der Spek and S. Klusener, "Applying a dynamic threshold to improve cluster detection of LSI," *Sci. Comput. Program.*, vol. 76, no. 12, pp. 1261–1274, 2011.

[62] S. Grant, J. R. Cordy, and D. B. Skillicorn, "Using heuristics to estimate an appropriate number of latent topics in source code analysis," *Sci. Comput. Program.*, vol. 78, no. 9, pp. 1663–1678, 2013.

[63] E. Hill, L. L. Pollock, and K. Vijay-Shanker, "Exploring the neighborhood with dora to expedite software maintenance," in *Proc. 22nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2007, pp. 14–23.

[64] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proc. 17th Work. Conf. Reverse Eng.*, 2010, pp. 35–44.

[65] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: An empirical study," *Empirical Softw. Eng.*, vol. 19, no. 5, pp. 1383–1420, 2014.

[66] H. Yu et al., "Automated assertion generation via information retrieval and its integration with deep learning," in *Proc. 44th IEEE/ACM Int. Conf. Softw. Eng.*, 2022, pp. 163–174.

[67] M. Unterkalmsteiner, T. Gorschek, R. Feldt, and N. Lavesson, "Large-scale information retrieval in software engineering - An experience report from industrial application," *Empirical Softw. Eng.*, vol. 21, no. 6, pp. 2324–2365, 2016.

[68] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requiriments Eng.*, vol. 12, no. 2, pp. 103–120, 2007.

[69] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Trans. Softw. Eng.*, vol. 39, no. 1, pp. 18–44, Jan. 2013.

[70] M. Borg, K. Wnuk, B. Regnell, and P. Runeson, "Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context," *IEEE Trans. Softw. Eng.*, vol. 43, no. 7, pp. 675–700, Jul. 2017.

[71] E. Hill et al., "AMAP: Automatically mining abbreviation expansions in programs to enhance software maintenance tools," in *Proc. Int. Work. Conf. Mining Softw. Repositories*, 2008, pp. 79–88.

[72] D. W. Binkley and D. J. Lawrie, "The impact of vocabulary normalization," *J. Softw. Evol. Process*, vol. 27, no. 4, pp. 255–273, 2015.

[73] Y. Jiang, H. Liu, J. Jin, and L. Zhang, "Automated expansion of abbreviations based on semantic relation and transfer expansion," *IEEE Trans. Softw. Eng.*, vol. 48, no. 2, pp. 519–537, Feb. 2022.

[74] A. De Lucia, M. Di Penta, and R. Oliveto, "Improving source code lexicon via traceability and information retrieval," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 205–227, Mar./Apr. 2011.

[75] E. Brill, "A simple rule-based part of speech tagger," in *Proc. Speech Natural Lang.: Proc. Workshop,* Harriman, New York, 1992, pp. 112–116.

[76] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery via noun-based indexing of software artifacts," *J. Softw. Evol. Process*, vol. 25, no. 7, pp. 743–762, 2013.

[77] N. Ali, H. Cai, A. Hamou-Lhadj, and J. Hassine, "Exploiting parts-of-speech for effective automated requirements traceability," *Inf. Softw. Technol.*, vol. 106, pp. 126–141, Feb. 2019.
[78] N. Ali, Z. Sharafi, Y. Guéhéneuc, and G. Antoniol, "An empirical study on the importance of source code entities for requirements traceability," *Empirical Softw. Eng.*, vol. 20, no. 2, pp. 442–478, 2015.
[79] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Improving after-the-fact tracing and mapping: Supporting software quality predictions," *IEEE Softw.*, vol. 22, no. 6, pp. 30–37, Nov./Dec. 2005.
[80] N. Ali, Y. Guéhéneuc, and G. Antoniol, "Requirements traceability for object oriented systems by partitioning source code," in *Proc. 18th Work. Conf. Reverse Eng.*, 2011, pp. 45–54.
[81] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation," *Inf. Softw. Technol.*, vol. 55, no. 4, pp. 741–754, 2013.
[82] M. Eaddy, A. V. Aho, G. Antoniol, and Y. Guéhéneuc, "CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, 2008, pp. 53–62.
[83] N. Ali, Y. Guéhéneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *IEEE Trans. Softw. Eng.*, vol. 39, no. 5, pp. 725–741, 2013.
[84] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia, "Using code ownership to improve ir-based traceability link recovery," in *Proc. 21st IEEE Int. Conf. Program Comprehension*, 2013, pp. 123–132.
[85] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: A silver bullet for traceability recovery," in *Proc. 22nd IEEE Int. Conf. Softw. Maintenance*, 2006, pp. 299–309.
[86] H. Gao et al., "Propagating frugal user feedback through closeness of code dependencies to improve IR-based traceability recovery," *Empirical Softw. Eng.*, vol. 27, no. 2, 2022, Art. no. 41.
[87] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. De Lucia, "Predicting query quality for applications of text retrieval to software engineering tasks," *ACM Trans. Softw. Eng. Methodol.*, vol. 26, no. 1, pp. 3:1–3:45, 2017.
[88] M. M. Rahman and C. K. Roy, "A systematic review of automated query reformulations in source code search," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 6, pp. 159:1–159:79, 2023.
[89] A. D. Rodriguez, J. Cleland-Huang, and D. Falessi, "Leveraging intermediate artifacts to improve automated trace link retrieval," in *Proc. 37th IEEE Int. Conf. Softw. Maintenance Evol.*, 2021, pp. 81–92.
[90] C. Duan and J. Cleland-Huang, "Clustering support for automated tracing," in *Proc. 22nd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2007, pp. 244–253.
[91] H. Sultanov, J. H. Hayes, and W. Kong, "Application of swarm techniques to requirements tracing," *Requir. Eng.*, vol. 16, no. 3, pp. 209–226, 2011, doi: 10.1007/s00766-011-0121-4.
[92] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *Proc. 39th ACM/IEEE Int. Conf. Softw. Eng.*, 2017, pp. 3–14.
[93] K. Moran et al., "Improving the effectiveness of traceability link recovery using hierarchical Bayesian networks," in *Proc. 42nd ACM/IEEE Int. Conf. Softw. Eng.*, 2020, pp. 873–885.
[94] X. Chen, J. G. Hosking, J. C. Grundy, and R. Amor, "DCTracVis: A system retrieving and visualizing traceability links between source code and documentation," *Automated Softw. Eng.*, vol. 25, no. 4, pp. 703–741, 2018.
[95] H. Ruan, B. Chen, X. Peng, and W. Zhao, "Deeplink: Recovering issue-commit links based on deep learning," *J. Syst. Softw.*, vol. 158, 2019, Art. no. 110406.
[96] J. Lin, Y. Liu, and J. Cleland-Huang, "Information retrieval versus deep learning approaches for generating traceability links in bilingual projects," *Empirical Softw. Eng.*, vol. 27, no. 1, p. 5, 2022.
[97] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms," in *Proc. 35th ACM/IEEE Int. Conf. Softw. Eng.*, 2013, pp. 522–531.
[98] S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang, "Improving trace accuracy through data-driven configuration and composition of tracing features," in *Proc. Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2013, pp. 378–388.
[99] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Parameterizing and assembling IR-based solutions for SE tasks using genetic algorithms," in *Proc. 23rd IEEE Int. Conf. Softw. Anal., Evol., Reeng.*, 2016, pp. 314–325.
[100] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova, "Best practices for automated traceability," *Comput.*, vol. 40, no. 6, pp. 27–35, 2007.
[101] A. De Lucia, R. Oliveto, and G. Tortora, "Assessing IR-based traceability recovery tools through controlled experiments," *Empirical Softw. Eng.*, vol. 14, no. 1, pp. 57–92, 2009.
[102] J. H. Hayes, A. Dekhtyar, J. Larsen, and Y. Guéhéneuc, "Effective use of analysts' effort in automated tracing," *Requirements Eng.*, vol. 23, no. 1, pp. 119–143, 2018.
[103] A. De Lucia, R. Oliveto, and G. Tortora, "IR-based traceability recovery processes: An empirical comparison of "one-shot" and incremental processes," in *Proc. 23rd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2008, pp. 39–48.
[104] G. Bavota, A. De Lucia, R. Oliveto, and G. Tortora, "Enhancing software artefact traceability recovery processes with link count information," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 163–182, 2014.
[105] D. Falessi, M. Di Penta, G. Canfora, and G. Cantone, "Estimating the number of remaining links in traceability recovery," *Empirical Softw. Eng.*, vol. 22, no. 3, pp. 996–1027, 2017.
[106] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and Systems Traceability*. London: Springer, 2012.
[107] Y. Liu, J. Lin, O. Anuyah, R. A. Metoyer, and J. Cleland-Huang, "Generating and visualizing trace link explanations," in *Proc. 44th IEEE/ACM Int. Conf. Softw. Eng.*, 2022, pp. 1033–1044.
[108] H. Jiang, T. N. Nguyen, I. Chen, H. Jaygarl, and C. K. Chang, "Incremental latent semantic indexing for automatic traceability link evolution management," in *Proc. 23rd IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2008, pp. 59–68.
[109] M. Rahimi and J. Cleland-Huang, "Evolving software trace links between requirements and source code," *Empirical Softw. Eng.*, vol. 23, no. 4, pp. 2198–2231, 2018.
[110] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *Proc. 31st IEEE Int. Requirements Eng. Conf.*, 2023, pp. 455–464.
[111] B. Li et al., "Trustworthy AI: From principles to practices," *ACM Comput. Surveys*, vol. 55, no. 9, pp. 177:1–177:46, 2023.
[112] "The tailor handbook of trustworthy AI," 2022. [Online]. Available: http://tailor.isti.cnr.it/handbookTAI/Accountability/L2.Traceability.html
[113] "Securing artificial intelligence (SAI): Traceability of AI models," *DTR/SAI-004, European Telecommunications Standards Institute (ETSI)*, no. TR 104 032 V1.1.1, 2024.
[114] S. Longpre et al., "Position: Data authenticity, consent, and provenance for AI are all broken: What will it take to fix them?" in *Proc. 41st Int. Conf. Mach. Learn.*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2404.12691
[115] "Mit: Data provenance for AI," 2024. [Online]. Available: https://www.media.mit.edu/projects/data-provenance-for-ai/overview
[116] N. Rodríguez-Barroso et al., "Federated learning and differential privacy: Software tools analysis, the sherpa.AI FL framework and methodological guidelines for preserving data privacy," *Inf. Fusion*, vol. 64, pp. 270–292, 2020.
[117] A. Marengo, "Navigating the nexus of AI and IoT: A comprehensive review of data analytics and privacy paradigms," *Internet Things*, vol. 27, pp. 101318:1–101318:22, 2024.
[118] G. Sun, Y. Cong, J. Dong, Q. Wang, L. Lyu, and J. Liu, "Data poisoning attacks on federated machine learning," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 11365–11375, 2022.
[119] M. A. Ramirez et al., "Poisoning attacks and defenses on artificial intelligence: A survey," 2022, *arXiv:2202.10276*.
[120] X. Fan, D. Fu, H. Gui, X. Zhang, and X. Zhou, "PCPT and ACPT: Copyright protection and traceability scheme for DNN models," 2023, *arXiv:2206.02541*.
[121] S. Wang, C. Xu, Y. Zheng, and C.-H. Chang, "A buyer-traceable DNN model IP protection method against piracy and misappropriation," in *Proc. 4th IEEE Int. Conf. Artif. Intell. Circuits Syst.*, 2022, pp. 308–311.
[122] L. Jia, H. Zhong, X. Wang, L. Huang, and X. Lu, "The symptoms, causes, and repairs of bugs inside a deep learning library," *J. Syst. Softw.*, vol. 177, pp. 110935:1–110935:14, 2021.
[123] M. M. Morovati, A. Nikanjam, F. Khomh, and Z. M. J. Jiang, "Bugs in machine learning-based systems: A faultload

benchmark," *Empirical Softw. Eng.*, vol. 29, no. 14, pp. 62:1–62:33, 2023.
[124] R. Dwivedi et al., "Explainable AI (XAI): Core ideas, techniques, and solutions," *ACM Comput. Surveys*, vol. 55, no. 9, pp. 194:1–194:33, 2023.
[125] A. Rawal, J. McCoy, D. B. Rawat, B. M. Sadler, and R. S. Amant, "Recent advances in trustworthy explainable artificial intelligence: Status, challenges, and perspectives," *IEEE Trans. Artif. Intell.*, vol. 3, no. 6, pp. 852–866, Dec. 2022.
[126] J. Gesi, X. Shen, Y. Geng, Q. Chen, and I. Ahmed, "Leveraging feature Bias for scalable misprediction explanation of machine learning models," in *Proc. 45th IEEE/ACM Int. Conf. Softw. Eng.*, 2023, pp. 1559–1570.
[127] S. Caton and C. Haas, "Fairness in machine learning: A survey," *ACM Comput. Surveys*, vol. 56, no. 7, pp. 166:1–166:38, 2024.
[128] Z. Chen, J. M. Zhang, M. Hort, M. Harman, and F. Sarro, "Fairness testing: A comprehensive survey and analysis of trends," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, pp. 137:1–137:59, 2024.
[129] C. Ferrara, G. Sellitto, F. Ferrucci, F. Palomba, and A. De Lucia, "Fairness-aware machine learning engineering: How far are we?" *Empirical Softw. Eng.*, vol. 29, no. 1, pp. 9:1–9:46, 2023.
[130] T. L. Quy, A. Roy, V. Iosifidis, W. Zhang, and E. Ntoutsi, "A survey on datasets for fairness-aware machine learning," *Wiley Interdisciplinary Rev.: Data Mining Knowl. Discovery*, vol. 12, no. 3, 2022, Art. no. e1452.

**Giuliano (Giulio) Antoniol** is an Aggregate Professor with the Polytechnique Montreal. From 2005 to 2019, he directed the Canada Research Chair Tier I in Software Change and Evolution. He has participated in the program and organization committees of numerous IEEE-sponsored international conferences. He served as the Program Chair, the Industrial Chair, a Tutorial, and the General Chair of international conferences and workshops. He contributed to the program committees of more than 30 IEEE and ACM conferences and workshops and acted as a referee for all major software engineering journals. He spent 18 years at the Ecole Polytechnique de Montreal, working in the areas of software evolution, software traceability, search-based software engineering, software testing, and software maintenance.

**Gerardo Canfora** is a Software Engineering Professor and a Rector with the University of Sannio, Benevento, Italy. He is old enough to have seen several waves of technology disappear, leaving more problems than they have been trying to solve, but this has not diminished his enthusiasm for researching into new solutions to engineer better software.

**Gerardo Casazza** is an IT Architect with DENSO Europe International, where he strives for the right balance between legacy environments and emerging technologies in a complex and continuously changing context.

**Andrea De Lucia** is a Professor in software engineering with the University of Salerno, Italy. He has co-authored more than 300 papers published in international journals, books, and conference proceedings. His research maintenance, software testing, empirical software engineering, software engineering for AI-based systems, and quantum software engineering. He serves on the program and organizing committees of several international conferences and on the editorial board of international journals in the field of software engineering. He is a Co-Editor-in-Chief of Science of Computer Programming and was a Member-at-Large of the executive committee of the IEEE Technical Council on Software Engineering.

**Ettore Merlo** has been a Professor with the Department of Computer and Software Engineering, Polytechnique Montreal, since 1993. His research interests and experience are in software engineering, software analysis, application security, software testing, reengineering, and artificial intelligence. More specific research interests include automated roles and privileges security vulnerability analysis, large scale software similarity analysis, and model-driven analysis of swarm robotics applications. His interests target AI robustness, fairness, bias, and ethics.