


Software assistants in software engineering: A systematic mapping study

Maxime Savary-Leblanc¹  | Lola Burgueño^{2,3} | Jordi Cabot^{2,4} | Xavier Le Pallec¹ | Sébastien Gérard⁵

¹UMR 9189 CRIStAL, Univ. Lille, CNRS, Inria, Centrale Lille, Lille, France

²SOM Research Lab, Open University of Catalonia, Barcelona, Spain

³Department of Computer Science and Programming Languages, University of Malaga, Malaga, Spain

⁴Internet Interdisciplinary Institute, ICREA, Barcelona, Spain

⁵Université Paris-Saclay, CEA List, Palaiseau, France

Correspondence

Maxime Savary-Leblanc, UMR 9189 CRIStAL, Univ. Lille, CNRS, Inria, Centrale Lille, F-59000 Lille, France.
Email:
maxime.savary-leblanc@univ-lille.fr

Funding information

Spanish Government, Grant/Award Numbers: PID2020-114615RB-I00, PID2021-125527NB-I00; ECSEL Joint Undertaking (JU), Grant/Award Numbers: 101007260, 101007350

Abstract

The increasing essential complexity of software systems makes current software engineering methods and practices fall short in many occasions. Software assistants have the ability to help humans achieve a variety of tasks, including the development of software. Such assistants, which show human-like competences such as autonomy and intelligence, help software engineers do their job by empowering them with new knowledge. This article investigates the research efforts that have been conducted on the creation of assistants for software design, construction and maintenance paying special attention to the user-assistant interactions. To this end, we followed the standard systematic mapping study method to identify and classify relevant works in the state of the art. Out of the 7580 articles resulting from the automatic search, we identified 112 primary studies that present works which qualify as software assistants. We provide all the resources needed to reproduce our study. We report on the trends and goals of the assistants, the tasks they perform, how they interact with users, the technologies and mechanisms they exploit to embed intelligence and provide knowledge, and their level of automation. We propose a classification of software assistants based on interactions and present an analysis of the different automation patterns. As outcomes of our study, we provide a classification of software assistants dealing with the design, construction and maintenance phases of software development, we discuss the results, identify open lines of work and challenges and call for new innovative and rigorous research efforts in this field.

KEYWORDS

software assistants, software construction, software design, software maintenance, systematic mapping study

Abbreviations: IDE, Integrated Development Environment; SE, Software Engineering; IPSE, Integrated Programming Support Environment; SEE, Software Engineering Environnement; CASE, Computer-Aided Software Engineering; CSCW, Computer-Supported Cooperative Work; DIKW, Data Information Knowledge Wisdom; RSSE, Recommender System for Software Engineering; MDE, Model-Driven Engineering; HCI, Human-Computer Interaction; ML, Machine Learning; RS, Recommender System; VCS, Version Control System; DSML, Domain-Specific Modeling Language.

This is an open access article under the terms of the [Creative Commons Attribution-NonCommercial-NoDerivs](https://creativecommons.org/licenses/by-nc-nd/4.0/) License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

© 2022 The Authors. *Software: Practice and Experience* published by John Wiley & Sons Ltd.

1 | INTRODUCTION

Essential complexity of software systems keeps increasing as new software projects need to embrace a growing number of technologies and domains.^{1,2} Two recent examples would be the Internet of Things—forcing software systems to deal with the management and run-time variability of hardware components embedded in the system—and artificial intelligence—pushing software to integrate more and more *smart* features. As such, current software engineering methods and practices struggle to catch up.

In particular, this problem occurs during the software design and construction phases. Its relevance is undeniable since both phases together account, on average, for more than 50% of the effort spent over the entire software development cycle³ and cover the main activities of software development.⁴ Thus, since decades ago, there has been an interest in providing any help developers can use to optimize and accelerate their work (while keeping the same quality standards) is valuable.^{5,6}

For instance, software engineers can resort to information available online, including technical knowledge of a variety of languages and algorithms together with source code examples and libraries for reuse. A number of question and answer websites specific to the software development problem (like StackOverflow) or code search platforms are populated and curated in a community effort.⁷

While this is helpful, it can be complemented with a proactive action and an effort to filter, comprehend, adapt and apply all the information available to both prevent the risk of ending up in a situation of information overload and to save time. It is worth noting that *being stuck in problem solving* is identified as the number one cause of developers' unhappiness, causing low productivity, low motivation and low quality of the produced artifacts.^{8,9} Although this main cause is mostly related to a knowledge problem, *time pressure*, *bad code quality and coding practice*, and *repetitive tasks* are also among the top-10 causes of developers' unhappiness. Unlike the former cause, the latter problems are not due to conceptual difficulties or lack of technical knowledge but are related to developers' performance and speed.

To overcome many of these problems, Integrated Development Environments (IDEs) have progressively implemented numerous *software tools* to simplify many tasks central to the software design and construction processes. Functions like code reformatting, copy and paste or code refactoring have made their way into developers' work process, to such a point that it seems hardly conceivable to remove them from the IDEs now. While these tools are clearly an improvement, they still fall short.¹⁰ For instance, the decision of what tool to use is still driven by the engineer, who must have enough knowledge to understand the problem at hand, imagine potential solutions, select the most suitable one for the context, and apply it with respect to the language and the architecture of the project.

As such, we are starting to see a new generation of more capable systems to help in software engineering called, in this article, *software assistants*. Although we are not aware of any universal definition of software assistant, these are expected to help users complete their tasks showing some degree of human-like competencies such as autonomy, decision-making, technical knowledge and social abilities.

Given the importance and potential impact of this new breed of support systems for software engineers, in this article, we have performed a systematic mapping study^{11,12} that aims to identify and classify relevant works. As a result, our work will provide a comprehensive view of the field by characterizing the goals, types, methods, interaction patterns, underlying technologies and overall characteristics of software assistants presented so far in the literature. Moreover, we also discuss the limitations of current software assistants and potential lines of further work that could make them even more useful and a key contribution to successfully complete large and complex software projects.

We believe our study could be useful to software practitioners looking to improve the productivity and quality of their work, by identifying new software to install and test. It may also help software researchers trying to understand the current trends in this field and the open challenges that remain to be solved. Tool vendors could also exploit this work to identify new potential assistants to integrate in the coming versions of their IDEs.

The rest of this article is structured as follows. Section 2 provides background, defines software assistants and introduces related works. Section 3 introduces the research questions and the protocol of our systematic mapping study. Section 4 presents the results of the data extraction and analysis and the classification of software assistants. Section 5 identifies the potential limitations and threats to validity of this work. Section 6 discusses the results and identifies research challenges and open lines based on the state of the art. Finally, Section 7 concludes on the general results of this review.

2 | BACKGROUND AND RELATED WORKS

The evolution of software engineering systems has always been tightly related to the advances of software technologies and user-understanding. In this section, we present the basis upon which this article builds.

2.1 | Supporting software engineering

Tools for programmers naturally exist since the beginning of software engineering around 1960. At that time, they were single tools focused on some specific tasks of the SE life cycle, cumbersome to use, and acting in isolation of each other.¹³ Around 1980, the increasing complexity of the solutions to be produced as well as the better understanding of the users' needs drive the improvement of the existing SE instrumentation.¹⁴ A new wave of systems then gradually replaces tools with more comprehensive functionalities gathered in *environments*, such as Integrated Programming Support Environments (IPSE) and later software engineering environments (SEE). These systems fall under the emerging field of computer-aided software engineering (CASE) tools, which lay the foundation for modern-day IDEs. As environments improve, other issues emerge such as the need for collaboration to produce ever more complex systems, which paves the way for the computer-supported cooperative work (CSCW) community and more specifically the collaborative software engineering (CSE) community.¹⁵ The CSE community then seeks to enhance environments to cope with different forms of collaboration.

During the 90s, the *agent* research fields explodes and brings to light a new opportunity for collaboration: that with the machine acting as an autonomous system with which users (or other agents) could interact and work.¹⁶ Some agents are refined into *intelligent agents* that are reactive, proactive, and social agents tailored for human-agent collaboration,¹⁷ and applied to support software engineering processes.^{18,19} However, due the lack of computing resources and/or data to exploit, such agent-based systems never became mainstream in software engineering.²⁰

Since, the broad Software Agent community has remained active, and has branched into several subcategories. Particularly, the notion of *conversational agent* (a.k.a. bot or chatbot—coined by Michael Mauldin in 1994) is gaining importance in the last years, and has quickly become a must-have, especially in the sectors of customer support or video game.^{20,21} In 2016, Storey and Zagalsky²² laid the foundation for research on bots in software engineering and described how bots are increasingly used to support tasks that traditionally required human intelligence. It has particularly been applied to software engineering to create *BOTse*²³ or *DevBots*²⁰ (bots for software engineering).²⁴ A consensual definition established during the BOTse Dagstuhl seminar in 2020²³ defines bots as systems featuring at least one of the following characteristics: (i) automates one or more feature(s), (ii) performs one or more function(s) that a human may do, (iii) interacts with a human or other agents.

At ICSE'06, Boehm predicted a new kind of developer-helping systems for 2020 as “that provide feedback to developers based on domain knowledge, programming knowledge, systems engineering knowledge, or management knowledge.”¹⁴ The description of previous bot systems is almost inline with these expectations but still lacks one essential characteristic that Boehm described as “the use of knowledge.” Storey and Zagalsky²² identify bots embedding knowledge as one specific type of bots. Thus, knowledge appear as an inflexion point, which opens the way for the study of a specific type of systems—knowledge-empowered DevBots—that we will call software assistants for software engineering.

2.2 | Knowledge provided by software systems

Knowledge appeared in the 80s in the scope of Software with knowledge-based systems.²⁵ However, it is with the increasing amount of available storage and computation resources that it appeared in the 2000s as a revolution for digital systems, involving fundamental changes in the way people relate to their own knowledge.²⁶ It is a broad term which encapsulates different notions and which has no consensual definition.²⁷ Nevertheless, it is commonly admitted that data can lead to information which, in turn, can lead to knowledge, based on the DIKW hierarchy.^{28,29} In the scope of software systems, we adopt the definition of knowledge as (i) the result of the analysis of structured information, (ii) related to the current context, problem, or activity, (iii) and tailored to the user's needs.²⁹⁻³¹

Based on the highly influential description of the DIKW hierarchy of Rowley,³⁰ we provide the following description of data, information and knowledge in the frame of software engineering:

- *Data* is the content of the considered software artifacts.
- *Information* is a fact about one or more artifacts, resulting from their simple reading.
- *Knowledge* is the result of the analysis and combination of information, valuable in the scope of the current problem, in the current context.

Thus, the notion of knowledge only makes sense when linked to a specific task or problem. Let us illustrate these concepts with an example. John codes a Java program and launches the execution in his IDE. An error occurs about a graphical element that John coded, and the error message is displayed in the console. Then John wants to understand what portion of the code causes the error. In this context, data is represented by the all the files containing Java code as well as the error message displayed in the console. One information could be that 35 Java files contain references to the graphical element (as all information, this is not context-dependent). Then, one knowledge would be that the file causing the error might be among a shortlist of three recommended files.

2.3 | Software assistants for software engineering

Software Engineers exploit knowledge in many software engineering activities,³² on a wide range of software and/or domain specific topics.³³ This calls for a new wave of systems that put knowledge at the heart of their logic and interactions, made possible by the current state of data and technologies. To describe these systems, we adopt the definition of software bots presented in Section 2.1, and define *software assistants for software engineering* as software bots which provide users with valuable knowledge to help them identify, understand, or solve a problem. The notion of knowledge refers to the definition introduced in Section 2.2. For the sake of clarity in the rest of this article, we will refer to software assistants for software engineering with the shorter version software assistants.

In some previous works, software assistants may also be referred as Intelligent Assistant,³⁴ IA-based digital Assistant,³⁵ Intelligent User Assistance System,³⁶ Virtual Assistant,³⁷ Smart Assistant,³⁸ Intelligent Agents,³⁹ or shortly Bots.²³ While the description of these systems seems to converge on the notion of assistant, some also involve the notion of Intelligence. As knowledge appears to be only one component of what constitutes intelligence,^{40,41} we refrain from qualifying software assistants of intelligent or smart. However, artificial intelligence techniques, such as machine learning or ontologies, might be embedded in software assistants to create, organize, or filter the knowledge that is required.

Software assistants may help users make a decision and eventually perform a task according to this decision with a certain degree of autonomy. Their outcomes might not be deterministic, as they adapt to each problem and context. They might be used to automate manual tasks to save time and reduce effort, but they must be able to come up with new information and ideas and that may be valuable to increase the knowledge of the user. Software assistants consist in complete and ready-to-use software systems, accessible through a user interface (to be able to provide users with valuable knowledge). Therefore, single components and algorithms are not considered as software assistants.

Although there is a wide variety of bots and software tools embedded into IDEs that perform automated tasks and are used during the software development process (e.g., refactoring, search, and indentation tools), it is worth noting that software assistants are still not as mainstream as these. This motivates our effort to build a comprehensive view of the literature about software assistants, in the hope of fostering research in potential directions and identifying challenges.

2.4 | Previous secondary studies on software assistants in SE

This section positions our work with respect to systematic literature reviews, mapping studies and surveys on topics related to software assistants and software engineering.

To the best of our knowledge, there is no study with the same focus as ours. On the contrary, the available studies either focus on one specific kind of assistant-related approach and study its application in the software engineering life cycle, or focus on one of its stages and identify the assistant-related approaches and systems. Let us list and describe them below.

Some related works aim to investigate the use of one specific approach during the various stages of software engineering. For instance, Gasparic and Janes⁴² conducted a systematic literature review on recommender systems for software

engineering (RSSE). They identified 47 implemented systems in papers published before 2013, and analyzed their inputs, their outputs, the effort required from engineers, and the benefits they provide to their users. Their results showed that RSSE mainly support reuse, debugging, implementation, and maintenance phases/activities, which we also cover in this study (see our inclusion criteria in Section 3.2). They found that most RSSE mainly recommend source code, and only some of them digital documents. Our work differs from this one in several aspects. First, the RSSE that the authors has studied qualify as one of the types of software assistant that we consider in this systematic mapping study—as long as they present at least a fully implemented prototype of the assistant. Second, our research questions are broader. For instance, we elaborate on aspects such as the nature and the environment of the software assistant which both condition how the knowledge should be presented. Finally, our inclusion time frame from 2010 to 2022 updates the overview of the trends of RSSE.

Almonte et al. conducted a systematic mapping review⁴³ about recommender systems targeting the model-driven engineering (MDE) paradigm. As the authors explain, they excluded recommender systems for activities not related to models or modeling. Unlike it, our study covers the design and construction phases of software development regardless the paradigm developers follow. This is, our study is not restricted to MDE, but includes works that propose assistants for any task or approach (e.g., TDD, BDD, etc.) and therefore, do not only deal with models but other software artifacts. On the other hand, the scope of our study is restricted to assistants for the design and construction phases, leaving out phases such as requirement elicitation, testing or maintenance. Furthermore, Almonte et al. study which tasks are subject to recommendations, the applicable recommendation techniques and how recommender systems are evaluated. While we also study on which tasks assistants assist users and the technologies that are used, in addition to these, we pay especial attention to HCI indicators and to what extent assistants are automated.

Savchenko et al. carried out a systematic mapping study⁴⁴ about smart tools in software engineering with the goal to answer the question “how could the technological innovations affect the software development ecosystems and software processes?”, and studied the state of the art between 2015 and 2019. While the authors try to disclose the impact of software innovations in businesses, our work puts the emphasis on the software assistants themselves (i.e., how they are built, with which tasks they help engineers, how users interact with software assistants, etc.).

Different studies also investigated the way machine learning (ML) has been applied in software engineering. Borges et al.⁴⁵ collected 177 studies from 1992 to 2019 in two groups: (i) software quality and software engineering management (40 papers) and (ii) software quality and software test (15 papers). They conclude that software quality is the most frequent SWEBOK knowledge area target for both clusters (51%) and that ML is mainly used to make predictions in SE. A similar study by Shafiq et al.⁴⁶ shows that based on 227 articles about ML for software engineering from 1991 to 2019, 21 were focusing on requirements, 39 on architecture and design, 21 on implementation, 119 on quality assurance and analytics and 9 on maintenance. Our study is richer and broader in the sense that we do not only focus on assistants empowered with ML techniques, and more specific in the sense that we are only interested in working assistants and ignore theoretical and unimplemented approaches.

Other mapping studies focused on the use of tools during one specific task of software engineering. For instance, Jung et al.⁴⁷ reported on the tools to enable domain-specific language development. Sebastian et al.⁴⁸ investigated code generation using model-driven architecture, and identified implemented systems enabling code generation. Similarly, Brunschwig et al.⁴⁹ provided an overview of tools to support software modeling on mobile devices.

To the best of our knowledge, there is no previous systematic mapping study on understanding and classifying implemented and ready-to-use software assistants to support software development tasks without restricting the techniques used to achieve this goal. With this work, we aim to fill the gap and summarize implemented software assistants for software engineering systems that were presented in scientific venues.

3 | RESEARCH METHOD

This study follows the guidelines proposed by Petersen et al.¹¹ to conduct systematic mapping studies (SMS). A pilot study was conducted on a small number of articles to assess the suitability of the criteria and the method, which led to discussions and updates on the protocol. This section describes the different steps of the exploration phase as defined in the final version of the protocol.

3.1 | Research questions

In this study, our goal is to study the work done on software assistants for software engineering. To do so, we examine what has been done, what can be considered as recurrent practice, and what has not yet been explored. Starting from the definition of software assistant provided in Section 2:

Software bots which provide users with valuable knowledge to help them identify, understand, or solve a problem.

(i)

(ii)

(iii)

(iv)

We have divided and analyzed each part of it and identify RQs that will help us understand the role of software assistants. First of all, in RQ1, we focus on (iii) and wonder how software assistants help practitioners with software engineering activities, that is, whether it is the general category of assistance (debugging, modeling, refactoring, etc.) or the technical environment. In RQ2, we focus on (iv) and the identification, understanding, or solving of a problem. We review the actions that software assistants are able to perform and the types of practitioner-SA interaction. As software assistants are bots (i), in RQ3, we assess the automation levels they propose and what they need to ask to the practitioner. Finally, in RQ4, we investigate (ii), and in particular, how assistants integrate valuable knowledge. We establish a list of the datasources they exploit.

We thus formulate four research questions that cover these different points:

- *RQ1: What are the tasks that software assistants help users achieve, in which environments do they operate and which languages do they support?* With this research question, we aim to identify which parts of the software design, construction and maintenance are best supported by assistants as well as trends in the supported environments and languages.
- *RQ2: How do software assistants assist users?* This question aims to identify the different types of assistants (as perceived by their users) and the way they present information to the user.
- *RQ3: What kind of software technologies are used to embed knowledge in software assistants?* This question investigates how software assistants are equipped with knowledge to support users. This includes analyzing their data usage as well as finding whether they exploit machine learning (ML) techniques.
- *RQ4: To what extent are software assistants automated?* To answer this question, we assessed the levels of automation and the trigger of each software assistant in our set of primary studies.

3.2 | Inclusion and exclusion criteria

The inclusion criteria define the scope of our systematic mapping study and enable us to identify papers which will help build answers to the research questions. We have focus on those papers that:

1. Are written in English, peer-reviewed, and published between January 2010 and January 2022.*
2. Focus on at least one of the these knowledge areas from the SWEBOK:⁵⁰
 - *Software design subject*, which relates to creating and checking software designs.
 - *Software construction subject*, which includes program editors, compilers and code generators, interpreters, and debuggers.
 - *Software maintenance subject*, which covers artifacts visualization and re-engineering.
 - *Socio-cultural systems for software engineering*, which cover socio-cultural aspects such as social networking as identified in Reference 51.
3. Focus on the assistance for the professional software practitioners involved in the four processes in the previous item.

*Motivated by the background provided in Section 2, for example, the recent emergence of bots, and knowledge-engineering technologies. Software assistants were expected for 2020, so we study the last 12 years to evaluate their evolution.

TABLE 1 Selected conferences

Acronym	Name
AAMAS	International Conference on Autonomous Agents and Multi-Agent Systems
ASE	International Conference on Automated Software Engineering
AVI	International Conference on Advanced Visual Interfaces
CAiSE	International Conference on Advanced Information Systems Engineering
CHI	Conference on Human Factors in Computing Systems
CSCWD	International Conference on Computer Supported Cooperative Work in Design
ECOOP	European Conference on Object-Oriented Programming
ER	International Conference on Conceptual Modeling
ESEC/FSE	Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering
ICIC	International Conference on Intelligent Computing
ICSE	International Conference on Software Engineering
IEA/AIE	International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems
IJCAI	International Joint Conference on Artificial Intelligence
IS	International Conference on Intelligent Systems
IUI	International Conference on Intelligent User Interfaces
MODELS	International Conference on Model Driven Engineering Languages and Systems
OOPSLA	Object-Oriented Programming, Systems, Languages and Applications
PETRA	Pervasive Technologies Related to Assistive Environments
SAC	Symposium on Applied Computing
TOOLS	Technology of Object-Oriented Languages and Systems
WWW	The Web Conference (e.g., World Wide Web)

Exclusion criteria enabled us to filter out irrelevant papers. Papers featuring one of the following characteristics were excluded:

- Does not provide assistance for at least one of the considered software engineering tasks.
- Does not introduce an implemented and ready to use software assistant—that is, we discarded papers that only introduce a new algorithm or technique that according to the authors *could* be integrated into a software assistant but it does not provide a software assistant in itself.
- Does not claim to have, describe or provide screenshots of the user interface.
- Is a survey, a systematic literature review, or a mapping study.

3.3 | Search process and paper selection

The search process was conducted automatically by querying the *dblp computer science bibliography*[†] website through its API with a custom script. Our algorithm searched for each keyword in Table 3 present in the title of papers that belong to any of the conference proceedings of Table 1 and journals of Table 2. The keyword was built to cover the notion of *assistance* and its synonyms for software engineering. This resulted in 676 queries (52 venues × 13 keywords) to the API, formatted as follows: **KEYWORD**+venue:**VENUE**:. An index of venue codes for the API is available online on our website.⁵²

[†]<https://dblp.org>

TABLE 2 Selected journals

Editor	Name
Springer	Automated Software Engineering
ACM	Communications of the ACM
Wiley	Computational Intelligence
PeerJ	Computer Science
Elsevier	Data and Knowledge Engineering
Springer	Data Science and Engineering
Elsevier	Decision Support Systems
Elsevier	Electronic Notes in Theoretical Computer Science
Springer	Empirical Software Engineering
Elsevier	Expert Systems with Applications
Taylor & Francis	Human–Computer Interaction
Elsevier	Information and Software Technology
Elsevier	Information Processing and Management
Elsevier	Information Sciences
Elsevier	Information Systems
IEEE	Intelligent Systems
Taylor & Francis	International Journal of Computational Intelligence Systems
World Scientific	International Journal on Artificial Intelligence Tools
Springer	Journal of Intelligent Information Systems
Elsevier	Journal of Systems and Software
Elsevier	Knowledge-Based Systems
Elsevier	Science of Computer Programming
Springer	Software Quality Journal
Springer	Software and Systems Modeling
IEEE	Software
ACM	Transactions on Information Systems
ACM	Transactions on Intelligent Systems and Technology
IEEE	Transactions on Knowledge and Data Engineering
ACM	Transactions on Software Engineering and Methodology
IEEE	Transactions on Software Engineering
Springer	User Modeling and User-Adapted Interaction

The venues had been collectively selected by the five authors, taking into account conferences and journals based on their peer-reviewing process and their themes about software engineering, software assistance, or both. Keywords had been chosen in a similar manner to convey the notion of assistance. Some of the keywords take advantage of the default completion feature provided by the dblp's API. For instance, the use of *facilitat* enabled us to retrieve titles containing words like *facilitate*, *facilitating*, or *facilitator*.

The automated search was performed on the 21st of January 2022 on the dblp's API and retrieved 7580 items. At the time of the query, we checked that each venue listed in Tables 1 and 2 was indexed and there were records for the last ten years.

TABLE 3 Search keywords

Search keywords

assist, recommend, help, facilitat, enhanc, answer, empower, augment, aid, suggest, repair, fix, support

TABLE 4 Pruning keywords

Pruning keywords

systematic literature review, mapping study, survey, elderly, health, medical, autism, clinical, disease, home, impairment, older, living, assistive, deaf, colorblind, disabilities, medication

We observed that the resulting list of papers contained a substantial number of articles from socio-medical disciplines. To avoid these, we identified the set of keywords frequently used in socio-medical disciplines and not in software engineering. These are captured in Table 4, which also includes keywords derived from the exclusion criteria mentioned in Section 3.2. We used these keywords to prune the list of papers by automatically discarding those whose titles contained any of these keywords. This automatic pruning phase removed 1543 articles and left 6037 to be processed manually by the authors.

As part of our protocol, all the authors defined and agreed on the fact that papers should be discarded or kept according to the decision diagram presented in Figure 1.

Before manually checking all the papers, and in order to ensure that the criteria was understood equally by the authors and did not lead to subjective interpretations, two authors used the defined criteria and, supported by the algorithm in Figure 1, they worked independently to perform the exclusion–inclusion phase on a subset of 5.6% of all these 6037 articles, that is, on 338 papers. Then, we computed the inter-rater reliability (IRR) using a two-way mixed, single score ICC(A, 1) check for absolute agreement.

The first time, the result was not as good as expected, showing that the co-authors had different views of some aspects defined in the criteria. This criteria was refined, validated with the rest of the authors and the inclusion-exclusion phase was performed again on a different subset of 388 papers independently. Then, we obtained $ICC(A, 1) = 0.723$ (95% CI: $0.664 < ICC(A, 1) < 0.773$) which indicates a moderate to good reliability,⁵³ and therefore the same understanding of the criteria defined (the agreement is 99.3% between the two raters).

Finally, one of these two authors performed the exclusion of the rest of the papers, applying the algorithm of Figure 1 to the reading of the title, the abstract and finally the full paper. It resulted on the exclusion of 5559 papers based on their title, 244 papers based on their abstract, and 180 papers based on their full content.

3.4 | Snowballing

Those papers that passed both inclusion and exclusion criteria were exploited during the snowballing step. The stage consisted in reading the articles, especially their related work section, to identify and add to our corpus candidate software assistants that had either not been obtained during our search or that were discarded by mistake in the exclusion process. In the meantime, the snowballing phase mitigated the threat to validity concerning the absence of a conference or a journal in the initial venues list (Tables 1 and 2) as it gathered new papers regardless of their origin. Each newly selected article was manually tested against inclusion and exclusion criteria, to maintain the consistency of our corpus.

We performed the snowballing phase twice. Once on the original set of papers, and then again on the papers discovered during the first snowballing phase. We did not carried out a third snowballing phase because we observed a large number of articles already overlapping previously identified articles.

These two snowballing steps lead to the gathering of 35 and 23 new articles respectively, increasing the final number of retained articles to 112, to which we will refer as *primary studies*. Figure 2 summarizes the whole search and assessment process.

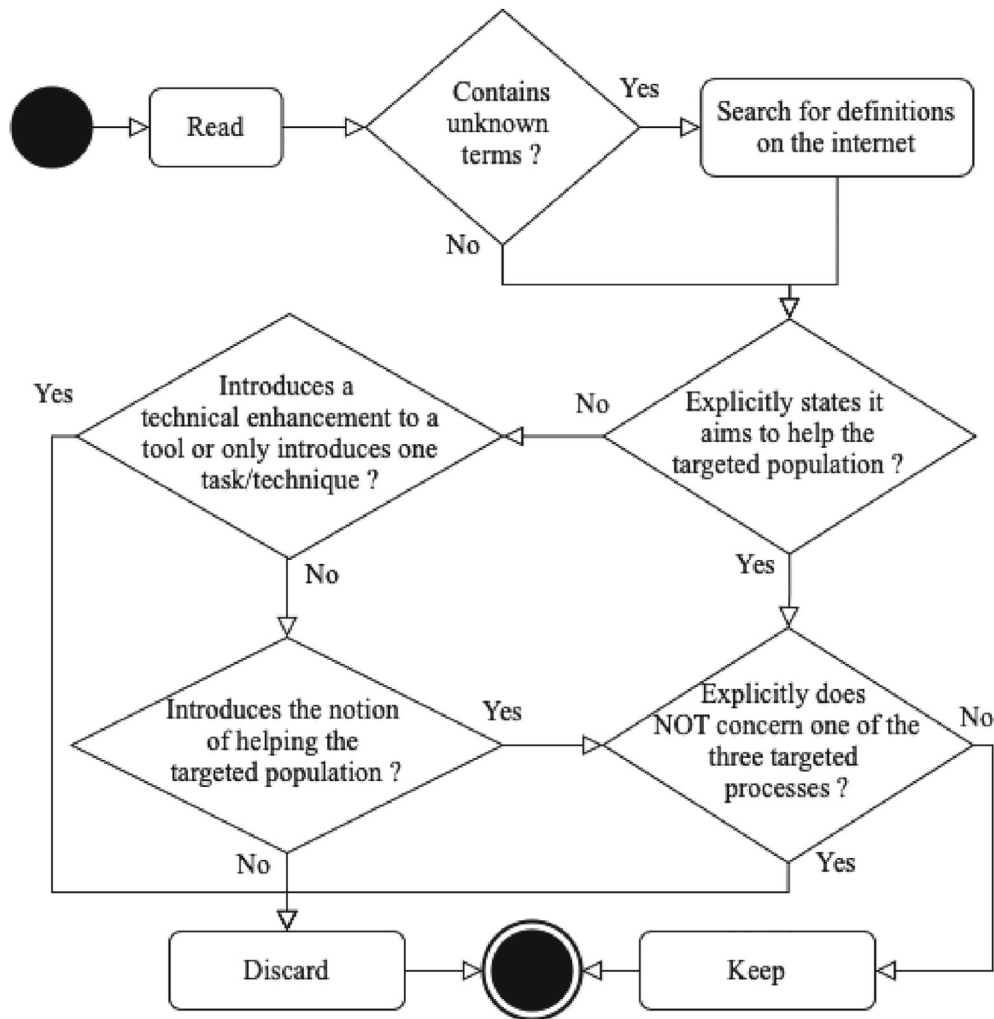


FIGURE 1 Inclusion–exclusion decision algorithm specification for the reading of title, abstract, and full paper

3.5 | Data extraction

The data we extracted from each article are:

1. The source (journal or conference), the publication year, the title, and authors. This information was extracted directly from the website of the editor.
2. The name of the tool that was eventually either provided in the title of the paper, or introduced in the full content of the paper that.
3. The supported language(s). This information was either directly given by the authors, or inferred from the datasets used for the evaluation, or deduced from the screenshots and/or the solution environment.
4. The execution environment of the assistant that was described by the authors, such as a standalone application, or an Eclipse plugin.
5. A summary of the description and goal of the assistant as provided by the authors.
6. The datasources exploited by the assistant as presented in architecture diagrams, or detailed in the description of the solution. We distinguished local datasources, for example, the local access to the IDE or the access to local versioning changes, from remote datasources, for example, the access to the StackOverflow database or the access to the continuous integration server of the project.
7. Whether the assistant is a recommender system (RS)—that is, it provides recommendations—for software engineering or not. The concept of recommender system for software engineering is based on the definition adopted

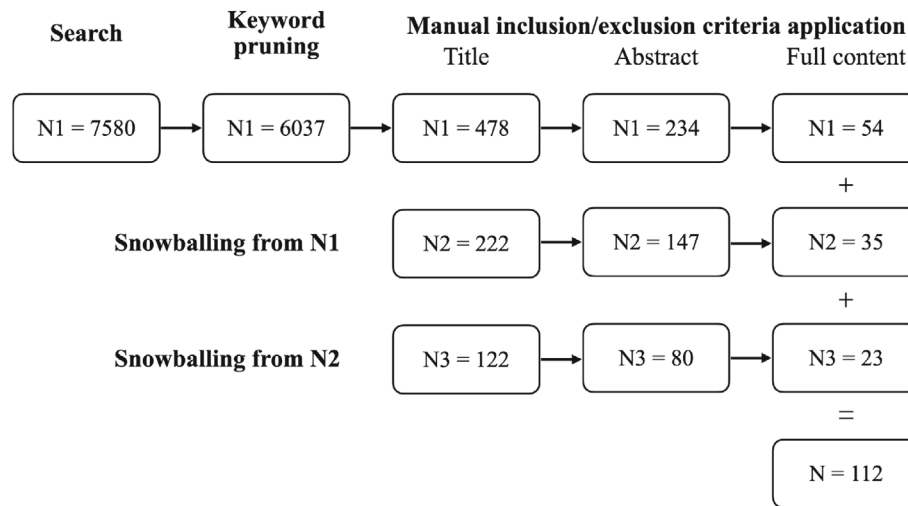


FIGURE 2 Exploration results

by Robillard and Walker in their book:⁵⁴ [...] *a software application that provides information items estimated to be valuable for a software engineering task in a given context.*

8. If the assistant is a RS: The nature of the output, the explanation system, the confidence indicator, and the feedback system. These elements relate to the way the RS supports human cognition to achieve certain tasks, hence we study (i) the nature of the output, that is, whether the RS presents the information *textually, graphically, or both*;⁵⁵ (ii) the explanation system, that is, the means with which it explains why an item is recommended (if any);⁵⁶ (iii) the confidence indicator, that is, how it shows how confident it is about a recommendation (if any);⁵⁷ and (iv) the feedback system, that is, the way it enables users to provide feedback about a recommendation (if any).⁵⁷
9. Whether the assistant uses machine learning. This information is either highlighted by the authors, or deduced from the algorithms used in the papers.
10. The automation levels of the assistant that relies on the Parasuraman et al.⁵⁸ framework for evaluating the automation levels of a system. They propose a model based on the four stages of human information processing to analyze the automation of a system over four different aspects: *information acquisition, information analysis, decision selection, and action implementation*. In order to measure the automation level for each of these four steps, we apply the 10-levels automation scale to for each step as suggested in Reference 58. The extended scales used to measure automation are provided in Appendix B.
11. Whether A user study had been conducted as described in the evaluation section of the paper.
12. Whether a replication package was provided for the evaluation.
13. Whether the source code of the assistant was provided.

Note that elements 3, 4, and 6 refer to the actual contribution presented in the paper at the time it was written, and does not take future work and directions into account (i.e., if the tool works for Java but the paper says that it could also work for C or that it is extensible to C, we do not consider C).

4 | RESULTS: ANALYSIS AND CLASSIFICATION OF SOFTWARE ASSISTANTS

This section presents the results of the data analysis we conducted on the dataset of primary studies, which are also available online.⁵²

4.1 | Selected papers

This section gathers statistics about the selected papers. Figure 3 illustrates the number of publications on software assistants published between 2010 and January 2022 (month in which we performed our search). On the left, it shows the

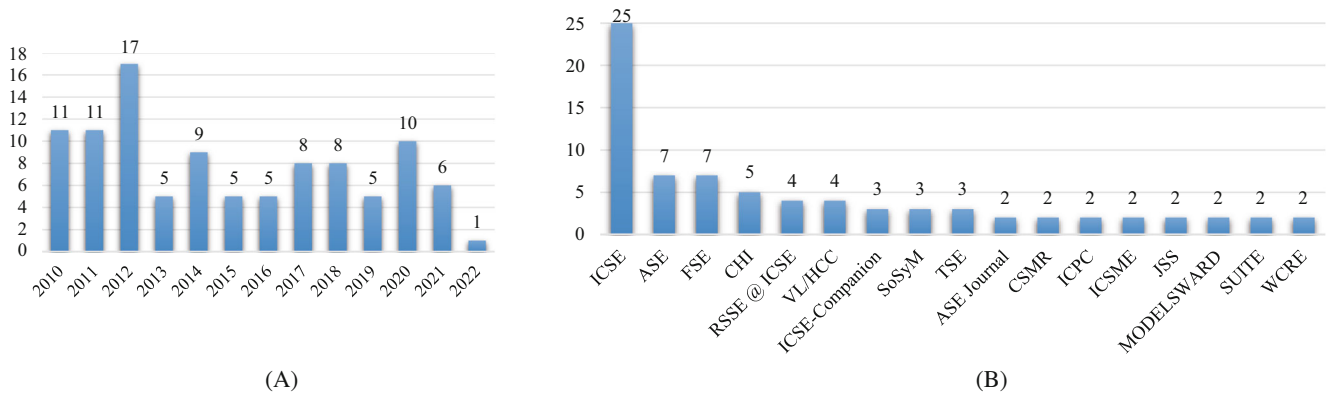


FIGURE 3 Number of publications (only those venues with 2 or more publications). (A) Number of publications per year and type. (B) Number of publications per venue

variation in the number per year. Although there is a maximum in 2012 with 18 papers and a minimum in 2013, 2015, 2016, and 2019 with 5 papers published, the number of submissions over time seems to be stable and there is continuous work. Figure 3B shows the venues in which our selected papers have been published. It is worth noting that there is no dedicated venue for the topic of software assistants and that these papers have been published in general SE conferences. It is also interesting to see how most of these papers were published in the International Conference on Software Engineering (ICSE) followed by the Automated Software Engineering Conference (ASE), which are both very prestigious conferences. The complete list of the 46 source venues of our primary studies is displayed in Table 5. We note that the two snowballing phases that we conducted enabled to collect 35 papers from 26 venues that were not included in our search queries.

4.2 | Analysis and classification results

This section is devoted to answer our research questions (cf. Section 3.1).

4.2.1 | RQ1: What are the tasks that the assistants help their users achieve, in which environments do they operate and which languages do they support?

To answer this research question, we relied on the description and goal of the assistant that we extracted from each paper as provided by its authors. From these details, we obtained information about the tasks that each assistant supports and identified the 51 tasks that are listed in the second column of Table 6. Note that assistants which support multiple languages or environments appear in different lines of Table 6. Then, we grouped those tasks into 12 coarse-grained categories to capture their purpose. These purposes are: API/code search, code completion and recommendation, code metrics, code visualization and understanding, command recommendation, find collaborators, interface prototyping, modeling, refactoring, fix and repair, resource identification, and version control system (VCS).

Figure 4 presents the number of assistants dedicated to each purpose. We can observe how some important efforts have been put into the creation of assistants for API/code search, code visualization and understanding, and fix and repair, while only isolated works have published assistants for purposes such as Interface Prototyping.

Among the wide variety of tasks, looking at Table 6, we can highlight that the Recommendation of Code Blocks from Queries is the most popular (supported by 13 assistants), followed by assistants for Enhancement of Default Code Completion Systems (supported by 9 assistants), the Recommendation of Model Elements, and the Suggestion of Code Fixes (supported by 7 assistants each). It is also worth noting how textual languages (i.e., coding) are very well supported across the whole development process—covering the tasks of finding code ideas and code excerpts, writing, refactoring and debugging/fixing as well as providing metrics—while graphical languages are under-represented.

TABLE 5 Source venues of the primary studies

Venue	Type	Initially searched	Number of studies
ICSE	Conference	Yes	25
ASE	Conference	Yes	7
FSE	Conference	Yes	7
Empirical Software Engineering	Journal	Yes	5
CHI	Conference	Yes	5
RSSE@ICSE	Workshop	Yes	4
Transactions on Software Engineering	Journal	Yes	4
VL/HCC	Conference	No	4
ICSE-Companion	Conference	Yes	3
Software and Systems Modeling	Journal	Yes	3
Automated Software Engineering	Journal	Yes	2
CSMR	Conference	No	2
Expert System with Applications	Journal	Yes	2
ICPC	Conference	No	2
ICSME	Conference	No	2
Journal of Systems and Software	Journal	Yes	2
MODELSWARD	Conference	No	2
SUITE	Workshop	No	2
WCRE	Conference	No	2
Programming Languages	Journal	No	1
AICCSA	Conference	No	1
BotSE	Workshop	No	1
CASCON	Conference	No	1
CBSof	Conference	No	1
COMPSAC	Conference	No	1
Transactions on Service Computing	Journal	No	1
Data and Knowledge Engineering	Journal	Yes	1
ECOOP	Conference	Yes	1
ESWC	Conference	No	1
HCI International	Journal	No	1
HIMI	Conference	No	1
IEA/AIE	Conference	Yes	1
Information and Software Technology	Journal	Yes	1
Internetware	Conference	No	1
IWSC	Conference	No	1
Knowledge-Based Systems	Journal	Yes	1
LIVE	Workshop	No	1
MODELS-C	Conference	Yes	1
MSR	Conference	No	1
OOPSLA	Conference	Yes	1
QUATIC	Conference	No	1
RecSys	Conference	No	1
SCAM	Conference	No	1
SLE	Conference	No	1
Transactions on Software Engineering and Methodology	Journal	Yes	1
UIST	Conference	No	1

TABLE 6 Assistant purposes and specific tasks

Purpose	Specific task	Environment	Lang./syntax	#	Primary studies	
API/code search	Enhances default code completion system	Eclipse	Java	1	[S1]	
		IntelliJ IDEA	Java	1	[S2]	
		Web browser	Java	4	[S3–S6]	
	Helps comparing libraries	Recommends code blocks from text query		Python	1	[S4]
			Adobe Flex Builder	Flex Builder*	1	[S7]
			Eclipse	Java	2	[S8, S9]
			Standalone app.	Java	1	[S10]
			Visual Studio	Java	1	[S11]
			Web browser	Java	7	[S12–S18]
					jQuery	1
		PHP	1	[S18]		
		Python	1	[S18]		
	Recommends new features with code	Web browser	Java	1	[S20]	
Code completion and recommend.	Enhances default code completion system	Android Studio	Java	1	[S21]	
		Eclipse	Java	7	[S22–S28]	
	Infers query from example expected results	Web browser	SPARQL	1	[S29]	
		Recommends code blocks from code analysis	Custom IDE	Hack	1	[S30]
	Recommends code documentation	Suggests additional query parameters		Java	1	[S30]
				Javascript	1	[S30]
				Python	1	[S30]
			Eclipse	Java	4	[S31–S34]
			Visual Studio	C#	1	[S35]
			Eclipse	Java	1	[S36]
Web browser	SPARQL	1	[S37]			
Code metrics	Augments code with indicators	Brackets	Javascript	1	[S38]	
		Eclipse	Java	3	[S39–S41]	
		Impromptu	Impromptu*	1	[S42]	
Code visualization and understanding	Annotates code with refactoring errors	Eclipse	Java	1	[S43]	
	Augments code with live examples	Eclipse	Java	1	[S44]	
		Web browser	Java	1	[S45]	
	Augments Q&A code excerpts with doc. links		Javascript	1	[S45]	
		Web browser	Natural Language	1	[S46]	
	Displays code call graphs	Eclipse	Java	1	[S47]	
	Displays library dependencies of project	Web browser	Java	1	[S48]	
	Explains code elements with rationales	Cloud9 IDE	CSS	1	[S49]	
			HTML	1	[S49]	
			jQuery	1	[S49]	
	Finds code responsible for graphical behavior	Standalone app.	Java	1	[S50]	
	Folds less informative code regions	Web browser	Java	1	[S51]	
	Proposes new code navigation system	XCode	XCode*	1	[S52]	
Recommends useful documentation pieces	Eclipse	Java	1	[S53]		
Represents code as UML models	Web browser	Java	1	[S54]		
Suggests code locations to explore	Eclipse	Java	2	[S55, S56]		
Command recommendation	Recommends tool commands to use	Eclipse	N.A.	1	[S57]	

(Continues)

TABLE 6 (Continued)

Purpose	Specific task	Environment	Lang./syntax	#	Primary studies	
Find collaborators	Suggests potential collaborators	Standalone app.	All	1	[S58]	
		Web browser	Java	1	[S59]	
Interface prototyping	Suggests examples for interface prototyping	Web browser	N.A.	1	[S60]	
Modeling	Provides a model search engine	Web browser	XMI models	1	[S61]	
	Recommends functional grouping of requirements	Standalone app.	Natural Language	1	[S62]	
	Recommends model elements	Eclipse	Ecore models	2	[S63, S64]	
		Generic Modeling Environment	DSL	1	[S65]	
		Papyrus	UML	1	[S66]	
		Standalone app.	BPMN	1	[S67]	
		Recommends modeling actions		UML	2	[S68, S69]
			Web browser	UML	1	[S70]
		Supports model transformation edition	Eclipse	ATL transformation	1	[S71]
		Supports modeling with rule checking	Standalone app.	SysML	1	[S72]
			UML	1	[S72]	
Refactoring	Suggests code refactorings	Eclipse	Java	4	[S73–S76]	
		IntelliJ IDEA	Java	1	[S77]	
	Detects and facilitates refactoring	Visual Studio	C#	1	[S78]	
Repair and fix	Recommends model fixes	Eclipse	UML	1	[S79]	
			SPEM	1	[S80]	
	Recommends Q&A posts	Eclipse	Java	2	[S81, S82]	
	Suggests source of potential build fail	Visual Studio	C#	1	[S83]	
	Suggests code fixes	Eclipse	Java	5	[S84–S88]	
			MaxCompute	Java	1	[S89]
			Web browser	C++	1	[S90]
	Suggests code for exception handling	Android Studio	Java	1	[S91]	
			Eclipse	Java	1	[S92]
			Web browser	Java	1	[S93]
Resource identification	Recommends code-related resources	Eclipse	Java	1	[S94]	
	Recommends error-related resources	IntelliJ IDEA	Java	1	[S95]	
			JSON	1	[S95]	
			XML	1	[S95]	
		Visual Studio	Visual Studio*	1	[S96]	
	Recommends libraries to add for project	Eclipse	Java	1	[S97]	
	Recommends Q&A posts	Eclipse	Eclipse*	1	[S98]	
			Java	3	[S99–S101]	
Version control system (VCS)	Displays VCS potential conflicts	Eclipse	All	1	[S102]	
			Java	1	[S103]	
			Standalone app.	All	1	[S104]
	Facilitate commit untangling	Standalone app.	N.A.	1	[S105]	
	Notifies of important VCS changes	Eclipse	N.A.	1	[S106]	
	Provides information about project changes	Web browser	Java	1	[S107]	
	Suggests ideal code reviewers	GitHub	N.A.	1	[S108]	
	Suggests conflict resolution solutions	Standalone app.	N.A.	1	[S109]	
			Eclipse	Ecore models	1	[S110]
			Standalone app.	C	1	[S111]

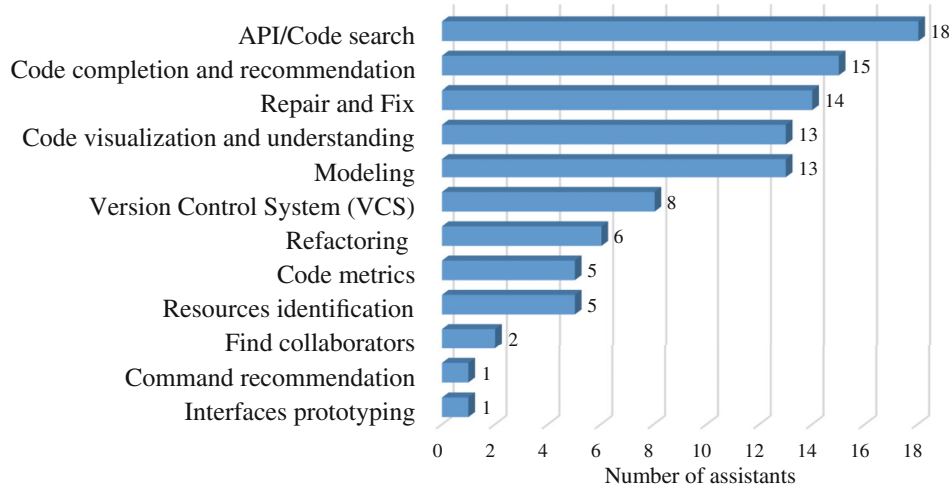


FIGURE 4 Number of assistants per purpose

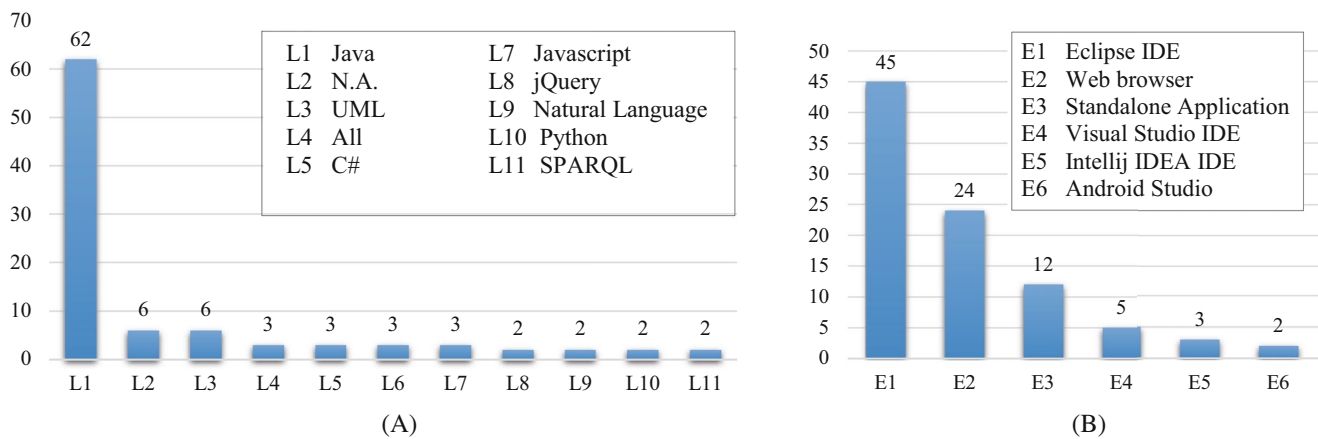


FIGURE 5 Number of publications. (A) Number of assistants per language (only languages used in two or more assistants). (B) Number of assistants per environment (only environments with 2 or more assistants)

We also extracted from each paper the environment in which the assistants work and the language (or syntax) that they support. The results can be found in the third and fourth columns of Table 6 and in Figure 5. A star (*) means that the assistant supports all languages supported by the IDE. The most popular environment by far is the Eclipse IDE with 52 out of the 112 assistants. In fact, out of all the assistants, 73 of them (65%) are part of an IDE. The second most popular environment is the Web Browser (27 out of 112—24%) and standalone applications (12 out of 112—11%). There is no clear correlation between the environments and the tasks toward which the assistants help.

With respect to the languages supported, Java has a strong monopoly with 72 out of the 112 studied assistants (64%). It is worth noting that only 5 assistants support the top-3 *programming, scripting, and markup languages* according to the Stack Overflow 2020 survey of *most popular technologies*,[‡] which are JavaScript, HTML/CSS, and SQL; and only 7 assistants support modeling languages (BPMN and XMI models).

4.2.2 | RQ2: How do software assistants assist users?

To answer this research question, we have identified the different *types* of assistants as perceived by its users and have studied, for each assistant, three Human-Computer Interaction (HCI) indicators, and the nature of the output that they provide.

[‡]<https://insights.stackoverflow.com/survey/2020#most-popular-technologies>

TABLE 7 Software assistant types

Type	Analyze and display	Help deciding	Perform actions
Informer S	Yes	No	No
Passive RS	Yes	Yes	No
Active RS	Yes	Yes	Yes

Types of assistants

We have identified that there are three main actions that assistants perform internally. The first one—and the one that every assistant integrates—is to analyze information and display the result of the analysis. The second is to help the user make a decision by suggesting one or several alternatives. The third is to perform an action based on a decision when required. Based on these three actions, we have classified the assistants and have obtained three different types as presented in Table 7. They are:

- *Informer system*, which helps toward reaching awareness about the work in progress or the environment. It takes raw data and information as input, analyze and/or aggregate it, and display the results without any side effect.
- *Passive recommender system* (passive RS), whose aim is to help the user make a decision during a software engineering task. To be able to provide meaningful potential decisions, it takes raw data or information as input, process and analyzes the inputs, and eventually produce one or several alternatives for the current decision-making problem.
- *Active recommender system* (active RS), which extends the passive RS by enabling the assistant to perform or implement the result of the decision.

For each paper included in the study, we used Table 7 to identify the type of software assistant. This process was performed manually by one author, in a systematic way, after reading the full content of each paper. Systems performing an analysis and displaying a report only were considered informer systems. As soon as the system provides a recommendation on an action to take or a choice to make, it is considered a recommender system. If it has the ability to perform the related action or implement the related choice, it is an active recommender system, else it is a passive recommender System. All analyzed papers fit in one and only one of the three types. Results of this classification were discussed among the authors to minimize the risk of errors and to validate the findings.

Figure 6 shows the number of assistants of each type grouped by purpose. We have observed that only 21% of the assistants are informer systems (24 out of 112), 40% of them are passive RS (112), and 39% are active RS (43 out of 112). This means that there is an equivalent distribution of passive and active recommender systems, that is, respectively assistants that help make decision and respectively but do not offer the possibility to implement it and offer the possibility to implement it. Correlating this with the set of purposes of assistants that we have identified, we can observe that all existing assistants for code metrics are of type informer, while there are no informer systems for purposes such as refactoring, code completion and recommendation, resources identification, and finding collaborators. This has some logic given that in these categories, the tasks that the assistants perform are likely to require making decisions and, in occasions, taking actions, too (Table 8).

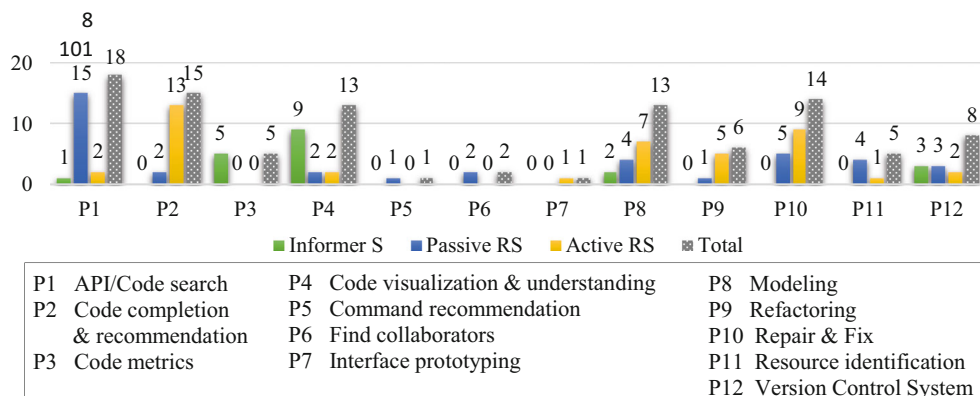


FIGURE 6 Number of assistant types for identified purposes

TABLE 8 Assistant types for specific tasks

Purpose	Specific task	Type	#	Primary studies
API/code search	Enhances default code completion system	Passive RS	1	[S1]
	Recommends API names	Informer	1	[S2]
	Recommends code blocks from text query	Passive RS	4	[S3–S6]
		Active RS	2	[S7, S8]
		Passive RS	11	[S9–S19]
	Recommends new features with code	Passive RS	1	[S20]
Code completion and recommendation	Enhances default code completion system	Active RS	8	[S21–S28]
	Infers query from example expected results	Active RS	1	[S29]
	Recommends code blocks from code analysis	Active RS	3	[S30, S34, S35]
		Passive RS	3	[S31–S33]
	Recommends code documentation	Active RS	1	[S36]
Suggests additional query parameters	Active RS	1	[S37]	
Code metrics	Augments code with indicators	Informer S	5	[S38–S42]
Code visualization and understanding	Annotates code with refactoring errors	Informer S	1	[S43]
	Augments code with live examples	Informer S	1	[S44]
	Augments Q&A code excerpts with doc. links	Informer S	1	[S45]
	Augments Q&A posts with related terms	Informer S	1	[S46]
	Displays code call graphs	Passive RS	1	[S47]
	Displays library dependencies of project	Informer S	1	[S48]
	Explains code elements with rationale	Informer S	1	[S49]
	Finds code responsible for graphical behavior	Passive RS	1	[S50]
	Folds less informative code regions	Informer S	1	[S51]
	Proposes new code navigation system	Informer S	1	[S52]
	Recommends useful documentation pieces	Passive RS	1	[S53]
	Represents code as UML models	Informer S	1	[S54]
	Suggests code locations to explore	Active RS	2	[S55, S56]
Command recommen.	Recommends tool commands to use	Passive RS	1	[S57]
Find collaborators	Suggests potential collaborators	Passive RS	2	[S58, S59]
Interfaces prototyping	Suggests examples for interface prototyping	Active RS	1	[S60]
Modeling	Provides a model search engine	Passive RS	1	[S61]
	Recommends functional grouping of requirements	Passive RS	1	[S62]
	Recommends model elements	Active RS	6	[S63–S68]
		Passive RS	1	[S69]
	Recommends modeling actions	Active RS	1	[S70]
	Supports model transformation edition	Informer S	1	[S71]
Supports modeling with rule checking	Informer S	1	[S72]	
Refactoring	Suggests code refactorings	Active RS	4	[S73, S74, S76, S77]
		Passive RS	1	[S75]
	Detects and facilitates refactoring	Active RS	1	[S78]

(Continues)

TABLE 8 (Continued)

Purpose	Specific task	Type	#	Primary studies
Repair and fix	Recommends error-related resources	Passive RS	1	[S96]
	Recommends model fixes	Active RS	1	[S79]
		Passive RS	1	[S80]
	Recommends Q&A posts	Passive RS	2	[S81, S82]
	Suggests source of potential build fail	Passive RS	1	[S83]
	Suggests code fixes	Active RS	6	[S84–S90]
		Passive RS	1	[S89]
	Suggests code for exception handling	Active RS	2	[S91, S92]
	Suggests reasons why build failed	Informer	1	[S93]
Useful resources identification	Recommends code-related resources	Passive RS	1	[S94]
	Recommends error-related resources	Passive RS	1	[S95]
	Recommends libraries to add for project	Active RS	1	[S97]
	Recommends Q&A posts	Passive RS	4	[S98–S101]
VCS	Displays VCS potential conflicts	Informer S	3	[S102–S104]
	Facilitates commit untangling	Active RS	1	[S105]
	Notifies of important VCS changes	Informer S	1	[S106]
	Provides information about project changes	Informer S	2	[S107, S112]
	Suggests ideal code reviewers	Passive RS	2	[S108, S109]
	Suggests conflict resolution solutions	Active RS	1	[S110]
		Passive RS	1	[S111]

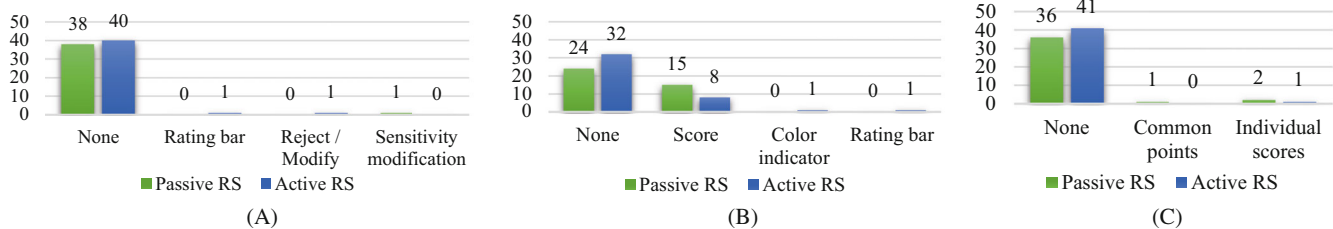


FIGURE 7 Number of assistants per HCI indicator. (A) Feedback, (B) confidence, and (C) explanations

Human–computer interaction indicators

The communication between informer systems and users is unidirectional and users are simply consumers. Unlike informer systems, both passive and active recommender systems and users interact. We have evaluated whether and how the 88 active and passive recommender systems of our primary studies implement three human–computer interaction (HCI) indicators: confidence, explanations, and feedback.

For confidence, we have observed that the assistants mostly either provide a confidence score (i.e., a single value) or they do not provide a confidence indicator at all. Only two assistants feature a graphical confidence indicator, through a colored circle (1 out of 88), or a rating bar (1 out of 88). Figure 7B shows the number of passive and active recommender systems for each group and we can observe how the majority of assistants do not provide any measure of the confidence of their recommendations.

The case is even more accentuated in the case of explanations. Figure 7C shows that only 5 assistants out of the 88 (5.7%) provides explanations. This means that, even if some assistants present confidence metrics (e.g., precision), they still act as black-boxes and do not explain the reason behind their suggestions.

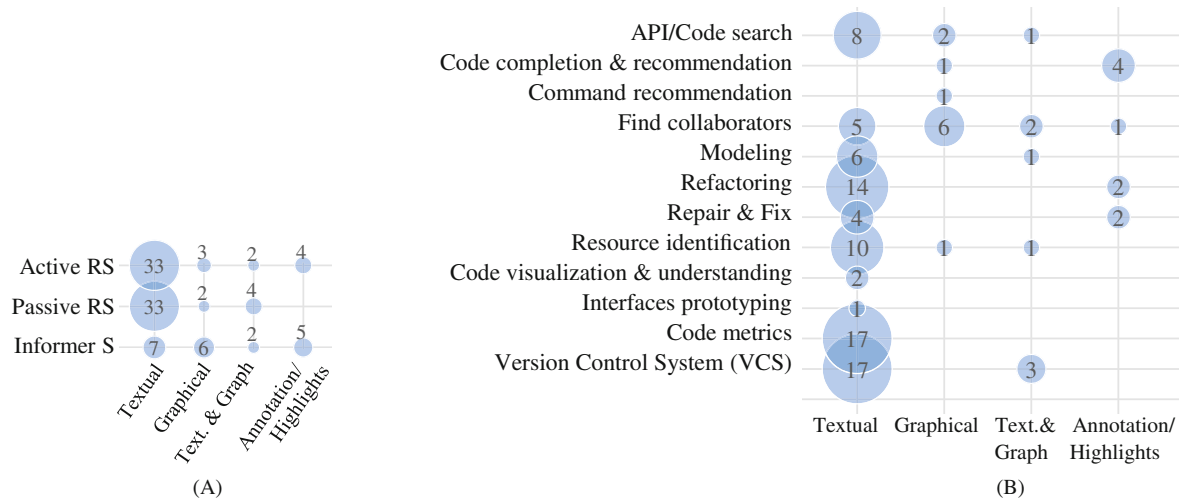


FIGURE 8 Nature of the output provided by the assistants. (A) Number of assistants grouped by type. (B) Number of assistants grouped by purpose

We have observed that only four recommender systems [S7, S18, S74, S99] include a feedback system and, therefore, let their users provide feedback, as Figure 7A presents. In [S7, S18], the recommender system allows users to rate the quality of the suggestions using a rating bar. In [S74], the system lets the users to reject or modify the suggestions keeping track of these actions. Finally, in [S99], the authors created a recommender that, by using a sensitivity feedback system, lets users adjust the recommendation confidence threshold.

Nature of the output provided by the assistants

We have analyzed what the nature of the output that the assistant provide to the users look like and have identified that these outputs are: textual, graphical, textual and graphical, and by means of annotations or highlights. These categories were proposed by Mens and Lozano⁵⁵ to describe the nature of source code recommender systems. This information was extracted manually from each paper included in our study by one author after reading the full content of the paper. All research papers included in our study features screenshots of the recommendation and/or a textual description of the output of the assistant. Thus, the categorization is based on the data provided by the authors in their papers, without personal interpretation required. Results of this classification were discussed among the authors to minimize the risk of errors.

Figure 8 presents the nature of the output of the 112 considered assistants grouped by type of assistant and by its purpose. It is worth noticing that there is no clear majority in the outputs of informer systems. In contrast, both passive and active recommender systems usually display their recommendations textually. These points are supported by the purposes of those assistants, for instance, an assistant that recommends code is likely to show code (text), and an informer assistant that is created for code visualization and understanding is likely to present its results graphically.

4.2.3 | RQ3: What kind of software technologies are used to embed knowledge in software assistants?

To answer this research question we have studied whether the assistants embed ML techniques to simulate *intelligence* and human-like decisions/actions, and what sources of information/knowledge they use.

Figure 9 shows that 6 out of 45 (13%) passive recommenders and 7 out of 43 (16%) active recommenders use machine learning. We also notice that none of the informers use machine learning. This observation might be related to the nature of the assistant; while informers only produce annotations and metrics, active and passive recommender systems aim to support the decision making process. Thus, they probably need to better adapt their behavior according to the user, what could be achieved through machine learning. Among the 13 papers using machine learning, one [S75] specifically

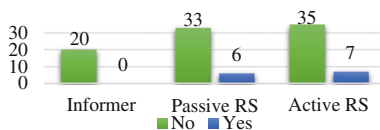


FIGURE 9 Machine learning usage (y-axis indicates number of assistants)

adapts its behavior according to user choices, one paper exploits data clustering techniques [S89], one learns query structures [S29], and 10 papers use training and testing datasets to train a predictive model to make recommendations [S10, S11, S21, S56, S58, S66, S70, S83, S85, S91].

The knowledge provided by software assistants can be extracted from various inputs, such as hard-coded rules, local analysis and metrics generation, but could also be retrieved by querying external datasources such as search-engines or dedicated APIs. Therefore, to provide information about the origin of the provided knowledge, we extracted the list of all datasources queried by the assistants included in this study. Table 9 lists all datasources exploited by the 112 software assistants under study. We have grouped all these datasources in different categories (column 2) and in two groups: local (LOC) and remote (REM) as shown in column 1.

Figure 10 presents details about the datasources that the assistants of our primary studies use—for each datasource, we present the number of assistants of each type that use it. We can see that most informer systems (17 out of 24) exploit the local information from the IDE, which is in line with what could be expected as these analyze data at hand to inform about the status of a particular environment. It is also worth noticing that passive RS use more knowledge sources than active RS.

RQ4: To what extent are software assistants automated?

In order to answer this research question, we extracted the automation features of each primary study in terms of *information acquisition*, *information analysis*, *decision selection*, and *action implementation*. During this study analysis, we also identified if each system was either triggered manually by the user, or automatically by the system: 46% of informer systems are system-triggered, while respectively only 36% and 27% of active and passive recommender systems are system-triggered.

Automation levels were extracted from the papers on the indications provided in Reference 58 and exploiting the automation scales presented in Appendix A.

Figure 11 presents the extracted automation patterns and Figure 12 summarizes the 9 patterns we identified by grouping primary studies according to their different levels of automation. Based on our analysis, 2 papers [S45, S65] (one informer system and one active recommender system) implement pattern 1, which consists in a fully automated system from information acquisition to action implementation. *Baker*, from Subramanian et al. [S45], is an informer system that augments the Stack Overflow Q&A posts in the web browser with documentation and related posts links. In this case, an overlay window is added hidden to the web page and is displayed when users hover on question elements. In their paper, Pati et al. [S65] implement a technique named *proactive modeling* into a modeling tool. The goal of this approach is to reduce the amount of manual modeling actions when creating a model with a domain-specific modeling language (DSML). To do so, the system proactively creates model elements that are deduced from the analysis of the syntax and constraints of the DSML. In both cases, the system is able to acquire and analyze information autonomously, to decide of what to do (what link to use for documentation, what elements to create), and to perform the action in the environment in use. From all the presented automation patterns, these are the two cases where users cannot influence the behavior of the system when started, which appears as fully autonomous.

We can observe that the information analysis fully automated except for one assistant [S20]. In this specific paper, the system generates a first set of recommendations and expects users to provide feedback about it. From that feedback, it works autonomously to produce final recommendations about software features to implement. Thus, this system has the same analysis abilities as other fully automated systems but integrates a human in the loop sooner to refine its results. As a consequence, we can say that all assistants are able to perform analysis on their own to produce recommendations. This was a requirement imposed by our characterization of assistants, since without this step, a piece of software would not be considered an assistant but a tool.

Figure 13 presents the number of assistants that implement these patterns aggregated by assistant type.

TABLE 9 Assistant datasources

Type	Category	Datasource	#	Primary studies	
LOC	Assistant knowledge	Collection database	1	[S55]	
		Local rules	2	[S80, S86]	
	Documentation repository	Project documents	3	[S62, S70, S72]	
		Execution environment	Access to runtime environment	1	[S42]
	ByteCode Access		1	[S50]	
	Java Binaries		1	[S25]	
	JVM Access		2	[S40, S50]	
	IDE	IDE Access	72	[S1, S2, S8, S9, S11, S21–S36, S38–S44, S47–S50, S52–S57, S63–S67, S70, S71, S73–S78, S80, S83–S88, S90–S92, S94–S104]	
		IDE Cache	1	[S39]	
		Tool commands registry	1	[S57]	
	Local sources and versioning	Sources + Versioning local	76	[S1, S2, S8, S11, S16, S21–S36, S38–S44, S47–S50, S52–S57, S63–S67, S70, S71, S73–S78, S80, S83–S88, S90–S92, S94–S99, S102–S106, S108–S111]	
			Web browser content	3	[S38, S94, S95]
			Access to web browser request and content	3	[S38, S94, S95]
			Access to internet web searches	1	[S14]
Access to web browser editor			2	[S45, S60]	
REM	API list repository	API libraries	7	[S2, S6, S11, S12, S14–S16]	
		Maven repository	1	[S3]	
	Code/Model/Q&A/Ontology repository	Stack Overflow database	17	[S4, S6, S8, S11, S15, S18, S19, S36, S45, S46, S81, S82, S96, S98–S101]	
		Source Code Corpus	18	[S1, S12, S17, S20, S23, S24, S28, S31, S33, S34, S51, S53, S79, S87, S89, S97, S104, S107]	
		Github Rest API	9	[S3, S5, S9, S27, S30, S35, S59, S100, S112]	
		Model Corpus	5	[S61, S63, S67–S69]	
		Android apps bytecode	2	[S21, S91]	
		SPARQL endpoints	2	[S29, S37]	
		Android Sources	1	[S13]	
		Bytes.com	1	[S96]	
		Codeguru.com	1	[S96]	
		Codeplex repository	1	[S35]	
		Daniweb.com	1	[S96]	
		DevShed	1	[S96]	
		Fdroid repository	1	[S10]	
		Feature Request List	1	[S12]	
		Fix Library	1	[S90]	
		GenMyModel repository	1	[S66]	
		Gitee repository	1	[S32]	
		libraries.io	1	[S3]	
		Ontology corpus	1	[S64]	
	Source Forge repository	1	[S58]		
	Tag wiki	1	[S4]		
Web pages corpus	1	[S60]			
Wikipedia	1	[S46]			
Wordnet	3	[S13, S25, S67]			
World of code	1	[S3]			

(Continues)

TABLE 9 (Continued)

Type	Category	Datasource	# Primary studies
Documentation repository	Project/Code documentation access	Project/Code documentation	4 [S12, S14, S15, S20]
		Android documentation	1 [S13]
Project server	Bug reports	Bug reports	1 [S85]
		CI reports	1 [S83, S93]
		Runtime execution traces	1 [S39]
Search engine	Google	Google	3 [S7, S82, S99]
		Bling access	2 [S82, S99]
		Blekko access	1 [S99]
		Yahoo access	1 [S82]
Versioning repository	Project VCS server	Project VCS server	6 [S89, S93, S103, S105, S108, S109]
		Closed feature request repo	1 [S12]

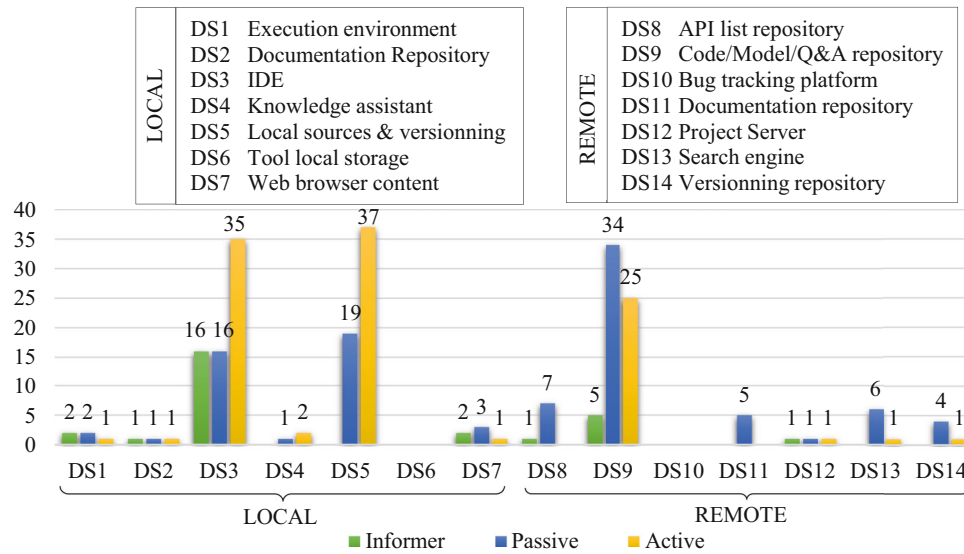


FIGURE 10 Data source usage by assistant type (y-axis indicates number of assistants)

Pattern	Number of assistants	Information Analysis	Action Implementation	Information Acquisition	Decision Selection
1	2 / 112	Fully Automated (10)	Fully Automated (10)	Fully Automated (10)	Fully Automated (10)
2	30 / 112	Fully Automated (10)	Fully Automated (10)	Fully Automated (10)	Not Automated (2-5)
3	11 / 112	Fully Automated (10)	Fully Automated (10)	Not Automated (2-5)	Not Automated (2-5)
4	2 / 112	Fully Automated (10)	Not Supported (1)	Fully Automated (10)	Fully Automated (10)
5	23 / 112	Fully Automated (10)	Not Supported (1)	Fully Automated (10)	Not Automated (2-5)
6	11 / 112	Fully Automated (10)	Not Supported (1)	Fully Automated (10)	Not Supported (1)
7	30 / 112	Fully Automated (10)	Not Supported (1)	Not Automated (2-5)	Not Automated (2-5)
8	2 / 112	Fully Automated (10)	Not Supported (1)	Not Automated (2-5)	Not Supported (1)
9	1 / 112	Not Automated (2-5)	Not Supported (1)	Not Automated (2-5)	Not Automated (2-5)

FIGURE 11 Automation patterns

subject of socio-cultural systems according to Reference 51. This naturally limits the generality of our conclusions to other phases of the software engineering process.

Another threat to external validity relates to our primary studies. This review presents 112 software assistants identified with the protocol in Section 3. First, it is possible that some existing assistants were missed, for instance, because they are not published in research papers or did not match our queries. To mitigate this concern as much as possible, our search contained a large set of keywords with wildcards to maximize matches. Second, our search was limited to number of peer-reviewed venues that guarantee the quality of the resulting articles, but we may have missed some papers that are not published in these venues or indexed in dblp.org. To mitigate these concern, a snowballing phase composed by two iterations was conducted, which enabled us to identify 26 new venues and 35 papers.

Finally, the last threat to external validity is linked to the nature of the considered systems. This work studies implemented prototypes or mature systems that can actually be used and tested. Therefore, it excludes articles presenting new algorithms and techniques which are not part of a usable software. This ensures a focus on systems that are really and directly ready to use by software engineers, but may have missed promising potential assistants that are not mature as of today.

We consider that this study has been validated through its systematic protocol and by the different reviews and discussions internally conducted by the authors. This work also provides all the required information for the mapping study to be replicated, which may reduce some external validity concerns.

5.2 | Construct validity

Construct validity refers to using the right tools and tests to get the right measures.

In the context of this systematic mapping study, the most critical point was the extraction of information from the articles. To the extent of our knowledge, the protocols and tools that we used and explained in Section 3 are the most appropriate.

Most of the extracted data represents nominal variables, with no intrinsic ordering. This is the case, for instance, of the papers' metadata, supported languages, or even datasources, which cannot be framed within a specific scale. For the identification of other variables such as the characteristics of the recommender systems (e.g., the nature of the output, the explanation system, the confidence indicator, and the feedback system), we relied on the classification of design decisions proposed by Mens and Lozano,⁵⁵ and for the automation level we used the scales in Reference 59 to match the needs of the analysis.

5.3 | Internal validity

Internal validity is defined as the extent to which the observed results are reliable and lack methodological errors.

The search process relies on an algorithm which performs automated queries, and hence prevents any human error. The exclusion process is the most sensitive part of the exploration protocol. It was conducted manually, and resulted in the rejection of 5925 papers. In order to limit the subjectivity of this treatment, the exclusion was conducted by only one of the authors, following a rigorous protocol defined in Section 3. Additionally, before performing the exclusion, an IRR score was computed with another author to verify compliance with the criteria.

The data extraction was also performed manually, and consisted in reading all the considered papers and filling in a table with the criteria defined in the protocol. This could lead to errors due to human misinterpretation of the content of the article, missing information or gray areas of the article. However, we feel in a position to say the content of the articles was sufficient to obtain all the required information. In addition, articles whose information was considered ambiguous were checked at least twice, at different times, in order to minimize errors due to fatigue.

6 | DISCUSSION, CHALLENGES, AND OPEN LINES OF WORK

In this section, we summarize the main findings for each research questions, and identify eventual research challenges that could drive innovation in the field of software assistants for software design, construction and maintenance. For each

challenge, we provide a name, a description of what is currently missing, and potential directions with concrete action points to improve this situation.

6.1 | RQ1: What are the tasks that the assistants help their users achieve, in which environments do they operate and which languages do they support?

We found that most of software assistants focus on helping software engineers with code-related tasks. Assistance with other tasks such as modeling, finding collaborators or learning IDE commands remains anecdotal. This shows that the literature focuses more on supporting software construction tasks rather than software design or maintenance tasks. Hence, we identify assistance systems for software design as an open line for research, to support a broader variety of tasks and participants within the software development life cycle.

64% (72 over 112) of the software assistants support Java, while most of the other identified programming languages are only supported by one assistant over 47. Therefore, 52 over 112 (81%) software assistants are mainly integrated in Eclipse IDE, or are not part of any specific development environment (respectively 46% and 35%).

Challenge 1: Java and Eclipse predominance. One reason for this distribution could be the large availability of resources for Eclipse plugin development, Eclipse mainly being used as a Java IDE. This technical motivation, however, might take the research away from the actual practices and needs of software engineers. Stack Overflow surveys from 2018 to 2020 show that Java is only the 5th most commonly used programming language, behind JavaScript, HTML/CSS or Python. Eclipse is the 8th most commonly used IDE,[§] behind Visual Studio or IntelliJ. Although these historical technologies remain important, new research work must lead innovation on these popular technologies and their inherent technical and human issues.

Action point: In addition to assistants for the Java language and the Eclipse IDE, the community may bring existing or new assistance mechanisms to these increasingly popular environments and languages.

Challenge 2: IDE versus standalone application. Most of the software assistants created as standalone application or websites have an alternative integrated to an IDE for the same task (see Table 6). The papers presenting these systems do not provide an explanation for this design decision, which might also be justified by the ease of development of a website or standalone application compared to an IDE-embedded solution. However, repeated changes of work environment (e.g., switching from the IDE to the web browser) have been correlated with interruptions in the cognitive process, which in turn are correlated with productivity losses.^{60,61}

Action point: Human-centric considerations must be taken into account, in addition to the quality of the algorithm, in order to create truly efficient systems.

The following research question emphasizes the previous notion, while showing that the existing literature performs poorly on such HCI aspects.

6.2 | RQ2: How do software assistants assist users?

This systematic mapping study identified three major types of assistants which respectively have increasing competences: informer systems, passive recommender systems, and active recommender systems. Some tasks call for one specific assistant type, such as providing code metrics, which only requires informer systems. Other tasks require recommender systems, which could in turn be *passive* or *active*.

Challenge 3: Lack of action mechanisms. We observed that papers featuring passive recommender systems did not justify the reason why they do not implement the related action mechanism, which stands for *active* in active recommender systems.

Action point: We observe a need for further work in the direction of exploring how passive recommender systems can be turned into active recommender systems (if applicable) by coming up with a set of actions to integrate and execution capabilities.

The acceptability and usability of information systems strongly relies on the relation of trust between one user and the system.⁶² This confidence is obtained, among other methods, by allowing the user to understand how the system works and to control it to influence its behavior.⁶³

[§]2020 survey do not provide information about IDE.

Challenge 4: HCI support. Our results about HCI indicators clearly state that the user interfaces of recommender systems hardly support these human aspects. When analyzing the 88 identified recommender systems, 30% display a confidence indicator for their results, 5% allow users to provide feedback on their recommendations, and 6% provide an explanation about their recommendations.

Action point: Studying these HCI aspects, such as how information is presented, how transparency is achieved, and how users control the system, is one major research challenge for software assistants, in order to make algorithms part of a whole *user experience*.^{64,65}

6.3 | RQ3: What kind of software technologies are used to embed intelligence in software assistants?

Software assistants must hold a minimal degree of human-like intelligence to be able to perform data analysis and produce new information. This is achieved by implementing hard coded rules and algorithms, which exploit popular technologies and libraries to produce results, with or without the need of external data or knowledge.

Challenge 5: ML support. Our analysis shows that only one software assistant out of the 112 fully implemented systems we studied exploits machine learning techniques which gives them the ability to change their behavior by learning from examples.

Action point: While the identified papers do not motivate their choice not to embed machine learning, artificial intelligence or even cognification mechanisms, one open line of work would be to investigate the use of such techniques to support software assistants adaptability regarding user preferences and profiles.

In order to perform analysis and provide recommendations, software assistants exploit data which they obtain locally and/or remotely. Local information represents the artifacts edited by software engineers (e.g., the source code), the state and history of their IDE, and eventually other client-side information. Remote information describes the data created by co-workers, third-parties or the community (e.g., source code, the content of company databases or model repositories) as well as answers to questions on social community websites (e.g., Stack Overflow), or available documentation (e.g., system requirements).

Challenge 6: Data exploitation. Our results first show that, as expected, informer systems mainly use data from local sources (almost 77% of their datasources usage). However, it appears that passive recommender systems tend to exploit much more data from remote data sources than active recommender systems (57% versus 27%). Based on the nature of this data, one may think that current passive recommender systems are likely to be more accurate and “intelligent” than the existing active recommender systems since they are exploiting external knowledge more heavily. One potential reason to justify this result could be the effort required to build that extra step that makes a passive recommender system into an active recommender system.

Action point: Future work could investigate how to facilitate the integration and exploitation of community data, information and knowledge into the analysis algorithms of software assistants with the goal to empower them. Remote datasources rely on the availability of artifacts and knowledge created by the community. Aggregating these elements creates a global knowledge database which references many general concepts, sometimes called *background knowledge*. Then, this background knowledge can help software engineers deal with common issues, related to general concepts. Building curated and reliable code and data repositories represents another open challenge for the research in software assistants.

6.4 | RQ4: To what extent are software assistants automated?

We have considered the automation of the software assistants in two distinct phases: (i) the activation of the system, that is, the trigger mechanism and (ii) the behavior of the system once started. Our results show that software assistants are either triggered manually by the user (user event trigger) or automatically by the system (system event trigger). Only 36% and 27% of respectively active and passive recommender systems are triggered automatically, compared to 46% of informer systems and passive recommender systems. While active recommender systems might present more risks to be automated because they can create, modify or delete content, as it is the case for other AI-empowered systems,⁶⁶ this does not apply to passive recommender systems.

Challenge 7: System automation. The extracted automation patterns for each system revealed that software assistants follow one or two main interaction patterns according to their type. To date, only two systems are completely automated from information acquisition to action implementation. This implies that software engineers are almost always involved in interacting with the system. Information analysis is fully automated for all software assistants, which aligns with our definition of software assistant. When supported (only for active recommender systems), action implementation is fully automated. Information acquisition is fully automated for informer systems, while it could either be poorly automated or fully automated for recommender systems. It is worth noting that only systems with a fully automated information acquisition step are triggered automatically, while the others require the users to trigger them manually. When supported (for recommender systems), decision selection remains the least automated part of the process, being poorly automated. The poor automation of these steps might be the consequence of technical limitations (such as user intent acquisition, or knowledge access) or acceptability issues (users might want to keep control over the system).

Action point: Exploring new automation configurations for software assistants, while studying its impact on acceptability could deal with the concerns previously identified. While we believe that our classification for interaction patterns might be useful to tool vendors who want to include assistance systems into their software solution, we encourage researchers to think outside the box and assess new interaction patterns for software assistants. Exploiting existing interaction patterns is in line with Jakob's law,⁶⁷ which states that users prefer your system to work the same way as all the other systems they already know.

6.5 | General discussion

Challenge 8: Professional software engineering versus end user software engineering. The majority of the assistants identified in this study target professional software developers by focusing on environments or functionalities that are part of the professional practice of software engineering. This is the case for the use of VCS, advanced refactoring, software modeling linked to code, graphical debugging tools, or users working in teams. The rest of the assistants mentioned in this study, however, provide features that can go beyond the professional work environment, being useful to non-professional developers practicing end-user software engineering (EUSE).⁶⁸ In the EUSE definition, any user coding for himself and not professionally for others is considered an end-user, who codes for his own benefit.⁶⁹

In our study, systems for recommending code, interesting resources (files or Q&A), or model elements, can indeed be useful to end users who develop software for their own use. The systematic study conducted by Barricelli et al.⁷⁰ shows that these functionalities are implemented with techniques common to those used in EUSE, such as wizard-based, text-based, model-based, natural language, template-based, or programming by demonstration. This same study shares our observations, that there is a lack of works that address the use of other interaction techniques and media (voice, touch, and the use of social media and crowdsourcing technologies) to design assistance systems. The paper from Sanctorem et al.⁷¹ investigates several aspects of recommendation systems for EUSE, that we also extracted in our study. This is especially the case for HCI indicators such as the nature of the output, the explanation of the recommendations, or the ability to provide feedback about the result.

The results of Sanctorem et al. highlight the gap between our results and the expectations of potential end-users of the assistants. They report that end-users would prefer recommendations to be mainly presented graphically while most of the identified assistants of our study provided textual results. Similarly, the results indicate that end-users would like results to be presented (mostly) in sidebars in the working environment, and that they should be able to provide rating and feedback on both recommendations and explanations. Only 5% and 6% of the identified assistants respectively allow users to provide feedback on the recommendations (none for the explanations), and provide an explanation about their recommendation. Other research papers also highlight that the identified assistants do not respect design guidelines for EUSE systems. Spahn et al.⁷² recommend EUSE working environments to be as natural as possible, so end-users could be able to express their ideas and implement them *in the same way as they think about them*. The availability of dedicated environments for end-users is not reflected in our findings (see Table 6). The state of the art in end-user software engineering of Ko et al.⁶⁹ discusses code recommendation for EUSE, and identify finding code and abstraction, or to know they exist as a fundamental challenge for code reuse. To address this challenge, they recommend EUSE assistants to enable users to (i) modify the code and to customize it for specific purpose, (ii) to enable the assistant to be tailored to adapt to target end-users who will differ from one another, and (iii) to help end-users understand in advance whether some API or library would be suitable for a task without causing issues in the future. The code (or resource) recommenders identified in this study do not offer this expected level of adaptability.

Action point: While some of the identified assistants can target both professional software engineers and end-users performing EUSE, it seems that these assistants are not adapted for a non-professional use according to main research results. As a consequence, future work should investigate if EUSE guidelines also applies to software engineering professionals, which therefore would imply that the design of the identified assistants shall be improved in all cases. As a takeaway message from this discussion, software editors should also take care to follow EUSE guidelines to broaden the potential use of their software assistants for software engineering.

7 | CONCLUSIONS

Since 2010, many types of assistants have become mainstream. For instance, voice assistants like Amazon Alexa and Apple's Siri and the exponential adoption of chatbots in e-commerce. In this article, we have studied whether this same trend is happening in software engineering, that is, whether assistants are becoming a mainstream tool to speed up software development projects. And if so, what the most popular types of assistants are and how they work.

We have observed that the number of research articles introducing fully finished and ready-to-use software assistants for software design, construction and maintenance tends to decrease. Furthermore, the assistants featured in our set of primary studies are not very well aligned with the software engineers' workflows and preferred programming languages and development environments. We also identified that these works do not take advantage from the recent progress of research in the fields of ML or HCI for information systems. This potentially reduces the effectiveness of the created systems, while limiting their usability and thus their acceptability. These two problems are even the more important as the number of smart IDEs from industry continues to increase (e.g., IntelliCode,[¶] Kite,[#] Codota,^{||} TabNine^{**}), in contrast to the number of related research papers.

Thus, research in the field of software assistants for software engineering seems to lag behind the practices and techniques available to date. We see this is a strong opportunity to develop a new generation of assistants that embraces some of the new research results (e.g., ML-based recommenders) and adapts them keeping in mind the findings challenges we described above.

At the same time, we think it is important to pay attention to the evaluation of this new breed of assistants. Rigorous research requires the replicability of the published work in order to compare and evaluate new solutions within the same field. This does not seem to be the case yet in this domain. Only 11 articles out of the 47 primary studies have made a dataset available to provide a benchmark for future comparisons. Finally, another important weakness that we have detected is that that only 43% of these user-centric systems conducted an evaluation with real users.

With the results of this systematic mapping study, we would like to encourage researchers to address the challenges that we have identified related to software assistants for helping with software development related activities.

ACKNOWLEDGMENTS

This work is partially supported by the Spanish Government under projects LOCOS (PID2020-114615RB-I00) and IPSCA (PID2021-125527NB-I00); and TRANSACT (Grant Agreement No. 101007260) and AIDOaRt (Grant Agreement No. 101007350), which have received funding from the ECSEL Joint Undertaking (JU). The JU receives support from the European Union's Horizon 2020 Research and Innovation Programme and Sweden, Czech Republic, France, Netherlands, Finland, Germany, Poland, Austria, Spain, Belgium, Denmark, and Norway.

AUTHOR CONTRIBUTIONS

The idea of investigating the topic of Software Assistants in Software Engineering was suggested by SG, XLP, and MSL. The conceptualization and writing of the protocol of the Systematic Mapping Study was conducted and reviewed by all authors equally. MSL and LB ran the research investigation phase of the study, including the verification step. The data extraction was performed by MSL, and data visualization artifacts were produced by LB. The online appendix was created by MSL. MSL and LB led the manuscript writing, and all authors equally contributed to its review.

[¶]<https://visualstudio.microsoft.com/services/intellicode/>

[#]<https://kite.com/>

^{||}<https://www.codota.com/>

^{**}<https://tabnine.com/blog/deep>

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in our repository at <https://software-assistant.univ-lille.fr>.

ORCID

Maxime Savary-Leblanc  <https://orcid.org/0000-0002-7327-5676>

REFERENCES

1. Mens T. On the complexity of software systems. *Computer*. 2012;45(8):79-81. doi:10.1109/MC.2012.273
2. Yu S, Zhou S. A survey on metric of software complexity. Proceedings of the 2010 2nd IEEE International Conference on Information Management and Engineering; 2010:352-356.
3. Blackburn JD, Scudder GD, Van Wassenhove LN. Improving speed and productivity of software development: a global survey of software developers. *IEEE Trans Softw Eng*. 1996;22(12):875-885. doi:10.1109/32.553636
4. Meyer AN, Barton LE, Murphy GC, Zimmermann T, Fritz T. The work life of developers: activities, switches and perceived productivity. *IEEE Trans Softw Eng*. 2017;43(12):1178-1193. doi:10.1109/TSE.2017.2656886
5. Ramsay A. A distributed programming assistant. *Softw Pract Exp*. 1983;13(11):983-992. doi:10.1002/spe.4380131102
6. Levary RR, Lin CY. Modelling the software development process using an expert simulation system having fuzzy logic. *Softw Pract Exp*. 1991;21(2):133-148. doi:10.1002/spe.4380210203
7. Rosen C, Shihab E. What re mobile developers asking about? A large scale study using stack overflow. *Emp Softw Eng*. 2016;21(3):1192-1223.
8. Graziotin D, Fagerholm F, Wang X, Abrahamsson P. On the unhappiness of software developers. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering; 2017.
9. Graziotin D, Fagerholm F, Wang X, Abrahamsson P. What happens when software developers are (un)happy. *J Syst Softw*. 2018;140:32-47.
10. Rech J, Ras E, Decker B. Intelligent assistance in German software development: a survey. *IEEE Softw*. 2007;24(4):72-79. doi:10.1109/MS.2007.110
11. Petersen K, Vakkalanka S, Kuzniarz L. Guidelines for conducting systematic mapping studies in software engineering: an update. *Inf Softw Technol*. 2015;64:1-18. doi:10.1016/j.infsof.2015.03.007
12. Petersen K, Feldt R, Mujtaba S, Mattsson M. Systematic mapping studies in software engineering. EASE'08; 2008:68-77; Swindon, GBR.
13. Ossher H, Harrison W, Tarr P. Software engineering tools and environments: a roadmap. Proceedings of the Conference on the Future of Software Engineering of ICSE'00. Association for Computing Machinery; 2000:261-277; New York, NY.
14. Boehm B. A view of 20th and 21st century software engineering. Proceedings of the 28th International Conference on Software Engineering of ICSE'06. Association for Computing Machinery; 2006:12-29; New York, NY.
15. Grudin J. Computer-supported cooperative work: history and focus. *Computer*. 1994;27(5):19-26. doi:10.1109/2.291294
16. Jennings N, Wooldridge M. Software agents. *IEE Rev*. 1996;42(1):17-20. doi:10.1049/ir:19960101
17. Wooldridge M. Intelligent agents: the key concepts. *Multi-Agent Systems and Applications II*. Springer; 2002:3-43.
18. Girgensohn A, Redmiles D, Shipman F. Agent-based support for communication between developers and users in software design. Proceedings of the 9th Knowledge-Based Software Engineering Conference KBSE'94; 1994:22-29.
19. Huo Q, Zhu H, Greenwood S. A multi-agent software engineering environment for testing Web-based applications. Proceedings of the 27th Annual International Computer Software and Applications Conference. COMPAC 2003; 2003:210-215.
20. Erlenhov L, Oliveira NG, Scandariato R, Leitner P. Current and future bots in software development. Proceedings of the 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE); 2019:7-11.
21. Ferrara E, Varol O, Davis C, Menczer F, Flammini A. The rise of social bots. *Commun ACM*. 2016;59(7):96-104. doi:10.1145/2818717
22. Storey MA, Zagalsky A. Disrupting developer productivity one bot at a time. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering of FSE 2016; 2016:928-931; Association for Computing Machinery, New York, NY.
23. Storey MA, Serebrenik A, Rosé CP, Zimmermann T, Herbsleb JD. BOTse: bots in software engineering (Dagstuhl Seminar 19471). *Dagstuhl Rep*. 2020;9(11):84-96. doi:10.4230/DagRep.9.11.84
24. Pérez-Soler S, Guerra E, de Lara J. Collaborative modeling and group decision making using chatbots in social networks. *IEEE Softw*. 2018;35(6):48-54. doi:10.1109/MS.2018.290101511
25. Shadbolt N, Motta E, Rouge A. Constructing knowledge-based systems. *IEEE Softw*. 1993;10(6):34-38. doi:10.1109/52.241964
26. Bélisle C. *Literacy and the Digital Knowledge Revolution*. Digital Literacies for Learning; 2006:51-67.
27. San SR. A new concept of knowledge. *Online Inf Rev*. 2002;26:239-245. doi:10.1108/14684520210438688
28. Ackoff RL. From data to wisdom. *J Appl Syst Anal*. 1989;16(1):3-9.
29. Fiore S, Elias J, Salas E, Warner N, Letsky M. From data, to information, to knowledge: measuring knowledge building in the context of collaborative cognition. *Macro-cognition Metrics and Scenarios: Design and Evaluation for Real-World Teams*; 2010:179-200.
30. Rowley J. The wisdom hierarchy: representations of the DIKW hierarchy. *J Inf Sci*. 2007;33(2):163-180. doi:10.1177/0165551506070706
31. Pearlson K, Saunders C. *Managing & Using Information Systems: A Strategic Approach*. John Wiley & Sons; 2013.
32. Rus I. Guest editors' introduction: knowledge management in software engineering. *IEEE Softw*. 2002;19:26-38. doi:10.1109/MS.2002.1003450
33. Lethbridge T. What knowledge is important to a software professional? *Computer*. 2000;33(5):44-50. doi:10.1109/2.841783

34. Mussbacher G, Combemale B, Kienzle J, et al. Opportunities in intelligent modeling assistance. *Softw Syst Model*. 2020;19:1045–1053. doi:10.1007/s10270-020-00814-5
35. Maedche A, Legner C, Benlian A, et al. AI-based digital assistants. *Bus Inf Syst Eng*. 2019;61(4):535-544. doi:10.1007/s12599-019-00600-8
36. Maedche A, Morana S, Schacht S, Werth D, Krumeich J. Advanced user assistance systems. *Bus Inf Syst Eng*. 2016;58(5):367-370. doi:10.1007/s12599-016-0444-2
37. Bernard D. Cognitive interaction: towards “cognitivity” requirements for the design of virtual assistants. Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC); 2017:210-215.
38. Murphy GC. Beyond integrated development environments: adding context to software development. Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER); 2019:73-76.
39. Bryndin E. Collaboration of intelligent interoperable agents via smart interface. *Int J Data Sci Technol*. 2019;5:66. doi:10.11648/j.ijdst.20190504.11
40. Ackerman P, Beier M. Knowledge and Intelligence. In: Wilhelm O, Engle RW, eds. *Handbook of Understanding and Measuring Intelligence*, SAGE Publications, Inc.; 2005:125-140.
41. Malone TW. How can human-computer “superminds” develop business strategies? *The Future of Management in an AI World: Redefining Purpose and Strategy in the Fourth Industrial Revolution*. Springer International Publishing; 2020:165-183.
42. Gasparic M, Janes A. What recommendation systems for software engineering recommend: a systematic literature review. *J Syst Softw*. 2016;113:101-113. doi:10.1016/j.jss.2015.11.036
43. Almonte L, Guerra E, Cantador I, de Lara J. Recommender systems in model-driven engineering. *Software Syst Model*. 2022;21:249–280. doi:10.1007/s10270-021-00905-x
44. Savchenko D, Kasurinen J, Taipale O. Smart tools in software engineering: a systematic mapping study. *Proceedings of the 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. Vol 2019; IEEE; 2019:1509-1513. doi: 10.23919/MIPRO.2019.8756975
45. Borges O, Couto J, Ruiz D, Prikladnicki R. How machine learning has been applied in software engineering? Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 2: ICEIS. INSTICC; 2020:306-313; SciTePress.
46. Shafiq S, Mashkooor A, Mayr-Dorn C, Egyed A. Machine learning for software engineering: a systematic mapping. arXiv preprint arXiv:2005.13299; 2020.
47. Iung A, Carbonell J, Marchezan L, et al. Systematic mapping study on domain-specific language development tools. *Empir Softw Eng*. 2020;25(5):4205-4249. doi:10.1007/s10664-020-09872-1
48. Sebastián G, Gallud JA, Tesoriero R. Code generation using model driven architecture: a systematic mapping study. *J Comput Lang*. 2020;56:100935. doi:10.1016/j.cola.2019.100935
49. Brunschwig L, Guerra E, de Lara J. Modelling on mobile devices. *Softw Syst Model*. 2022;21:179–205. doi:10.1007/s10270-021-00897-8
50. Bourque P, Fairley RE, Society IC. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. 3rd ed. IEEE; 2014.
51. Portillo-Rodríguez J, Vizcaino A, Piattini M, Beecham S. Tools used in global software engineering: a systematic mapping review. *Inf Soft Technol*. 2012;54(7):663-685.
52. SLR data tables and resources. <https://software-assistant.univ-lille.fr>.
53. Koo TK, Li MY. A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *J Chiropract Med*. 2016;15(2):155-163.
54. Robillard MP, Walker RJ. An Introduction to Recommendation Systems in Software Engineering; 2014:1-11.
55. Mens K, Lozano A. Source code-based recommendation systems. Recommendation Systems in Software Engineering; 2014:93-130.
56. Pu P, Chen L. Trust building with explanation interfaces. Proceedings of the 11th International Conference on Intelligent User Interfaces of IUI'06; 2006:93-100; Association for Computing Machinery, Sydney, Australia.
57. O'Donovan J, Smyth B. Trust in recommender systems. Proceedings of the 10th International Conference on Intelligent User Interfaces of IUI'05; 2005:167-174; Association for Computing Machinery, San Diego, CA.
58. Parasuraman R, Sheridan TB, Wickens CD. A model for types and levels of human interaction with automation. *IEEE Trans Syst Man Cybern A Syst Humans*. 2000;30(3):286-297. doi:10.1109/3468.844354
59. Vagia M, Transeth AA, Fjerdingen SA. A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed? *Appl Ergon*. 2016;53:190-202.
60. Bailey BP, Konstan JA, Carlis JV. The effects of interruptions on task performance, annoyance, and anxiety in the user interface. Proceedings of the International Conference on Human-Computer Interaction; 2001:593-601.
61. Parnin C, Rugaber S. Resumption strategies for interrupted programming tasks. Proceedings of the 2009 IEEE 17th International Conference on Program Comprehension; 2009:80-89.
62. Bahmanziari T, Pearson JM, Crosby L. Is trust important in technology adoption? A policy capturing approach. *J Comput Inf Syst*. 2003;43(4):46-54.
63. Chopra K, Wallace W. Trust in electronic environments. Proceedings of the 36th Hawaii International Conference on System Sciences; 2003; IEEE.
64. Hall R. Trusting your assistant. Proceedings of the 11th Knowledge-Based Software Engineering Conference; 1996:42-51.
65. Hertzum M. The importance of trust in software engineers' assessment and choice of information sources. *Inf Org*. 2002;12(1):1-18.
66. Feldt R, de Oliveira Neto FG, Torkar R. Ways of applying artificial intelligence in software engineering. Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering; 2018:35-41.
67. Nielsen J. *End of Web Design*. Nielsen Norman Group; 2000.

68. Burnett M. What is end-user software engineering and why does it matter? In: Pipek V, Rosson MB, de Ruyter B, Wulf V, eds. *End-User Development*. Springer; 2009:15-28.
69. Ko AJ, Abraham R, Beckwith L, et al. The state of the art in end-user software engineering. *ACM Comput Surv*. 2011;43(3):1-44. doi:10.1145/1922649.1922658
70. Barricelli BR, Cassano F, Fogli D, Piccinno A. End-user development, end-user programming and end-user software engineering: a systematic mapping study. *J Syst Softw*. 2019;149:101-137. doi:10.1016/j.jss.2018.11.041
71. Sanctorum A, Rukonic L, Signer B. Design requirements for recommendations in end-user user interface design. In: Fogli D, Tetteroo D, Barricelli BR, Borsci S, Markopoulos P, Papadopoulos GA, eds. *End-User Development*. Springer International Publishing; 2021:204-212.
72. Spahn M, Dörner C, Wulf V. End user development: approaches towards a flexible software design. In: Golden W, Acton T, Conboy K, van de Heijden H, Tuunainen VK, eds. *16th European Conference on Information Systems, ECIS 2008*; Association for Information Systems AIS Electronic Library (AISeL); 2008:303-314.

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.

How to cite this article: Savary-Leblanc M, Burgueño L, Cabot J, Le Pallec X, Gérard S. Software assistants in software engineering: A systematic mapping study. *Softw: Pract Exper*. 2023;53(3):856-892. doi: 10.1002/spe.3170

APPENDIX A. PRIMARY STUDIES

- [S1] Asaduzzaman M, Roy CK, Schneider KA, Hou D. FEMIR: A tool for recommending framework extension examples. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017: 967-972.
- [S2] El-Hajj R, Nadi S. *LibComp: An IntelliJ Plugin for Comparing Java Libraries*: 1591-1595; Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery, 2020.
- [S3] He H, Xu Y, Cheng X, Liang G, Zhou M. MigrationAdvisor: Recommending Library Migrations from Large-Scale Open-Source Data. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2021: 9-12.
- [S4] Chen C, Xing Z. SimilarTech: Automatically recommend analogical libraries across different programming languages. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), 2016: 834-839.
- [S5] Chen C. SimilarAPI: Mining Analogical APIs for Library Migration. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2020: 37-40.
- [S6] Uddin G, Khomh F. Opiner: An opinion search and summarization engine for APIs. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2017: 978-983.
- [S7] Brandt J, Dontcheva M, Weskamp M, Klemmer SR. Example-centric programming: integrating web search into the development environment. In: CHI'10. Association for Computing Machinery; 2010; New York, NY, USA: 513-522.
- [S8] Campbell BA, Treude C. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017: 628-632.
- [S9] Abid S, Shamail S, Basit HA, Nadi S. FACER: An API usage-based code-example recommender for opportunistic reuse. *Empirical Software Engineering* 2021; 26(6): 110. doi:10.1007/s10664-021-10000-w
- [S10] Jiang H, Nie L, Sun Z, et al. ROSF: Leveraging Information Retrieval and Supervised Learning for Recommending Code Snippets. *IEEE Transactions on Services Computing* 2019; 12(1): 34-46. doi:10.1109/TSC.2016.2592909
- [S11] Zhou Y, Jin H, Yang X, Chen T, Narasimhan K, Gall HC. *BRAID: An API Recommender Supporting Implicit User Feedback*: 1510-1514; Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery. 2021.

- [S12] Xu C, Min B, Sun X, Hu J, Li B, Duan Y. MULAPI: A Tool for API Method and Usage Location Recommendation. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2019: 119-122. ISSN: 2574-1934.
- [S13] Yuan W, Nguyen HH, Jiang L, Chen Y. LibraryGuru: API recommendation for Android developers. In: ICSE'18. Association for Computing Machinery; 2018; New York, NY, USA: 364-365.
- [S14] Wang L, Fang L, Wang L, Li G, Xie B, Yang F. APIExample: An effective web search based usage example recommendation system for java APIs. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011: 592-595. ISSN: 1938-4300.
- [S15] Cai L, Wang H, Huang Q, Xia X, Xing Z, Lo D. BIKER: a tool for Bi-information source based API method recommendation. In: ESEC/FSE 2019. Association for Computing Machinery; 2019; New York, NY, USA: 1075-1079.
- [S16] Yu H, Song W, Mine T. APIBook: An Effective Approach for Finding APIs. In: Proceedings of the 8th Asia-Pacific Symposium on Internetware. Association for Computing Machinery; 2016; New York, NY, USA: 45-53.
- [S17] Bajracharya S, Ossher J, Lopes C. Searching API Usage Examples in Code Repositories with Sourcerer API Search. In: Proceedings of 2010 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools and Evaluation. Association for Computing Machinery; 2010; New York, NY, USA: 5-8.
- [S18] Silva dRFG, Roy CK, Rahman MM, et al. CROKAGE: effective solution recommendation for programming tasks by leveraging crowd knowledge. *Empirical Software Engineering* 2020; 25(6): 4707-4758. doi:10.1007/s10664-020-09863-2
- [S19] Zagalsky A, Barzilay O, Yehudai A. Example Overflow: Using social media for code recommendation. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), 2012: 38-42. ISSN: 2327-0942.
- [S20] McMillan C, Hariri N, Poshvanyk D, Cleland-Huang J, Mobasher B. Recommending source code for use in rapid software prototypes. In: 2012 34th International Conference on Software Engineering (ICSE), 2012: 848-858. ISSN: 1558-1225.
- [S21] Nguyen TT, Pham HV, Vu PM, Nguyen TT. Recommending API Usages for Mobile Apps with Hidden Markov Model. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015: 795-800.
- [S22] Zhang C, Yang J, Zhang Y, et al. Automatic parameter recommendation for practical API usage. In: 2012 34th International Conference on Software Engineering (ICSE), 2012: 826-836. ISSN: 1558-1225.
- [S23] Santos AL, Prendi G, Sousa H, Ribeiro R. Stepwise API usage assistance using n-gram language models. *Journal of Systems and Software* 2017; 131: 461-474. doi:10.1016/j.jss.2016.06.063
- [S24] Nguyen AT, Nguyen HA, Nguyen TT, Nguyen TN. GraPacc: A graph-based pattern-oriented, context-sensitive code completion tool. In: 2012 34th International Conference on Software Engineering (ICSE), 2012: 1407-1410. ISSN: 1558-1225.
- [S25] Duala-Ekoko E, Robillard MP. Using Structure-Based Recommendations to Facilitate Discoverability in APIs. In: *Lecture Notes in Computer Science*. Springer; 2011; Berlin, Heidelberg: 79-104.
- [S26] Lin Y, Peng X, Xing Z, Zheng D, Zhao W. Clone-Based and Interactive Recommendation for Modifying Pasted Code. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. Association for Computing Machinery; 2015; New York, NY, USA: 520-531.
- [S27] Rahman MM, Roy CK. On the Use of Context in Recommending Exception Handling Code Examples. In: 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation, 2014: 285-294.
- [S28] Mooty M, Faulring A, Stylos J, Myers BA. Calcite: Completing Code Completion for Constructors Using Crowds. In: 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, 2010: 15-22.
- [S29] Lehmann J, Böhmann L. AutoSPARQL: Let Users Query Your Knowledge Base. In: Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I. Springer-Verlag; 2011; Berlin, Heidelberg: 63-79.
- [S30] Luan S, Yang D, Barnaby C, Sen K, Chandra S. Aroma: Code Recommendation via Structural Code Search. *Proc. ACM Program. Lang.* 2019; 3(OOPSLA). doi:10.1145/3360578
- [S31] Takuya W, Masuhara H. A spontaneous code recommendation tool based on associative search. In: SUITE'11. Association for Computing Machinery; 2011; New York, NY, USA: 17-20.

- [S32] Ai L, Huang Z, Li W, Zhou Y, Yu Y. SENSORY: Leveraging Code Statement Sequence Information for Code Snippets Recommendation. In: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2019: 27-36.
- [S33] Abid S, Javed S, Naseem M, Shahid S, Basit HA, Higo Y. Codeease: harnessing method clone structures for reuse. In: 2017 IEEE 11th International Workshop on Software Clones (IWSC), 2017: 1-7.
- [S34] Hsu SK, Lin SJ. MACs. *Expert Syst. Appl.* 2011; 38(6): 7291-7301. doi:10.1016/j.eswa.2010.12.021
- [S35] Lv F, Zhang H, Lou Jg, Wang S, Zhang D, Zhao J. CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015: 260-270.
- [S36] Vassallo C, Panichella S, Di Penta M, Canfora G. CODES: Mining Source Code Descriptions from Developers Discussions. In: Proceedings of the 22nd International Conference on Program Comprehension. Association for Computing Machinery; 2014; New York, NY, USA: 106-109.
- [S37] Gombos G, Kiss A. SPARQL Query Writing with Recommendations Based on Datasets. In: Human Interface and the Management of Information. Information and Knowledge Design and Evaluation. Springer International Publishing; 2014; Cham: 310-319.
- [S38] Lieber T, Brandt JR, Miller RC. Addressing misconceptions about code with always-on programming visualizations. In: CHI'14. Association for Computing Machinery; 2014; New York, NY, USA: 2481-2490.
- [S39] Cito J, Leitner P, Bosshard C, Knecht M, Mazlami G, Gall HC. PerformanceHat: augmenting source code with runtime performance traces in the IDE. In: ICSE'18. Association for Computing Machinery; 2018; New York, NY, USA: 41-44.
- [S40] Beck F, Moseler O, Diehl S, Rey GD. In situ understanding of performance bottlenecks through visually augmented code. In: 2013 21st International Conference on Program Comprehension (ICPC), 2013: 63-72. ISSN: 1092-8138.
- [S41] Lopez N, Hoek v. dA. The code orb: supporting contextualized coding via at-a-glance views (NIER track). In: 2011 33rd International Conference on Software Engineering (ICSE), 2011: 824-827.
- [S42] Swift B, Sorensen A, Gardner H, Hosking J. Visual code annotations for cyberphysical programming. In: 2013 1st International Workshop on Live Programming (LIVE), 2013: 27-30.
- [S43] Murphy-Hill E, Black AP. Programmer-Friendly Refactoring Errors. *IEEE Trans. Softw. Eng.* 2012; 38(6): 1417-1431. doi:10.1109/TSE.2011.110
- [S44] Omar C, Yoon YS, LaToza TD, Myers BA. Active code completion. In: 2012 34th International Conference on Software Engineering (ICSE), 2012: 859-869.
- [S45] Subramanian S, Inozemtseva L, Holmes R. Live API Documentation. In: Proceedings of the 36th International Conference on Software Engineering. Association for Computing Machinery; 2014; New York, NY, USA: 643-652.
- [S46] Chen X, Chen C, Zhang D, Xing Z. SEthesaurus: WordNet in Software Engineering. *IEEE Transactions on Software Engineering* 2021; 47(9): 1960-1979. doi:10.1109/TSE.2019.2940439
- [S47] LaToza TD, Myers BA. Visualizing call graphs. In: 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011: 117-124. ISSN: 1943-6106.
- [S48] Bauer V, Heinemann L. Understanding API Usage to Support Informed Decision Making in Software Maintenance. In: 2012 16th European Conference on Software Maintenance and Reengineering, 2012: 435-440. ISSN: 1534-5351.
- [S49] Oney S, Brandt J. Codelets: Linking Interactive Documentation and Example Code in the Editor. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Association for Computing Machinery; 2012; New York, NY, USA: 2697-2706.
- [S50] Ko AJ, Myers BA. Extracting and answering why and why not questions about Java program output. *ACM Transactions on Software Engineering and Methodology* 2010; 20(2): 4:1-4:36. doi:10.1145/1824760.1824761
- [S51] Fowkes J, Chanthirasegaran P, Ranca R, Allamanis M, Lapata M, Sutton C. TASSAL: autofolding for source code summarization. In: ICSE'16. Association for Computing Machinery; 2016; New York, NY, USA: 649-652.
- [S52] Karrer T, Krämer JP, Diehl J, Hartmann B, Borchers J. Stackplorer: call graph navigation helps increasing code maintenance efficiency. In: UIST'11. Association for Computing Machinery; 2011; New York, NY, USA: 217-224.
- [S53] Robillard MP, Chhetri YB. Recommending Reference API Documentation. *Empirical Softw. Engg.* 2015; 20(6): 1558-1586. doi:10.1007/s10664-014-9323-y
- [S54] Duruisseau M, Tarby JC, Le Pallec X, Gérard S. VisUML: A Live UML Visualization to Help Developers in Their Programming Task. In: Lecture Notes in Computer Science. Springer International Publishing; 2018; Cham: 3-22.

- [S55] Lee S, Kang S, Staats M. NavClus: A graphical recommender for assisting code exploration. In: 2013 35th International Conference on Software Engineering (ICSE), 2013: 1315-1318. ISSN: 1558-1225.
- [S56] Antunes B, Cordeiro J, Gomes P. An Approach to Context-Based Recommendation in Software Development. In: Proceedings of the Sixth ACM Conference on Recommender Systems. Association for Computing Machinery; 2012; New York, NY, USA: 171-178.
- [S57] Viriyakattiyaporn P, Murphy GC. Improving program navigation with an active help system. In: CASCON'10. IBM Corp.; 2010; USA: 27-41.
- [S58] Surian D, Liu N, Lo D, Tong H, Lim EP, Faloutsos C. Recommending People in Developers' Collaboration Network. In: 2011 18th Working Conference on Reverse Engineering, 2011: 379-388. ISSN: 2375-5369.
- [S59] Teyton C, Falleri JR, Morandat F, Blanc X. Find your library experts. In: 2013 20th Working Conference on Reverse Engineering (WCRE), 2013: 202-211. ISSN: 2375-5369.
- [S60] Lee B, Srivastava S, Kumar R, Brafman R, Klemmer SR. Designing with interactive example galleries. In: CHI'10. Association for Computing Machinery; 2010; New York, NY, USA: 2257-2266.
- [S61] Lucrédio D, M. Fortes dRP, Whittle J. MOOGLE: a metamodel-based model search engine. *Software & Systems Modeling* 2012; 11(2): 183-208. doi:10.1007/s10270-010-0167-7
- [S62] Casamayor A, Godoy D, Campo M. Functional grouping of natural language requirements for assistance in architectural software design. *Knowledge-Based Systems* 2012; 30: 78-86. doi:10.1016/j.knosys.2011.12.009
- [S63] Almonte L, Pérez-Soler S, Guerra E, Cantador I, Lara dJ. *Automating the Synthesis of Recommender Systems for Modelling Languages: 22-35*; Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering. New York, NY, USA: Association for Computing Machinery. 2021.
- [S64] Agt-Rickauer H, Kutsche RD, Sack H. DoMoRe? A Recommender System for Domain Modeling. In: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development. SCITEPRESS - Science and Technology Publications, Lda; 2018; Setubal, PRT: 71-82.
- [S65] Pati T, Kolli S, Hill JH. Proactive Modeling: A New Model Intelligence Technique. *Softw. Syst. Model.* 2017; 16(2): 499-521. doi:10.1007/s10270-015-0465-1
- [S66] Savary-Leblanc M, Le-Pallec X, Gérard S. A modeling assistant for cognifying MBSE tools. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021: 630-634.
- [S67] Koschmider A, Hornung T, Oberweis A. Recommendation-based editor for business process modeling. *Data & Knowledge Engineering* 2011; 70(6): 483-503. doi:10.1016/j.datak.2011.02.002
- [S68] Elkamel A, Gzara M, Ben-Abdallah H. An UML class recommender system for software design. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), 2016: 1-8.
- [S69] Paydar S, Kahani M. A semantic web enabled approach to reuse functional requirements models in web engineering. *Autom. Softw. Eng.* 2015; 22(2): 241-288. doi:10.1007/s10515-014-0144-4
- [S70] Saini R, Mussbacher G, Guo JLC, Kienzle J. Automated, interactive, and traceable domain modelling empowered by artificial intelligence. *Software and Systems Modeling* 2022. doi:10.1007/s10270-021-00942-6
- [S71] Cuadrado JS, Guerra E, Lara dJ. AnATLyzer: An Advanced IDE for ATL Model Transformations. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings. Association for Computing Machinery; 2018; New York, NY, USA: 85-88.
- [S72] Aquino ER, Saqui-Sannes dP, Vingerhoeds RA. A Methodological Assistant for Use Case Diagrams. In: 8th MODELSWARD: International Conference on Model-Driven Engineering and Software Development, 2020; La Valette, MT: 1-11. ISBN: 978-989-758-400-8.
- [S73] Thies A, Roth C. Recommending rename refactorings. In: RSSE'10. Association for Computing Machinery; 2010; New York, NY, USA: 1-5.
- [S74] Mkaouer MW, Kessentini M, Bechikh S, Deb K, Ó Cinnéide M. Recommendation system for software refactoring using innovization and interactive dynamic optimization. In: ASE'14. Association for Computing Machinery; 2014; New York, NY, USA: 331-336.
- [S75] Vidal SA, Marcos CA. Building an expert system to assist system refactorization. *Expert Systems with Applications* 2012; 39(3): 3810-3816. doi:10.1016/j.eswa.2011.09.084
- [S76] Fokaefs M, Tsantalis N, Stroulia E, Chatzigeorgiou A. JDeodorant: identification and application of extract class refactorings. In: 2011 33rd International Conference on Software Engineering (ICSE), 2011: 1037-1039.

- [S77] Siles Antezana A. TOAD: A Tool for Recommending Auto-Refactoring Alternatives. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2019: 174-176. ISSN: 2574-1934.
- [S78] Ge X, Murphy-Hill E. Manual Refactoring Changes with Automated Refactoring Validation. In: Proceedings of the 36th International Conference on Software Engineering. Association for Computing Machinery; 2014; New York, NY, USA: 1095-1105.
- [S79] Ohrndorf M, Pietsch C, Kelter U, Kehrer T. ReVision: a tool for history-based model repair recommendations. In: ICSE'18. Association for Computing Machinery; 2018; New York, NY, USA: 105-108.
- [S80] UL Muram F, Gallina B, Gómez Rodríguez L. Preventing Omission of Key Evidence Fallacy in Process-Based Argumentations. In: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 2018: 65-73.
- [S81] Cordeiro J, Antunes B, Gomes P. Context-based recommendation to support problem solving in software development. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), 2012: 85-89. ISSN: 2327-0942.
- [S82] Rahman MM, Yeasmin S, Roy CK. Towards a context-aware IDE-based meta search engine for recommendation about programming errors and exceptions. In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014: 194-203.
- [S83] Kawalerowicz M, Madeyski L. Jaskier: A Supporting Software Tool for Continuous Build Outcome Prediction Practice. In: *Advances and Trends in Artificial Intelligence. From Theory to Practice - 34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2021, Kuala Lumpur, Malaysia, July 26-29, 2021, Proceedings, Part II*. Springer; 2021: 426-438.
- [S84] Muşlu K, Brun Y, Holmes R, Ernst MD, Notkin D. Improving IDE recommendations by considering global implications of existing recommendations. In: 2012 34th International Conference on Software Engineering (ICSE), 2012: 1349-1352. ISSN: 1558-1225.
- [S85] Saha RK, Yoshida H, Prasad MR, Tokumoto S, Takayama K, Nanba I. Elixir: an automated repair tool for Java programs. In: ICSE'18. Association for Computing Machinery; 2018; New York, NY, USA: 77-80.
- [S86] Zhou Y, Wang C, Yan X, Chen T, Panichella S, Gall H. Automatic Detection and Repair Recommendation of Directive Defects in Java API Documentation. *IEEE Transactions on Software Engineering* 2020; 46(9): 1004-1023. doi:10.1109/TSE.2018.2872971
- [S87] Yoshida H, Bavishi R, Hotta K, Nemoto Y, Prasad MR, Kikuchi S. *Phoenix: A Tool for Automated Data-Driven Synthesis of Repairs for Static Analysis Violations*: 53-56; Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. New York, NY, USA: Association for Computing Machinery. 2020.
- [S88] Barik T, Song Y, Johnson B, Murphy-Hill E. From Quick Fixes to Slow Fixes: Reimagining Static Analysis Resolutions to Enable Design Space Exploration. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016: 211-221.
- [S89] Zhang X, Zhu C, Li Y, Guo J, Liu L, Gu H. *Large-Scale Patch Recommendation at Alibaba*: 252-253; Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings. New York, NY, USA: Association for Computing Machinery. 2020.
- [S90] Hartmann B, MacDougall D, Brandt J, Klemmer SR. What would other programmers do: suggesting solutions to error messages. In: CHI'10. Association for Computing Machinery; 2010; New York, NY, USA: 1019-1028.
- [S91] Nguyen T, Vu P, Nguyen T. *Code Recommendation for Exception Handling*: 1027-1038; Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery. 2020.
- [S92] Kistner F, Beth Kery M, Puskas M, Moore S, Myers BA. Moonstone: Support for understanding and writing exception handling code. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2017: 63-71.
- [S93] Vassallo C, Proksch S, Zemp T, Gall HC. Every build you break: developer-oriented assistance for build failure resolution. *Empirical Software Engineering* 2020; 25(3): 2218-2257. doi:10.1007/s10664-019-09765-y
- [S94] Sawadsky N, Murphy GC, Jiresal R. Reverb: Recommending code-related web pages. In: 2013 35th International Conference on Software Engineering (ICSE), 2013: 812-821. ISSN: 1558-1225.

- [S95] Ponzanelli L, Scalabrino S, Bavota G, et al. Supporting Software Developers with a Holistic Recommender System. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017: 94-105. ISSN: 1558-1225.
- [S96] Kononenko O, Dietrich D, Sharma R, Holmes R. Automatically locating relevant programming help online. In: 2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2012: 127-134. ISSN: 1943-6106.
- [S97] Nguyen PT, Di Rocco J, Di Ruscio D, Di Penta M. CrossRec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software* 2020; 161: 110460. doi:10.1016/j.jss.2019.110460
- [S98] Bacchelli A, Ponzanelli L, Lanza M. Harnessing Stack Overflow for the IDE. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), 2012: 26-30. ISSN: 2327-0942.
- [S99] Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M. Mining StackOverflow to turn the IDE into a self-confident programming prompter. In: MSR 2014. Association for Computing Machinery; 2014; New York, NY, USA: 102-111.
- [S100] Campos EC, Souza dLB, Maia MdA. Nuggets Miner: Assisting Developers by Harnessing the StackOverflow Crowd Knowledge and the GitHub Traceability.
- [S101] Rubei R, Di Sipio C, Nguyen PT, Di Rocco J, Di Ruscio D. PostFinder: Mining Stack Overflow posts to support software developers. *Information and Software Technology* 2020; 127: 106367. doi:10.1016/j.infsof.2020.106367
- [S102] Hattori L, Lanza M. Syde: a tool for collaborative software development. In: ICSE'10. Association for Computing Machinery; 2010; New York, NY, USA: 235-238.
- [S103] Sarma A, Redmiles D, André Hoek v. d. Palantir: Early Detection of Development Conflicts Arising from Parallel Code Changes. *IEEE Transactions on Software Engineering* 2011; 38(4): 889- 908.
- [S104] Brun Y, Holmes R, Ernst MD, Notkin D. Proactive detection of collaboration conflicts. In: ESEC/FSE'11. Association for Computing Machinery; 2011; New York, NY, USA: 168-178.
- [S105] Shen B, Zhang W, Kästner C, et al. SmartCommit: A Graph-Based Interactive Assistant for Activity-Oriented Commits. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery; 2021; New York, NY, USA: 379-390.
- [S106] Holmes R, Walker RJ. Customized awareness: recommending relevant external change events. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010: 465-474.
- [S107] Abdellatif A, Badran K, Shihab E. MSRBot: Using bots to answer questions from software repositories. *Empirical Software Engineering* 2020; 25(3): 1834-1863. doi:10.1007/s10664-019-09788-5
- [S108] Mirsaedi E, Rigby PC. Mitigating Turnover with Code Review Recommendation: Balancing Expertise, Workload, and Knowledge Distribution. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Association for Computing Machinery; 2020; New York, NY, USA: 1183-1195.
- [S109] Rebai S, Amich A, Molaei S, Kessentini M, Kazman R. Multi-objective code reviewer recommendations: balancing expertise, availability and collaborations. *Autom. Softw. Eng.* 2020; 27(3): 301-328. doi:10.1007/s10515-020-00275-6
- [S110] Brosch P, Seidl M, Kappel G. A Recommender for Conflict Resolution Support in Optimistic Model Versioning. In: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion. Association for Computing Machinery; 2010; New York, NY, USA: 43-50.
- [S111] Niu N, Yang F, Cheng JRC, Reddivari S. A cost-benefit approach to recommending conflict resolution for parallel software development. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), 2012: 21-25.
- [S112] Paikari E, Choi J, Kim S, et al. A Chatbot for Conflict Detection and Resolution. In: 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE), 2019: 29-33.