Journal of Software Engineering Research and Development, 2025, 13:5, doi: 10.5753/jserd.2025.4494

The Influence of Search Engine Ranking Quality on the Performance of Programmers

Adriano M. Rocha [**Universidade Federal de Uberlândia** | ***adriano.rocha@ufu.br***]
Carlos E. C. Dantas [**Instituto Federal do Triângulo Mineiro** | ***carloseduardodantas@iftm.edu.br***]
Marcelo de Almeida Maia [**Universidade Federal de Uberlândia** | ***marcelo.maia@ufu.br***]

Abstract

Software development is an activity characterized by the continuous search for information by programmers. General-purpose search engines, such as Google, Bing, and Yahoo, are widely used by programmers to find solutions to their problems. However, the best solutions are not always among the initial pages of search results that are ranked according to the search engine's algorithms. This study aims to investigate the influence that the order of the pages returned by the ranking of search engines exerts on programmers' performance when performing programming tasks. We designed an empirical within-subject study with programmers to understand and evaluate their performance when solving programming tasks using a ranked list of pages returned by the Google search engine and artificially modified with two different ranking quality levels (Higher Quality Ranking and Lower Quality Ranking). Moreover, for each entry in the ranking of pages, the most frequent methods mentioned on the respective page were listed in the ranking visualization. Analysis of participants' recorded videos was conducted through a mixed-methods research approach to provide insights into the results. We found that programmers spent approximately 8 minutes longer resolving tasks associated with a Lower Quality Ranking, spending more time on irrelevant pages compared to relevant ones, due to efforts to fix problematic code or new searches for another page. The addition of a list of frequent methods in the ranking visualization could help programmers to skip irrelevant pages and reduce time wastage. The ranking quality influences the programmers' performance during the development of programming tasks. Therefore, we suggest the development of filters aimed at improving the quality of results delivered by search engines. Moreover, the results may encourage the adaptation of this study for other approaches that require information foraging, such as chatting with LLMs.

Keywords: *Software Reuse, Search Engines, Ranking Quality*

1 Introduction

Software development is an activity that requires a constant search for information by programmers (Sadowski et al., 2015). One challenge faced by programmers is the lack of code examples in the official documentation of software development technologies (Kim et al., 2010). Code examples help programmers to improve their comprehension of the technology (Nykaza et al., 2002; Robillard, 2009), as well as providing software reuse (Buse and Weimer, 2012; Johnson, 1992). An alternative widely used by programmers is to resort to the Web for solutions that have code examples to solve problems related to their daily software activities (Stolee et al., 2014; Sim et al., 2011; Gallardo-Valencia and Elliott Sim, 2009; Kim et al., 2010). Search engines, such as Google, Bing, and Yahoo, help programmers to find pages with solutions to their problems (Hora, 2021b; Niu et al., 2017; Fischer et al., 2021). However, the best solutions are not always among the first ones returned by these search engines (Chatterjee et al., 2009; Hora, 2021a). Due to the lack of quality of top-ranked pages, programmers may spend a significant amount of time searching for the desired solution (Xia et al., 2017).

Common sense may suggest that programmers would allocate additional time to complete tasks when they search the web for information but do not find relevant code examples in the upper pages of the search engine rankings. However, whether the extra time spent finding a useful page among the search results is negligible or not, still seems to be an open question. Therefore, we argue that it is important to investigate the impact of these rankings on programmer performance to decide whether improved ranking mechanisms are necessary in the field of software development. Additionally, understanding how programmers navigate irrelevant top-ranked pages containing unsuitable code examples for their tasks is essential.

1.1 Motivating Example

As a motivating example, one of the participants in our experiment wanted to find a code example to insert the user-entered information from the form fields into the database. After searching the available pages for that code, the participant verified that the first four pages returned by the search engine did not contain relevant code for the intended solution.

In this scenario, the participant first analyzed the title and description of the first page in the search results and decided to click on it. Afterward, the participant examined the page's content and found that there was no relevant code (105 seconds were spent analyzing the search results and page content). The participant then returned to the search results and analyzed the titles and descriptions of the pages, deciding to click on the second page. Upon analyzing the content, the participant copied a portion of the source code from the page and pasted it into their Integrated Development Environment

(IDE) (211 seconds were spent analyzing the search results and page content).

Next, the participant began editing the copied code from the website. After making edits, the participant tried to compile the code and discovered errors in the copied code (38 seconds were spent editing and compiling the code). The participant decided to go back to the search results and analyzed the descriptions of the pages, opting to click on the third page. After analyzing the page's content, the participant found no relevant content that could be used in the intended solution (33 seconds were spent analyzing the search results and page content).

The participant then returned to the search results and analyzed the page titles and descriptions, deciding to click on the fourth page. On this page, the participant found a code snippet and used it to try to fix the error in the code obtained from the second page. After several edits and tests, the programmer could not fix the error (175 seconds were spent editing and testing the code).

Subsequently, the participant decided to return to the search results and analyzed the titles and descriptions of the pages again. Then, the participant clicked on the fifth page. After analyzing the page's content, the participant copied a code snippet from the page, removed the code snippets obtained from the previous pages, and pasted the code snippet from the fifth page into the IDE. After some edits, the programmer compiles and tests the program, which works as desired.

In this motivating example, the participant spent 562 seconds (9 minutes and 22 seconds) unnecessarily until finding the solution on the fifth page of the search engine's results. According to the study conducted in this research, it is possible that the ranking of a relevant page is not high, thus making it even more challenging for the programmer to construct the desired solution. Figure 1 illustrates the top five available pages for the participant to implement the task.

Figure 1. The top five available pages in the ranking for the participant to implement the task.

1.2 Research Questions

Considering the previously mentioned challenges, this study aims to investigate the influence that the order of the pages returned by the ranking of search engines exerts on programmers' performance when performing programming tasks. The objective of the present study is to answer the following research questions:

RQ1: *Does the ranking quality of query results have any influence on the performance of programmers in the development of programming tasks?*

RQ2: *Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for programmers?*

RQ3: *How do irrelevant pages present in lower quality rankings hinder programmers' goal of finding relevant solutions?*

To the best of our knowledge, we have not encountered any prior research that explores the impact of search engine result rankings on programmers' performance during programming tasks. In our findings, when programmers encounter irrelevant pages in the top-ranked results, they not only invest more time in solving the task but also waste valuable time trying to fix inappropriate code extracted from these irrelevant pages. Conversely, when programmers find relevant pages among the top-ranked results, they require less time to complete the task, and they also visit fewer pages. This underscores the significance of search engine result rankings for programmers.

This article is structured as follows. Section 2 presents the methodology to answer the research questions. The results are reported in Section 3. Section 4 discusses the results. Section 5 addresses threats to validity. Section 6 explores related literature. Section 7 concludes the article and directions for future work.

2 Study Setting

To answer the research questions, we designed a user study involving programmers, aiming to assess their performance while solving programming tasks. We employed a mixed-methods research approach, blending both qualitative and quantitative methodologies. This assessment involved the ranking of pages returned by the search engine with two distinct levels of quality: Higher Quality Ranking and Lower Quality Ranking. The general idea of the study encompassed the engagement of programmers in the execution of pairs of programming tasks, so the comparison of the performance was within-programmer. The quality of ranking was varied within the pair of tasks, while striving to maintain consistency across other influencing factors, such as the task complexity. Therefore, the study's methodology encompassed

the presentation of programmer pairs with two tasks each – the first and second tasks shared a comparable level of complexity, as did the third and fourth tasks. This within-subject design enabled a direct comparison of programmer performance under differing ranking quality scenarios. The overall approach for answering the three research questions is illustrated in Figure 2.

Recordings of participants' screens were captured. Subsequently, an in-depth analysis of the recorded video material was undertaken, in order to verify the programmers' performance within varied ranking quality scenarios. To establish an appropriate time threshold for task completion, the authors of this paper conducted the tasks themselves, measuring the time taken for each. This yielded an average solution time of approximately 30 minutes. Consequently, the predefined time limit for task execution was set at twice this average duration, affording participants a one-hour time limit per task. Before solving the proposed tasks, the participants were instructed not to use any other sources than the pages provided by the researchers. They were also instructed not to click on links on these pages. For each programming task, the programmers also answered a form containing questions related to the available pages for solving the programming task.

2.1 Definition of Programming Tasks for the Study

To conduct this study, we selected four Java programming tasks that exclusively employed native APIs. The selection of these four tasks was strategically chosen in order to make pairwise comparisons between pairs of tasks with similar difficulty levels and effort to execute the tasks. This approach enabled us to isolate the impact of ranking quality on programmer performance by varying only this single variable, while keeping other factors constant. We deliberately focused on native APIs to ensure a smoother training phase for participants, avoiding the complexities that could arise from non-native APIs. For the first and second tasks, we opted for the Swing API, whereas for the third and fourth tasks, we chose the JDBC API. These API choices were driven by their alignment with our study's goals and the ease of configuration during participant training.

To choose pairs of similar tasks, that is, similar first and second tasks, as well as the third and fourth tasks, we searched the web for tutorials containing programming tasks centered around the Swing and JDBC APIs. After the identification of two pairs of programming tasks, we refined and adapted these tasks to enhance their similarity. In order to carry out these adaptations, we planned to make the number of modifications performed by the programmers while solving the pairs of programming tasks remained as similar as possible. To achieve this, we built tables to compare the technical elements (such as classes, methods, and control structures) manipulated by the programmers during the resolution of the proposed programming tasks. Table 1 illustrates the comparison of these technical elements, highlighting the similarities between the technical demands that will have to be handled by the programmers while addressing the challenges presented in the first and second tasks.

Table 1. Comparison of the technical elements handled during the resolutions of the first and second tasks.

Technical Elements	Task 1	Task 2
# Methods Inserted	10	14
# Methods Changed	4	3
# Methods Removed	3	3
# Classes Inserted	3	3
# Changed Classes	3	3
# Classes Removed	3	3
# IFs	3	3
# Variables Inserted	4	0
Totals of Changes in Source Code	**33**	**32**

As illustrated in Table 1, the Technical Elements column outlines the technical elements that need to be handled by programmers while performing the proposed programming tasks. The Task 1 and Task 2 columns present the number of additions, modifications, and removals of methods and classes, along with the number of introductions of IF structures and variables, which programmers need to perform in order to solve the respective task schedules. The number of changes required for programmers is quite comparable, there are 33 changes in the source code of the first task and 32 changes for the second task. We assessed that the level of complexity in manipulating variables, methods, classes, and IF structure are quite similar.

Table 2 shows the comparison of the technical elements that programmers need to address during the resolutions of the third and fourth tasks. Both programming tasks require a similar number of changes in the source code, precisely 10 modifications each. Subsequently, we produced a detailed description of the four programming tasks proposed in the study, all of them involving the modification of a partially implemented Java project. The Java projects used in the study can be obtained through a Zenodo repository[1].

Table 2. Comparison of the technical elements handled during the resolutions of the third and fourth tasks.

Technical Elements	Task 3	Task 4
# Methods Inserted	7	9
# Methods Changed	0	0
# Methods Removed	0	0
# Classes Inserted	2	1
# Changed Classes	0	0
# Classes Removed	0	0
# IFs	1	0
# Variables Inserted	0	0
Totals of Changes in Source Code	**10**	**10**

The **first task** is defined as a user registration system, containing fields *Name*, *Cell Phone Number*, and three radio-type options enabling users to select their preferred game type. There are two buttons below the fields: *Submit* and *Reset*. Upon clicking the *Submit* button, the information entered in the *Name* and *Cell Phone* fields is displayed in a designated print area adjacent to the fields. The *Reset* button clears all information entered in the fields. The objective of this task is to replace the radio-type options with checkboxes and subsequently exhibit the selected options within the print area.

The **second task** contains a chat application configuration system, which includes three configuration options: au-

[1] https://doi.org/10.5281/zenodo.14009184

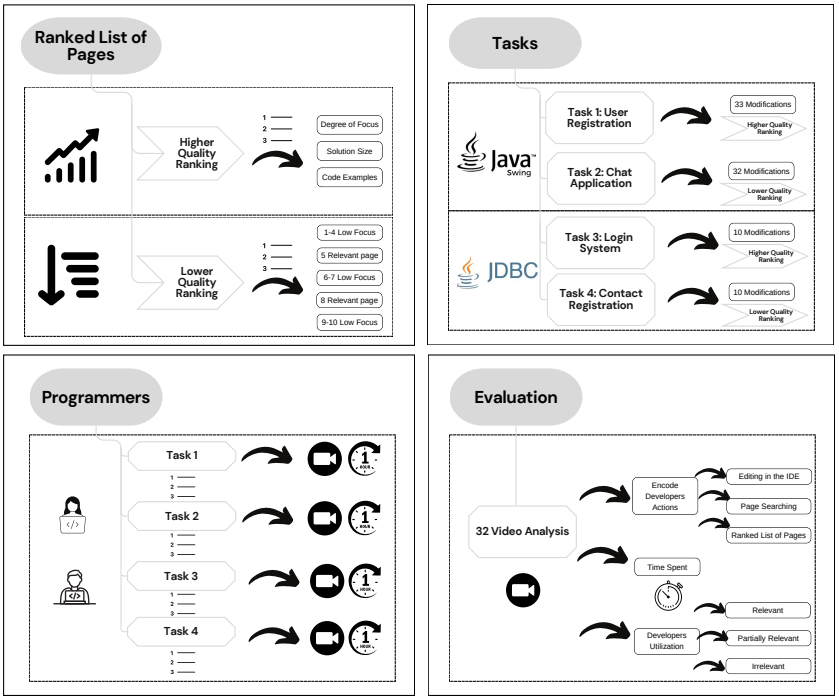

Figure 2. Overall proposed architecture

dio settings, automatic download preferences, and storing conversation history. Each option is accompanied by a button labeled "Enable". When users click these buttons, no action is triggered. The proposed task requires changing these JButton-type to toggle buttons and updating the button text to "Disable" when pressed by the user.

The **third task** encompasses a login system with *Username* and *Password* fields. Currently, upon clicking the login button, no action occurs. The objective of this task is to implement the necessary code to verify whether the information entered into the *Username* and *Password* fields matches the database records. Additionally, it involves incorporating two distinct messages. If the entered information aligns with the database records, the message "Correct login." should be displayed on the screen; otherwise, the message "Incorrect username or password." should appear.

The **fourth task** contains a contact registration system, with the following fields: *Name*, *Telephone Number*, *E-mail*, *Address*, *City* and *State*. This system also has a button labeled "Register", which has no action when pressed. The objective of this task is to implement the necessary code to insert the user-entered information from the form fields into the database. Upon successful data insertion, the system should display the message "Contact registered successfully." on the screen.

It is worth noting that the order of tasks in the study does not influence the analysis, since different tasks were intentionally designed to avoid potential learning effects. Each task is distinct, ensuring that prior experiences with one task do not influence the performance or analysis of the subsequent ones.

2.2 Building Higher Quality and Lower Quality Rankings

The ranked list of pages that are presented to the participants is similar to the one shown in Figure 1, resembling a typical list of pages returned by a search engine. However, one key difference is that a list of frequently mentioned methods on each page appears at the end of its description.

Following, we describe how the ranking was constructed.

To create the queries, we use the following structure: "how to" + programming task + programming language + API(s). The inclusion of the "how to" phrase serves as a filter, allowing us to retrieve pages that pertain to the implementation of specific functionalities within the program (Nasehi et al., 2012). This filter prevents pages with content related to the debug-corrective type from being returned by the search engine, aligning with our study's primary objective of sourcing material that aids programmers in implementing solutions to programming tasks. To obtain the content needed to solve the programming task, we incorporate the sentence associated with the proposed task within the query construction. This inclusion drives the search engine to return content addressing the specific challenges programmers encounter in solving the task. Furthermore, we specify the programming language and API names within the query to access content related to these crucial elements employed in the proposed task. Regarding the order of sentences in the query and the addition or omission of specific terms, these nuances have a minimal impact on the search engine results, as demonstrated in previous work (Hora, 2021b).

Following the aforementioned method, we built the following queries and obtained the list of the top-20 pages returned from Google, for each of the four proposed programming tasks.

1. *how to insert multiple checkboxes in form Java swing.*
2. *how to create switch button Java swing.*
3. *how to create login with database Java swing jdbc.*
4. *how to insert data in database Java swing jdbc.*

In the construction of the Higher Quality Ranking, we examined the first 20 pages returned by Google for each of the proposed programming tasks. Our evaluation was based on the criteria of Degree of Focus, Solution Size, and Code Examples, as proposed in our previous work (Rocha and Maia, 2023). Pages evaluated using the Degree of Focus criterion can contain one of the following values.

1. *Very low, solutions that are not related to the search performed by the user.*
2. *Low, solutions that have some connection with the search performed by the user.*
3. *Neutral, when it is not possible to evaluate the page for this criterion.* Neutral evaluations are selected when participants have no positive or negative opinion on the criterion.
4. *High, solutions that are related to the search, however, it has minor focus deviation.*
5. *Very high, solutions completely related to the search performed by the user.*

For the Solution Size criterion, pages could be categorized as follows, based on the number of lines of source code observed in the solutions:

1. *Very small, far below the observed average size.*
2. *Small, marginally below observed the average size.*
3. *Neutral, when it is not possible to evaluate the page for this criterion.*
4. *Large, marginally above the observed average size.*
5. *Very large, far above the observed average size.*

Under the Code Example criterion, pages were assigned a binary value, either *"yes" or "no"*, depending on the presence or absence of code examples in their solutions. After the evaluation of solutions on these pages, we ranked them based on the above criteria. Relevant pages, i.e. pages that exhibit a very high degree of focus, incorporating code examples, and showcasing solution sizes closely aligned with the average size of solutions encountered within the top 20 pages (as assessed by small or large values for the Solution Size criterion) were positioned at the top of the ranking.

To build the Lower Quality Ranking, we first carried out a study with 14 different queries, given as input in the Google search engine, in order to verify in which position of ranking the relevant page is located. These relevant pages were characterized by having a very high degree of focus, code examples, and solution sizes approximating the average size of solutions found within the top 20 pages. Among the 14 queries analyzed, we considered the 10 queries that had been selected in our prior work (Rocha and Maia, 2023), along with the four queries introduced in this study. Table 3 presents the 14 queries with their respective positions at which relevant pages appeared in the rankings. To determine the placement of relevant pages within the Lower Quality Ranking, we calculate the median of the positions where these pages appeared, excluding queries where the relevant page occurs in the first position, as such queries are considered as Higher Quality Ranking, we find the median equals to **five**. Consequently, the structure of the Lower Quality Ranking is as follows: In the first four positions, we feature pages with low-focus solutions, i.e. irrelevant pages (solutions unrelated to the user's query). In the fifth position, we position a relevant page. Moving to the sixth and seventh positions, we include pages containing low-focus solutions (irrelevant pages). The eighth position is reserved for another relevant page, followed by the ninth and tenth positions occupied by pages hosting low-focus solutions (irrelevant pages), and so forth. The ranked pages used in the study can be accessed from a Zenodo repository[2].

Table 3. Positions at which the relevant pages are located.

Queries	Position
How to insert multiple checkboxes in form Java swing	10
How to create switch button Java swing	4
How to create login with database Java swing jdbc	4
How to insert data in database Java swing jdbc	4
How to upload image Java spring	15
How to implement crud operations Java jpa	1
How to search a product item on an e-commerce web application Java selenium	4
How to implement menu Java Javafx	1
How to build a neural network Java tensorflow	5
How to classify image Java opencv	5
How to implement matchers test Java junit	15
How to process JSON data Java jackson	7
How to implement animation Java libgdx	3
How to draw 2D objects Java opengl	1
Median	5

It is worth noting that the sequence of higher and lower quality rankings followed by the participants in the study is unlikely to influence the results, given the conditions of the study. Participants are unaware of whether the task they are performing is associated with a higher quality ranking or a lower quality ranking. Consequently, the rankings are independent of each other, and the order in which tasks are presented does not affect the outcome. Since the tasks are different and presented pages are different for each task, there is no inherent influence on the results. If the pages were identical, there might be an impact, but under the current conditions, the sequence does not have an effect.

2.3 Recruitment for the Study

The study's target participants comprised undergraduate students currently enrolled in Object-Oriented Programming and Mobile Device Programming courses, as well as graduate students in Software Engineering, all from the Federal University of Uberlândia. We selected these individuals due to their proximity and relevance to the study. Invitations to participate in the study were extended to these students via email. The group of graduate students in Software Engineering consisted of 19 members, while the group of undergraduate students enrolled in Object-Oriented Programming and

[2]https://doi.org/10.5281/zenodo.14009184

Mobile Device Programming courses had 88 members. In total, eight participants agreed to participate in the study, three of them belonging to the group of graduate students and five participants belonging to the group of undergraduate students.

2.4 Ranking Evaluation

In order to evaluate the rankings related to the programming tasks proposed in the study, we created a form containing the following multiple-choice and open-ended questions. In multiple-choice questions, participants select one of five options on a Likert scale, while in open-ended questions, participants are encouraged to provide free-text responses. For each of the proposed programming tasks, participants were requested to complete the questionnaire after either completing the task or reaching the designated time limit. The questionnaire includes the following four questions:

1. Regarding navigability by ranking, how agile/productive was it to find the solution for the programming task?
   (a) **Very agile/productive.** The solution was easily found among the first pages of the ranking. That is, you browsed one to two pages of the ranking to find the solution.
   (b) **Agile/productive.** The solution was found among the first pages of the ranking, but it was necessary to visit some pages to find it. That is, you browsed three to five pages to find the solution.
   (c) **Neutral.** This option is chosen when you have no positive or negative opinion on the criterion.
   (d) **Not agile/productive.** The solution was found among the ranking pages, but it was necessary to visit several ranking pages to find it. That is, you browsed from six to ten pages to find the solution.
   (e) **Very little agile/productive.** The solution was found after a lot of effort, browsing through most of the ranking pages or the solution was not found among the ranking pages. That is, you browsed eleven or more pages to find the solution, or the solution was not found after browsing through the pages.
2. Did the summary provided below the link, which highlights the five most frequently occurring methods on the pages, relevant to your decision to click on the page?
   (a) **Very relevant.** The decision to click on the page or not was solely based on the content of the method list. In other words, only the method list itself influenced the decision, and other content in the link and page description was irrelevant.
   (b) **Relevant.** Both the content of the method list and the content of the link and page description contributed to the decision to click on the page or not.
   (c) **Neutral.** This option is chosen when you have no positive or negative opinion on the criterion.
   (d) **Irrelevant.** The contents of the method list were read but did not influence the decision to click on the page. In other words, the decision was not based on the method list.
   (e) **Very irrelevant.** The contents of the method list were not considered at all when deciding whether to click on the page. This means that the method list was not read, and it did not impact the decision to click on the page or not.
3. Regarding the pages you used to solve the programming task, what were your positive points? What did they have that helped you solve the task?
   The purpose of this question is to collect data on the characteristics of relevant pages (pages that were used to solve the proposed task).
4. Regarding the pages that you did NOT use to solve the programming task, what were your negative points? What did they NOT have that did NOT help you solve the task?
   The purpose of this question is to collect data on the characteristics of irrelevant pages (pages that did not contribute to solving the proposed task).

2.5 Study Steps

As previously mentioned, we initiated the study by sending an initial email invitation to students, providing them with information about the study's purpose. We requested interested participants to respond to this email to express their willingness to contribute.

Upon receiving positive responses to the initial email, we sent out a second email containing a link to an explanatory video. This video clarified the study's procedures and included instructional videos on how to install and utilize screen recording software, as well as guidance on sharing the recorded videos via a cloud storage service. Additionally, the email concluded with a link to an online questionnaire where participants answered basic questions. These questions covered topics such as their comfort level with reading web pages in English, years of experience with the Java programming language, and confirmation of whether they had viewed all the training materials provided in the email.

Subsequently, we dispatched a third email to participants who completed the questionnaire from the second email. This email contained instructions detailing the tasks participants were required to perform. We also provided explanatory videos on how to import the projects associated with the proposed programming tasks into the Integrated Development Environment (IDE), along with explanations of the task statements. For each programming task, participants followed this procedure: They (1) watched the video explaining the task statement, (2) initiated screen recording, (3) proceeded to solve the task using the available ranking of pages, and then (4) completed a questionnaire containing questions related to the pages they used to solve the proposed task. After completing these steps, participants submitted the recorded videos to the research team.

2.6 Methodology for Evaluating the Results

To analyze and assess the results obtained from the participants' videos, we examined the recorded footage of the task

resolutions to identify patterns in the programmers' actions. Once patterns are identified, we proceed to coding. The process is iterative, that is, new patterns may emerge as new videos are analyzed. When we identify a new pattern, we perform a retrospective analysis of previous videos to verify its possible occurrence. After analyzing all the videos, the resulting codes have been grouped into three broader categories. We then analyzed the aggregated data to extract meaningful findings. Below we show the details of the application of our method.

The initial step was analyzing the resolution of task 01 for participant 01, to encode their actions. This process was repeated for participants 02, 03, and 04. The following codes were created after the examination of task 01 resolutions by these four participants. Notably, after analyzing participant 03's data, no new codes were created suggesting that we have reached a convergent set of codes. Each of the following codes is accompanied by a description of how it was identified in the video:

1. **Analysis of the solution source code in the IDE.** The participant stopped at the source code in the IDE.
2. **Change the solution source code in the IDE.** The participant typed or removed characters in the source code in the IDE.
3. **Quick analysis of the textual content on the page.** The participant scrolled slowly through the textual content of the page, with a few short stops.
4. **Quick analysis of the source code on the page.** The participant scrolled slowly through the source code present on the page, with short stops.
5. **Analysis of textual content on the page.** The participant stopped at the textual content of the page (long stops).
6. **Analysis of the source code present on the page.** The participant stopped at the source code present on the page (long stops).
7. **Skipped textual content on the page.** The participant quickly scrolled through the textual content of the page (non-stop), or the participant used the browser's search bar with a technical element name as input (class, method, etc.).
8. **Skipped the source code present on the page.** The participant quickly scrolled through the source code present on the page (without stops).
9. **Copy of source code present on the page.** The participant copied a snippet of the source code present on the page and pasted it into the source code in the IDE.
10. **Copy of some part of the ranked content.** The participant copied page link content (description or list of most frequent methods on the page) of ranking and pasted it into the source code in the IDE.
11. **Analysis of ranked list of pages.** The participant rolled slowly through the list of ranked pages with long stops.

Following the definition of the codes mentioned above, we proceeded to analyze the videos containing the resolutions of the four tasks performed by the first four participants, resulting in a total of 16 analyses. During each analysis, we measured the time that each participant spent performing each of the actions coded above on every page within the ranking they utilized to solve the respective task.

We then organized the previously established codes into the following categories:

1. **Editing in the IDE.** This category groups the following codes associated with the IDE:
    - *(Code 1) - Analysis of the solution source code in the IDE*
    - *(Code 2) - Change the solution source code in the IDE*
2. **Page searching.** This category includes the following codes related to pages:
    - *(Code 3) - Quick analysis of the textual content on the page*
    - *(Code 4) - Quick analysis of the source code on the page*
    - *(Code 5) - Analysis of the textual content on the page*
    - *(Code 6) - Analysis of the source code present on the page*
    - *(Code 7) - Skipped textual content on the page*
    - *(Code 8) - Skipped the source code present on the page*
    - *(Code 9) - Copy of source code present on the page*
3. **Ranked list of pages.** This category encompasses the following codes associated with ranking:
    - *(Code 10) - Copy of some part of the ranked content*
    - *(Code 11) - Analysis of ranked list of pages*

We analyzed the time duration that participants spent in each of the categories. Additionally, we calculated the total time dedicated to solving each task, which involves the cumulative time spent on searching the ranked list of pages, the cumulative time spent on searching the pages, and the cumulative time spent editing in the IDE.

Furthermore, we analyzed the participants' utilization of each visited page and categorized them as follows:

1. **Relevant.** The page content was used to completely solve the desired solution.
2. **Partially relevant.** The page content partially contributed to achieving the desired solution.
3. **Irrelevant.** The page content did not contribute to the desired solution.

For each task, we calculated the time participants expended on pages falling into the categories of relevant, partially relevant, and irrelevant. This analysis provides insights into the effectiveness of the pages consulted during the process of solving the programming tasks. Starting with participant 05, we performed the coding at the category level, since the analyses performed in the current work are based on the category level. The coding process, along with the time spent on each of the participants' actions, can be found at the following link.[3]

[3] https://doi.org/10.5281/zenodo.14009184

3 Results

This section presents the results obtained through the analysis of the participants' videos. The study involved eight participants, where each of the participants received four programming tasks. This resulted in a total of 32 task samples. These tasks were evenly split, with 16 performed using Higher Quality Ranking and the remaining 16 using Lower Quality Ranking. Consequently, we established 16 pairs of tasks for paired comparison. Regarding the profile of the participants, all participants answered the basic questions in the questionnaire. They said they were comfortable or neutral regarding the content used in the experiment in English. Regarding years of experience with the Java programming language, the group of graduate students has 5 to 30 years of experience. The group of undergraduate students have 1 to 4 years of experience with the Java language. Below is a table with information about each participant.

Table 4. Participants' profiles

ID	Are you comfortable with the content of the pages being in English?	How many years of experience with the JAVA programming language do you have?	Graduate or Undergraduate student
1	Very comfortable	From 11 to 20 years	Graduate Student
2	Comfortable	From 5 to 10 years	Graduate Student
3	Very comfortable	From 21 to 30 years	Graduate Student
4	Neutral	From 3 to 4 years	Undergraduate student
5	Comfortable	From 1 to 2 years	Undergraduate student
6	Neutral	From 1 to 2 years	Undergraduate student
7	Neutral	From 1 to 2 years	Undergraduate student
8	Neutral	From 1 to 2 years	Undergraduate student

3.1 On the Influence of Ranking Quality

Table 5 presents the relationship between completely resolved, partially resolved, and unresolved tasks, considering Higher Quality Ranking (HQR) and Lower Quality Ranking (LQR).

Table 5. Tasks completion considering the Higher Quality Ranking (HQR) and Lower Quality Ranking (LQR)

	HQR	LQR
Completed	14	11
Partially Completed	2	3
Not Completed	0	2

As we can see in Table 5, the ranking quality seems to influence the completion of the tasks. Of the 16 tasks related to the Higher Quality Ranking, 14 were completely solved and 2 were partially solved by the participants. In the case of the Lower Quality Ranking, of the 16 tasks, 11 were completely solved, 3 were partially solved and 2 were not solved by the participants. When analyzing the participants' profiles, of the 3 partially completed tasks related to the Lower Quality Ranking, 2 of them are attributed to graduate students, suggesting the LQR may affect even more experienced programmers. The other partially completed and not completed tasks are assigned to undergraduate students.

Table 6 shows the questionnaire results for the following question: *"Regarding navigability by ranking, how agile/productive was it to find the solution for the programming task?"*. In this table, the symbols $++$, $+$, $-$ and $--$, correspond to the response options "very agile/productive", "agile/productive", "not agile/productive" and "very little agile/productive", respectively. The unfilled circles represent the responses of participants who belong to the group of graduate students. The filled circles represent the responses of participants who belong to the group of undergraduate students.

Table 6. Results of the navigability by ranking, for the four proposed programming tasks. ○-grad., ●-undergrad.

Navigability	$++$	$+$	$-$	$--$
Task 1 (higher quality ranking)	○ ● ●	○ ●	○	
Task 2 (lower quality ranking)		● ●	○ ○ ○	
Task 3 (higher quality ranking)	○ ○ ● ● ● ●			
Task 4 (lower quality ranking)	●	○ ○ ● ●	○ ● ●	

As shown in Table 6, participants found the tasks associated with Higher Quality Ranking (Tasks 1 and 3) to be more agile and productive compared to those associated with Lower Quality Ranking (Tasks 2 and 4). When we perform a pairwise comparison between Tasks 1 and 2, Table 5 reveals a greater number of positive evaluations for Task 1, which uses the Higher Quality Ranking, in contrast to Task 2 with the Lower Quality Ranking. A similar trend is observed in the comparison between Tasks 3 and 4, where Task 3, utilizing Higher Quality Ranking, received more favorable evaluations. Notably, all participants rated the navigability of Task 3 as very agile/productive.

In Figure 3, we present the time spent on different activities, where participants are using either the Lower Quality Ranking (LQR) or the Higher Quality Ranking (HQR). We observe that the time spent on all activities tends to be higher when using a Lower Quality Ranking. Nonetheless, the application of Wilcoxon Signed Rank one-sided has shown statistically significant difference only for the *TimeInRanking*, *TimeInIrrelevantPages*, and *TotalTimeTasks*. We calculated the effect size with Cliff's $\delta = 0.19$ for the *TotalTimeTasks*, which is a small effect size. This could be explained that after the participant has reached a useful page there is no statistically significant difference in using the IDE for coding the solution or reading that page. Moreover, since these latter activities are where the programmer spends most of the time, we could not observe a large difference in the overall time. Nonetheless, the apparently higher time spent in IDE for the LQR may be explained by the programmer trying to

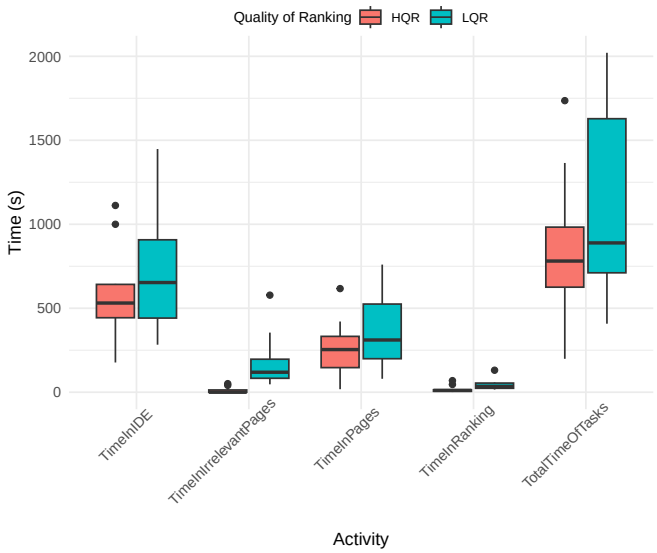

Figure 3. The time spent on different activities categorized by the groups HQR (Higher Quality Ranking) and LQR (Lower Quality Ranking). *TimeInIrrelevantPages*, *TimeInRanking* and *TotalTimeInTasks* were statistically significant with Wilcoxon Signed Rank one-sided, $p = 0.021, p = 0.00049, p = 0.0415$, respectively. For *TimeInIDE*, *TimeInPages*, $p = 0.1602$, $p = 0.06152$, respectively.

use a page that is not adequate and thus having to rework the task.

Furthermore, we provide an analysis of the video recordings with the results being summarized in Tables 7, 9, and 10. Table 7 specifically presents findings related to RQ1 (*Does the ranking quality of query results influence programmers' performance in programming tasks?*). The **Finding 1** indicates that among the eleven task resolution pairs analyzed, seven exhibited shorter resolution times for tasks associated with Higher Quality Ranking compared to those for Lower Quality Ranking tasks. We removed five out of 16 resolution pairs from the analysis because participants did not complete those tasks. On average, there was a time difference of 493 seconds (equivalent to 8 minutes and 13 seconds) between these seven task pairs. Conversely, when Lower Quality Ranking tasks had shorter resolution times than Higher Quality Ranking tasks, i.e. in the other four task pairs, the average difference was 76 seconds (equivalent to 1 minute and 16 seconds).

The **Finding 2** indicates that in tasks related to Higher Quality Ranking, participants primarily relied on the top-ranked pages. This finding was obtained through an analysis of twelve fully completed tasks associated with Higher Quality Ranking. In five tasks, participants used only the first page of the ranking to solve the task, while in another three tasks, they utilized both the first and second pages. In one task, a participant employed the first and third pages, and two participants used the first and fourth pages. Interestingly, one participant solely relied on the fourth page, and in another task, a participant extracted information solely from the descriptions in the ranking without accessing any actual pages.

The **Finding 3** reveals that, on average, participants were more inclined to visit pages associated with Lower Quality Ranking, in contrast to those related to Higher Quality Ranking. Specifically, participants visited an average of 4.18 pages linked to Lower Quality Ranking and only 1.69 pages connected to Higher Quality Ranking.

On the **Finding 4**, participants visited more irrelevant pages when engaging with Lower Quality Ranking compared to Higher Quality Ranking. Specifically, participants encountered a total of 29 irrelevant pages while interacting with Lower Quality Ranking, whereas only four irrelevant pages were encountered during interactions with Higher Quality Ranking.

The **Finding 5** demonstrates that in partially completed tasks involving Lower Quality Ranking, participants tended to visit pages located at lower positions within the ranking. For instance, Participant 02 visited the 14th page, and Participant 03 visited the 11th page in their respective interactions.

The **Finding 6** shows that regarding the total time spent on the pages, participants spent 90.4% of the time on useful and partially useful pages when performing tasks related to the Higher Quality Ranking. In the Lower Quality Ranking, participants spent 49.0% of the time on useful and partially useful pages when solving the tasks.

Answer to RQ1: *Does the ranking quality of query results have any influence on the performance of programmers in the development of programming tasks?*
This study shows that tasks associated with Lower Quality Ranking required noticeably more effort to resolve compared to those related to Higher Quality Ranking, suggesting that the complexity of the ranking influenced the programmers' performance during task resolution. This difference arises because programmers tend to spend less time on irrelevant pages within Higher Quality Ranking since the most relevant pages are conveniently situated at the top of the ranking. In the context of Higher Quality Ranking, other influencing factors include participants' tendency to visit fewer pages. Conversely, in the case of Lower Quality Ranking, participants exhibited different behaviors, such as visiting more irrelevant pages and exploring positions deeper within the ranking, even delving into the lowest-ranking positions. Another point is that the ranking quality seems to influence task completion because Higher Quality Ranking shows a higher rate of complete task resolution compared to Lower Quality Ranking, where some tasks remained unsolved or were only partially solved by participants.

3.2 On the Influence of the List of Methods

Table 8 shows the results of the questionnaire for the question: *"Did the summary below the link, where it shows the five methods that occurred most on the pages, have any relevance in your decision to click on the page?"*. In this table, the symbols $++$, $+$, $-$ and $--$, correspond to the response options "very relevant", "relevant", "irrelevant" and "very irrelevant", respectively. The unfilled circles represent the responses of participants who belong to the group of graduate students. The filled circles represent the responses of participants who belong to the group of undergraduate students.

As shown in Table 8, the addition of a list of methods in the summary below the links of the pages proved to be relevant

Table 7. Findings related to **RQ1**, found in the results of the analysis of the participants' videos.

#	Findings	Sources for the Findings
1	Participants apparently spend more time in activities related to Lower Quality Ranking compared to the Higher Quality Ranking, although only when analyzing the ranking and using irrelevant pages were found to be statistically significant.	On average, programmers in the assessment spent 8 minutes and 13 seconds more on Lower Quality Ranking tasks. This result was obtained through an analysis of eleven pairs (out of the total of sixteen tasks, only eleven were completed, leaving five tasks unfinished) of task resolutions, where seven samples showed that the resolution time for tasks using Higher Quality Ranking was shorter than that for tasks using Lower Quality Ranking.
2	In tasks related to Higher Quality Ranking, participants predominantly relied on the pages at the top of the rankings.	In five tasks, participants used only the first page of the ranking to solve the task. In three other tasks, participants utilized both the first and second pages of the ranking. One participant employed the first and third pages for one task, while two participants resorted to the first and fourth pages to complete their tasks. Additionally, in one task, a participant solely relied on the content descriptions within the ranking pages without accessing any of the pages themselves.
3	Participants tended to visit more pages associated with Lower Quality Ranking compared to those related to Higher Quality Ranking.	On average, participants visited 4.18 pages related to Lower Quality Ranking, while they visited an average of 1.69 pages related to Higher Quality Ranking.
4	Participants visited more irrelevant pages in their interactions with Lower Quality Ranking compared to Higher Quality Ranking.	Participants visited a total of 29 irrelevant pages in the context of Lower Quality Ranking, while they encountered only four irrelevant pages in the context of Higher Quality Ranking.
5	In the partially completed tasks involving Lower Quality Ranking, participants tended to visit pages located at the lowest positions within the ranking.	For instance, Participant 02 visited the 14 page and Participant 03 visited the 11 page in their respective interactions.
6	The average percentage of time spent on useful and partially useful pages in the Higher Quality Ranking is higher than the percentage spent on the Lower Quality Ranking.	Regarding the total time spent on the pages, participants spent 90.4% of the time on useful and partially useful pages when performing tasks related to the Higher Quality Ranking. In the Lower Quality Ranking, participants spent 49.0% of the time on useful and partially useful pages when solving the tasks.

Table 8. Results of adding a list of methods in the summary below the links of the pages for the four proposed programming tasks. ○-grad., ●-undergrad.

Added Methods List	++	+	−	−−
Task 1	●	○ ●	○ ○ ●	●
Task 2	● ● ●		○ ○ ●	
Task 3	○ ● ●	●	○ ●	●
Task 4	●	○ ● ● ●	○	●

for fifteen evaluations across the four tasks. We can also observe that the group of undergraduate students found the list of methods more relevant than the group of graduate students. In these evaluations, eight were rated as "very relevant" (++), while seven were considered "relevant" (+). In twelve evaluations, participants found the list of methods to be irrelevant. Out of these, three were marked as "very irrelevant" (–) and nine as simply "irrelevant" (-). To ensure the consistency of responses in the questionnaire, we analyzed the participants' actions in the video recordings. In one instance, we identified an inconsistency where a participant had used the methods listed (by copying and pasting them into the source code in the IDE) but marked the form response as "irrelevant." In this case, we adjusted the participant's response to "relevant," as their actions in the video recording indicated the relevance of the list of methods.

Table 9 shows the findings related to RQ2 (*Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for programmers?*). The **Finding 7** highlights an interesting case where one participant effectively utilized the methods listed to solve two programming tasks. This participant analyzed the method list on one of the pages, copied the method name, pasted it into the IDE's source code, and completed the task without the need to access any ranking page. The same participant replicated this process by copying a method name from the list for another proposed task.

The **Finding 8** revealed that, with the exception of one participant, all other participants consistently skipped irrelevant pages. Among the eight participants, seven demonstrated a pattern of bypassing irrelevant pages. This behavior is likely influenced by the presence of a list of frequent methods in the page descriptions within the ranking. The list of methods appears to play a pivotal role in participants' decision-making processes regarding whether to enter a particular page or not. When presented with a list of methods that are not relevant to the desired solution, participants are inclined to skip the page. Similarly, for pages lacking any methods in the list (resulting in an empty list), participants may infer that the page does not contain relevant source code, leading them to make the decision to skip the page. This behavior underscores the potential utility of method lists in aiding participants in determining the relevance of pages in their search for solutions.

Table 9. Findings related to **RQ2**, found in the results of the analysis of the participants' videos.

#	Findings	Sources for the Findings
7	One of the participants used the methods in the list of methods to solve two programming tasks.	This participant examined the list of methods on one of the pages, copied the method's name, pasted it directly into the source code within IDE, and performed the task without having to enter any ranking page. Remarkably, this participant replicated the same process by copying a method name from the list for another proposed task.
8	With one exception, all participants consistently avoided irrelevant pages.	In our analysis of the tasks where participants skipped irrelevant pages in the ranking, we observed a noteworthy pattern. Specifically, four participants hovered their mouse cursor over the names of the methods listed in the method list. This behavior suggests that these participants actively engaged with and analyzed the list of methods, which could potentially influence a programmer's decision on whether or not to navigate to a particular page. In contrast, the remaining participants appeared to be stuck at the description of the pages where the list of methods was located, although it is unclear whether they were actively analyzing the list of methods or not.

Answer to RQ2: *Is there any influence by adding a list of frequent methods in the description of the pages in the result returned by the search engine for programmers?* As indicated by the findings in Table 9, it appears that there is indeed an influence. This influence is discernible through the behavior of the participants. Specifically, seven participants chose to skip irrelevant pages within the ranking, while only one participant did not skip irrelevant pages. Furthermore, one of the participants demonstrated the significance of the method lists in two proposed tasks. Remarkably, in one of these tasks, this participant solely relied on the content of the list of methods without needing to access any ranking pages. This suggests that the presence of these lists may have an impact on programmers' decision-making processes.

3.3 On the Influence of Irrelevant Pages on Programmers

The Table 10 shows the findings related to RQ3 (*How do irrelevant pages present in lower quality rankings hinder programmers' goal of finding relevant solutions?*). The **Finding 9** shows that the time spent analyzing irrelevant pages within the Lower Quality Ranking was notably longer than the time spent analyzing irrelevant pages within the Higher Quality Ranking. Among the eleven pairs of tasks solved, seven samples indicated a greater time effort in analyzing the Lower Quality Ranking irrelevant pages compared to the Higher Quality Ranking pages. This resulted in an average time difference between task pairs of 205 seconds (equivalent to 3 minutes and 25 seconds). Conversely, in the remaining four samples where the time spent analyzing Higher Quality Ranking pages exceeded that of the Lower Quality Ranking pages, the average time difference between tasks was 70 seconds (equivalent to 1 minute and 10 seconds). The Cliff's $\delta = 0.96694$ indicates a large effect size.

The **Finding 10** demonstrates that on average, the time spent in the IDE was higher on tasks related to Lower Quality Ranking, however not statistically significant at $p < 0.05$. Among the eleven pairs of tasks fully completed by the participants, in six pairs, there was a notable average difference in time spent in the IDE between tasks related to Higher Quality Ranking and Lower Quality Ranking. This difference averaged 295 seconds, equivalent to 4 minutes and 55 seconds, suggesting that participants spent significantly more time in the IDE on tasks associated with Lower Quality Ranking. In contrast, in the remaining five pairs where the time spent in the IDE was higher for Higher Quality Ranking tasks, the mean difference in time spent in the IDE was 94 seconds (equivalent to 1 minute and 34 seconds).

On the **Finding 11**, for all pairs of tasks analyzed, the time spent on irrelevant pages was consistently higher for Lower Quality Ranking with $p = 0.00049$. Across the eleven task resolution pairs, there was consistently more time allocated to irrelevant pages within the Lower Quality Ranking in comparison to the Higher Quality Ranking. This resulted in an average time difference of 161 seconds (equivalent to 2 minutes and 41 seconds) between the task pairs. The Cliff's δ indicated a large effect size. Nonetheless, we should put in perspective that the average time spent in irrelevant pages is not as large as the spent time in IDE and in relevant pages, explaining the small effect size on the total time for executing the whole task.

Answer to RQ3: *How do irrelevant pages present in lower quality rankings hinder programmers' goal of finding relevant solutions?*
As shown in the results of Table 10, programmers spent more time in the IDE when working on tasks related to Lower Quality Ranking. Additionally, they spent more time on irrelevant pages for these tasks, highlighting a noticeable difference in effort compared to tasks associated with Higher Quality Ranking. This occurs because when programmers encounter suboptimal code on irrelevant pages, they expend considerable time attempting to salvage and reuse this code. In their efforts to fix the problematic code, programmers often navigate to other pages in search of related content, further extending their time investment. If, ultimately, the programmer cannot fix the issue with the flawed code, all the effort invested in locating the solution becomes futile, compelling the programmer to recommence the search for a solution on another page. This highlights how irrelevant pages in Lower Quality Rankings can significantly impede programmers in their quest to find pertinent solutions, resulting in both time and effort being wasted in the process.

Table 10. Findings related to **RQ3**, found in the results of the analysis of the participants' videos.

#	Findings	Sources for the Findings
9	The time spent analyzing irrelevant pages within the Lower Quality Ranking was notably longer than the time spent analyzing pages within the Higher Quality Ranking	Among the eleven pairs of task resolutions, it was observed that in seven samples, participants spent more time analyzing Lower Quality Ranking pages compared to Higher Quality Ranking pages. This resulted in an average time difference between the task pairs of 205 seconds (equivalent to 3 minutes and 25 seconds). Conversely, in the remaining four samples where the time spent analyzing Higher Quality Ranking pages exceeded that of Lower Quality Ranking pages, the average time difference between tasks was 70 seconds (equivalent to 1 minute and 10 seconds).
10	The time spent in the IDE was on average higher on tasks related to Lower Quality Ranking, but not statistically significant.	Out of the eleven pairs of tasks completely solved by the participants, six pairs exhibited a mean difference in time spent in the IDE between Higher Quality Ranking and Lower Quality Ranking tasks of 295 seconds. In other words, participants spent an average of 4 minutes and 55 seconds more in the IDE when working on tasks related to Lower Quality Ranking. Conversely, in the remaining five pairs where the time spent in the IDE was higher for Higher Quality Ranking tasks, the mean difference in time spent in the IDE was 94 seconds (equivalent to 1 minute and 34 seconds). Nonetheless, this difference has not been shown to be statistically significant, suggesting that after the programmers are within the IDE, the ranking plays no major influence.
11	The time spent on irrelevant pages was consistently higher for Lower Quality Ranking.	For all pairs of fully resolved tasks, the time spent on irrelevant pages was higher for the Lower Quality Ranking. This resulted in an average time difference between task pairs of 161 seconds, which is equivalent to 2 minutes and 41 seconds.

4 Discussion

In this section, we will discuss the findings reported previously, moving on to an in-depth analysis of our results in the context of Information Foraging Theory (IFT) and Generative AI, exploring the implications and relevance of our findings within these topics. This work was conducted before generative AI became so pervasive in programmer workflows. Based on this, we provide a transparent discussion on the impact of generative AI on the search world and how our findings remain relevant, while also considering new challenges and perspectives introduced by these advancements. Additionally, we included IFT to address the assumption that filtering irrelevant search results is purely negative, as IFT suggests that this process can contribute to learning and adapting strategies during information seeking.

4.1 Discussion of the Results

This section aims to discuss the findings presented in Tables 7, 9 and 10. We will begin by addressing the findings related to RQ1.

Upon analyzing the findings related to RQ1, it becomes evident that the quality of ranking has an important impact on programmers' performance. On average, the time required to complete tasks associated with the Higher Quality Ranking is shorter than that required for tasks linked to the Lower Quality Ranking. This difference arises from the fact that in Higher Quality Rankings, relevant pages occupy prominent positions at the top, allowing programmers to bypass irrelevant pages efficiently. Conversely, in the Lower Quality Ranking, programmers are compelled to navigate through numerous irrelevant pages that hinder their performance until they eventually locate a relevant page that addresses the desired problem. This contrast underscores the critical role that ranking quality plays in streamlining programmers' workflows and enhancing their efficiency.

Indeed, we expected that participants would spend less time on tasks with a higher quality ranking, as we placed the pages with the best solutions at the top. However, it is still relevant to analyze the behavior of programmers in these situations, because when solving tasks, they were not aware of whether they were using the Higher Quality Ranking or Lower Quality Ranking. Therefore, our intention was not only to measure time spent on pages, but also to observe whether programmers tend to click on the first few pages without analyzing the pages' titles, descriptions, and most frequent method lists. The results indicated that programmers take this information into consideration. In tasks with higher quality ranking, in most cases, programmers click on the first few pages, while in tasks with lower quality ranking, they skip several irrelevant pages during rank analysis.

Regarding the RQ2, we could observe a behavior where a participant used the information from the list of most frequent methods present in the description of the ranked pages to try to solve a task. This behavior may be explained in cases where programmers already expect some functions and this list of methods works as a hint for them. Another observed point was that most of the participants skipped the irrelevant pages. This behavior may be due to the list of frequent methods in the page descriptions in the rank. This list probably helps in the decision to enter the page or not, when faced with a list of methods that are not related to the desired solution, the participant decides to skip the page. For pages that do not have methods, the list of methods is empty, so the participant may infer that there is no source code on the page, consequently, the participant would skip the page.

Findings related to Research Question 3 (RQ3) reveal that irrelevant pages hinder the performance of programmers. For instance, when programmers encounter sub-optimal code on these pages and are uncertain about its quality, they may attempt to incorporate it into their project. Once they realize that the code does not enhance the solution, they have two options: either they remove the problematic code, or they seek information on other pages to fix it. The most undesirable scenario arises when programmers are unable to fix the code, resulting in a significant waste of time dedicated to these ir-

relevant pages.

Participants' comments shed light on specific characteristics of irrelevant pages and provide valuable insights. Following some of these comments are discussed:

- *Comment 1:"The irrelevant pages had only formal definitions, with little applicability. One page did not even show the solution in Java."* - the comment demonstrates a lack of practical examples, i.e., irrelevant pages often contain only formal definitions with limited applicability. In some instances, pages may not even provide solutions in the desired programming language. This lack of practical examples forces programmers to expend valuable time searching for relevant content that the page fails to offer.
- *Comment 2:"The irrelevant pages did not have examples that matched or related to what the task asked for."* - this comment highlights that irrelevant pages frequently do not align with the specific requirements of the programming task at hand. These pages fail to provide solutions or content relevant to the task's context, leading programmers astray.
- *Comment 3:"The irrelevant pages had extensive content and sometimes did not have the necessary content to perform the task."* - irrelevant pages can also be characterized by their extensive content that lacks a clear focus on the proposed task. Programmers are compelled to sift through this extensive material, even when it does not contribute to task resolution, resulting in time wastage.
- *Comment 4:"They were not specifically for the task I was looking to solve. I wanted snippet to save in the database, not to connect or consult in the database."* - another comment emphasizes that irrelevant pages do not address the specific solution sought by the programmer. This misalignment with the programmer's objective forces them to analyze content that does not aid in solving the problem.
- *Comment 5:"They did not have the insertion, only codes for other operations, even when dealing with the same API."* - irrelevant pages may fail to cater to the programmer's intended operations. For instance, a comment notes that these pages lacked information related to a specific operation, even when dealing with the same API.
- *Comment 6:"The irrelevant pages had examples that did not work."* - this comment highlights the presence of non-functional examples on irrelevant pages. Programmers are enticed to spend valuable time attempting to make these non-functional examples work, diverting their efforts away from finding a practical solution.

We also analyzed the positive aspects of the relevant pages that helped participants solve the proposed tasks and found the following: 81.25% of the participants' responses mentioned as a positive point the presence of practical examples, which facilitated the resolution of the tasks. Among these responses, some of the participants highlighted that the code was direct, concise, and precise. These results reinforce the importance of search engines developing filters that return pages with practical examples and solutions focused on the real needs of users.

It is worth noting that, although technologies related to Large Language Models (LLMs) are gaining ground in the area of searching for content related to software development, search engines are still relevant and widely used by programmers. In this way, our results can help the development of search engines with a better understanding of how programmers interact with the pages returned by the engine, especially when the first pages returned are irrelevant to the solution sought by the programmer. This way, filters to remove irrelevant pages from the top of the rankings would be proposed. Such filters would help programmers bypass irrelevant pages and access content that directly addresses their specific needs, ultimately improving their efficiency and productivity in programming tasks. Another audience for our results would be browser extension programmers, or IDE plugin programmers who want to develop approaches to obtain only the relevant pages in the rank returned by the search engine. The results of this article may also be useful for developing solutions related to the presentation of development information in search engines. As we observed in our results, the inclusion of a list of methods below the description of pages in the ranking influenced programmers' decisions as to whether or not to click on pages. In this way, additional information related to software development would help programmers choose relevant pages related to their search, by evaluating the additional information and deciding whether they want to click on the page or not.

4.2 Generative AI as an Alternative to Traditional Search Engines

With the emergence of Large Language Models (LLMs), many programmers have been seeking to solve their programming tasks using solutions provided by tools like ChatGPT. A recent example is the DevGPT dataset (Xiao et al., 2024), constructed from thousands of GitHub examples containing pull requests, issues, and even source code comments where ChatGPT provided a solution that was employed by programmers. LLMs have also emerged as an alternative for providing new kinds of solutions to programmers, as they often ask ChatGPT for tasks such as performing refactoring operations into their source code, fixing bugs, and others (Tufano et al., 2024). While search engines aim to find ready-made examples (how-to-do) that closely match programmers' needs, LLMs present an alternative by generating possibly original answers, including modified versions from programmers' source code (Ebert and Louridas, 2023).

However, single-query responses from LLMs might not fully address the user's needs. LLMs maintain a context window that leverages a series of interactions, or prompts, to generate the desired solution. This iterative process between the programmer and the LLM involves refining and expanding the dialogue until the model generates an accurate and helpful response (Ebert and Louridas, 2023) (Mondal et al., 2024). This process requires *prompt engineering* because the quality of the prompts influences the programmers' performance during programming tasks.

Our findings reveal that when search engines return pages in a Low Quality Ranking, programmers tend to visit more pages and spend additional time on content that may not be

relevant to their needs. Those irrelevant pages might be a consequence of a difficulty for programmers to accurately express their query intention (Zha et al., 2010). Such difficulties can lead programmers to reformulate their queries in an attempt to find the desired information (Cao et al., 2021). Similar to search engines, the LLMs might also involve a process of query or prompt reformulation, which could also impact the programmers' performance in solving tasks. However, while search engines require the programmer to click on links and navigate through pages, the LLMs require them to read the response and reformulate the prompt as necessary.

Another dimension to consider is that LLMs provide answers based on vast amounts of data on which it has been trained, such as those archived and indexed by search engines. This indicates that the model's ability to assimilate new programming language features is contingent upon the availability of data embodying these features. In contrast, search engines can more swiftly address solutions concerning new features since users continually feed the web with such information. This suggests that while LLMs benefit from extensive datasets for learning, search engines have the advantage of quickly tapping into the latest user-generated content for up-to-date solutions (Dantas et al., 2023).

Furthermore, a notable challenge for LLMs in contrast to search engines is the phenomenon of *hallucinations*, where the model generates content that is either nonexistent or not discernible to human observers. This aspect underscores a critical limitation in the reliability of Generative AI, highlighting the importance of continuous improvement in model training methodologies to mitigate the occurrence of such inaccuracies (Zuccon et al., 2023).

However, the limitations currently observed in LLMs may be mitigated as the Generative AI continues to evolve. As more data becomes available and the algorithms underpinning these models are refined, there is potential for significant improvements in accuracy and reliability.

In the end, we suggest that a similar study as the one presented in this paper would play an important contribution to understanding the impact of low-quality responses of LLMs in the programmers' performance.

4.3 The Information Foraging Theory

The Information Foraging Theory (IFT) elucidates and predicts how people orient themselves based on the information present in their environment. IFT suggests that individuals (in this case, programmers) forage for information by following scents that they believe will lead them to valuable information. This has proven especially helpful in the context of programming, where programmers frequently search for code snippets, debugging information, and software development methodologies. This could help create more intuitive search mechanisms and reduce the time and effort required for programmers to find relevant solutions.(Fleming et al., 2013).

The IFT connects with the findings of this paper. For instance, in Finding 5, participants, upon noticing that the initial pages did not deliver what they expected, then visited pages located in lower positions in the ranking. This illustrates the navigation and reading through pages that did not meet their expectations, as demonstrated in Finding 6. Findings 7 and 8 further revealed that incorporating a list of methods positively influenced programmers' ability to forage for information and differentiate between contextually similar variants.

The concept of ranking quality and the list of methods outlined in this research directly correspond to the quality of these information scents. The Higher Quality Rankings and the list of methods empower programmers to independently evaluate a page's relevance and quality to help with their own programming tasks. Exploring alternatives that enable programmers to make more informed assessments of page quality can lead to the provision of stronger cues, guiding them more effectively to the solutions that are most pertinent to their programming tasks (Kuttal et al., 2021).

This concept could also be applied to how the responses of LLMs may provide such scents, such that, the programmer would find if the responses are adequate or not for the proposed prompts.

5 Threats to Validity

In this section, we show some threats to the validity of this study.

Internal validity. This kind of threat is about how sure we can be that the treatment actually caused the outcome. In this study, a threat to internal validity was the degree of difficulty and the features of the programming tasks to be implemented considered in the study. Tasks with different degrees of difficulty or with different features to be implemented between the lower quality ranking and the higher quality ranking can influence the results. To minimize this threat, we performed a paired analysis between pairs of tasks, with a similar degree of difficulty and we sought pairs of tasks from the same application domain, for example, pairs of tasks 1 and 2, are related to the graphical user interface. Pairs of tasks 3 and 4 are related to database manipulation. We adjusted these pairs of tasks to make them as similar as possible, but still making them different enough to avoid the learning bias. We also opted for a within-subjects design to mitigate the influence of individual differences. If a between-subjects design had been used, it would have required a larger sample size to account for these variations. Additionally, the participants had varying levels of experience, which would have introduced further complexity in a between-subjects approach. By choosing a within-subjects design, we aimed to control for all other variables and focus on measuring the impact of changes in the ranking quality. Another threat to validity is the lack of control in the online setting, to mitigate this threat, we did a thorough analysis of the videos and found that participants completed the tasks in a continuous manner. The screens did not remain idle for extended periods, which would have indicated pauses in task resolution. This observation suggests that the online setting did not result in significant interruptions or disengagement during task completion.

External validity. This kind of threat is related to whether we can generalize the results outside the scope of our study. In this study, a threat to external validity was the number of study participants. However, this restriction can be justified when the methodology adopted emphasizes a qualitative ap-

proach, where the depth of analysis is prioritized over the breadth of the sample. Research that requires a high degree of detail in the assessments often requires considerable time for data collection and analysis, especially when applied to a large number of individuals. In the case of this study, a significant amount of time was required to analyze each action of the participants in the recorded video, coding each of them. Thus, the use of reduced samples ensures the quality and depth of the analysis. Another threat to external validity was the results were limited to the Java programming language, which is a popular language and has a lot of content on the Web. For other programming languages, the results may be different, especially for less popular languages, where there may be a lack of content, even considering the whole Web.

Construct validity. This kind of threat is about the relation between the theory behind the study and the observation. In this study, a threat to construct validity was the interpretation of the videos of the resolutions of the tasks, carried out by the participants, since an action performed by the participant may not be the same as interpreted by the video analysis. For example, regarding the list of methods inserted in the description of the pages, when the participant stops at the description of the page, it is not possible to know exactly if he/she is reading or not the text of the description or the list of methods inserted, which could impact the results. To mitigate this threat, we focused on collecting more direct relationships, for example, in the case of the list of methods, we observed actions performed by the participants, such as copying methods from the list of methods and pasting them into the source code present in the IDE and hovering the mouse cursor above the names of the methods present in the list of methods, followed by clicking on the link on the page. Another threat to the validity of the construction was the questionnaires designed to obtain information for the research. They may not capture the intention of the questions, as participants may answer the questions wrongly, because of subjective interpretations. To mitigate this threat, we have aimed to frame the questions as clearly as possible. We also analyzed the participants' videos to verify that the answers to the questionnaires are consistent. For example, in the questionnaire where the participant is asked to rate the degree of relevance of the list of methods that we have inserted in the description of the page, we analyze the participant's video to verify if they used the methods present in the list or not, then we check the option that the participant marked in the questionnaire.

Conclusion validity. This kind of threat is about how sure we can be that the treatment is really (statistically) related to the actual outcome. The statistical analyses presented are seen as complementary, but not central to the study's conclusions. Our aim was not to support the conclusions based solely on statistical data, but rather to highlight observed trends, which we understand can enrich the qualitative perspective. We focus on a qualitative study, using statistical insights to complement and deepen the understanding gained through qualitative analysis.

6 Related Work

Xia et al. (2017) try to understand what programmers have been searching online to increase their productivity, evaluating the quality of pages returned by search engines. The authors argue that several reviewers have raised concerns about the low quality of search engine results, which often forces programmers to spend more time finding the desired content. In our study, we also sought to explore how programmers interact with search engine results, but our focus was on understanding their use of these pages across two ranking quality levels: higher and lower quality rankings.

Rahman et al. (2018) investigated if general-purpose search engines like Google are an optimal choice for source code-related searches. They aimed at understanding whether the performance of searching with Google varies for code vs. non-code related searches. To carry out these investigations, the authors collected search logs from 310 programmers that contain nearly 150,000 search queries from Google and the associated result clicks. To differentiate between code-related searches and non-code related searches, they built a model which identifies the code intent of queries. Leveraging this model, they built an automatic classifier for code and non-code related query. The authors noted that code related searching often requires more effort (e.g., time, result clicks, and query modifications) than general non-code search, which indicates code search performance with a general search engine is less effective. The results of this work show the importance of general-purpose search engines for source code search, which motivated us to carry out a study on the influence of the ranked list of result pages on the performance of programmers during programming tasks.

Sadowski et al. (2015) conducted a case study related to how programmers search for code using the Google search engine. The results of the study show that programmers search for code very frequently, conducting an average of five search sessions with 12 total queries each workday. The authors also show that programmers are generally seeking answers to questions about how to use an API, what code does, why something is failing, or where code is located. Based on the free text answers, the most common questions were about finding code samples, meaning the programmers who participated in this study look for examples more than anything else. The results of this study also motivated us to carry out a study on the influence of the ranking quality on the performance of programmers, since programmers perform several queries on search engines during the day in search of code examples for solving day-to-day problems and these search engines can return irrelevant pages to programmers, hindering their performance during the resolution of programming.

Keane et al. (2008) evaluated whether people are biased in their use of a search engine, specifically, whether they are biased in clicking on those items that are presented at the top of the result list, returned by the search engine. To test this bias hypothesis, they simulated the Google environment systematically reversing Google's normal relevance-ordering of the items presented to users. The results showed that people do manifest some bias, favoring items at the top of result lists, although they also sometimes seek out high-relevance items listed further down the list. These results motivated us to bet-

ter understand how programmers behave when faced with a lower quality ranking returned in top-ranked positions of a search engine result.

Several other works have studied the impacts that search engines exert during activities related to software development. In the work of Fischer et al. (2021), the authors showed the effects that Google search engines have on software security. They showed that the probability of an insecure result appearing in the top three is 22.78%, while only 9.19% secure ones. In the work of Hora (2021b), the authors conducted an empirical study to understand what programmers search on the web and what they find. They found that queries performed by programmers typically start with keywords, e.g. Python, Android, etc., they have three words on the median, tend to omit function words, and are similar among each other. Another observation was that minor changes to queries do not broadly affect Google search results; however, some cosmetic changes can have a non-negligible impact. This study motivated us to investigate the impact that the ranking quality of the pages returned by the search engine has on the programmer. Since the results show how programmers make use of the Google search engine to search for content related to software development on the Web.

Cho and Roy (2004) studied the impact that search engines have on the evolution of the popularity of web pages. They analytically estimated how long it takes for a new page to attract a large number of web users, while searches return only popular pages at the top of search results. The results of this work show that search engines can have an immensely worrying impact on the discovery of new web pages. These results show that there may be new pages at the bottom of the rankings that have content that is relevant to the user. In the context of our work, there may be relevant solutions in lower positions on new pages.

In summary, while prior research has explored how programmers use search engines for programming tasks and the challenges associated with retrieving relevant content, our study advances this area by specifically focusing on how the quality of search engine result rankings affects programmers' performance. Unlike previous studies that concentrated on general-purpose search engines' effectiveness or the nature of programmers' search behavior, our work provides a deeper investigation into the impact of ranking quality on the programmers' performance while performing the task. By examining both higher and lower quality rankings, we aim to offer new insights into how programmers navigate search engine results and how these rankings influence their performance in solving tasks. This study contributes to the state-of-the-art by highlighting the role that search engine ranking quality plays in optimizing—or hindering—programmer performance

7 Conclusion

The results of this study underscore the significant influence of ranking quality on the performance of programmers during the development of programming tasks. On average, programmers spent less time solving tasks associated with a Higher Quality Ranking compared to those associated with a Lower Quality Ranking. This difference in time expenditure can be attributed to the fact that Higher Quality Ranking positions relevant pages at the top, thereby minimizing the time wasted on irrelevant pages.

Furthermore, the addition of a list of frequent methods below the descriptions of ranking pages influenced the participants' decision-making processes. Notably, one participant successfully completed a task using only the content provided in the descriptions, including the methods listed. Additionally, other participants were observed to skip irrelevant pages, potentially influenced by the presence of this information.

Another important observation is the negative influence of irrelevant pages on the development of programming tasks, particularly within the Lower Quality Ranking. In this case, the irrelevant pages often occupy top positions in the ranking, hindering the participants in their search for solutions. When programmers encounter suboptimal code on these irrelevant pages and are unsure of its quality, they tend to make efforts to utilize it. Upon realizing that the code does not contribute to the solution, programmers face a choice: either delete the problematic code or attempt to repair it by seeking information on other pages. In the worst-case scenario, programmers are unable to fix the code, resulting in the loss of valuable time invested in irrelevant pages.

Overall, we suggest the development of filters aimed at improving the quality of results delivered by search engines. As evident from this study, pages in Lower Quality Rankings can significantly hinder the performance of programmers, underscoring the need for strategies to improve the relevance of search results. Moreover, the results may encourage the adaptation of this study for other approaches that require information foraging, such as chatting with LLMs.

In terms of future work, we propose the development of a quantitative study to reinforce the results presented in this paper. Additionally, we recommend conducting research aimed at developing and evaluating more efficient filters for removing irrelevant pages within search engine results. We also suggest that a similar study as the one presented in this paper would play an important contribution to understanding the impact of low-quality responses of LLMs in the programmers' performance. Finally, the use of eye-tracking technology could indeed provide valuable insights into the elements that programmers tend to focus on when searching for solutions in search engines. This approach could help us better understand the influence of the list of methods proposed in the study. This technique could be applied in future studies.

Acknowledgements

We acknowledge CAPES, CNPq, and FAPEMIG for partial funding. We also acknowledge the students and professionals who participated in this study.

References

Buse, R. P. L. and Weimer, W. (2012). Synthesizing API usage examples. In *Proceedings of the 34th International*

Conference on Software Engineering, ICSE '12, page 782–792. IEEE Press.

Cao, K., Chen, C., Baltes, S., Treude, C., and Chen, X. (2021). Automated query reformulation for efficient search based on query logs from stack overflow. *CoRR*, abs/2102.00826.

Chatterjee, S., Juvekar, S., and Sen, K. (2009). Sniff: A search engine for java using free-form queries. In Chechik, M. and Wirsing, M., editors, *Fundamental Approaches to Software Engineering*, pages 385–400, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cho, J. and Roy, S. (2004). Impact of search engines on page popularity. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, page 20–29, New York, NY, USA. Association for Computing Machinery.

Dantas, C., Rocha, A., and Maia, M. (2023). Assessing the readability of ChatGPT code snippet recommendations: A comparative study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, SBES '23, page 283–292, New York, NY, USA. Association for Computing Machinery.

Ebert, C. and Louridas, P. (2023). Generative AI for software practitioners. *IEEE Software*, 40:30–38.

Fischer, F., Stachelscheid, Y., and Grossklags, J. (2021). The effect of Google search on software security: Unobtrusive security interventions via content re-ranking. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 3070–3084, New York, NY, USA. Association for Computing Machinery.

Fleming, S. D., Scaffidi, C., Piorkowski, D., Burnett, M., Bellamy, R., Lawrance, J., and Kwan, I. (2013). An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks. *ACM Trans. Softw. Eng. Methodol.*, 22(2).

Gallardo-Valencia, R. E. and Elliott Sim, S. (2009). Internet-scale code search. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, SUITE '09, page 49–52, USA. IEEE Computer Society.

Hora, A. (2021a). Characterizing top ranked code examples in Google). *Journal of Systems and Software*.

Hora, A. (2021b). Googling for software development: What developers search for and what they find. In *International Conference on Mining Software Repositories*, pages 1–12.

Johnson, R. E. (1992). Documenting frameworks using patterns. In *Conference Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA '92, page 63–76, New York, NY, USA. Association for Computing Machinery.

Keane, M. T., O'Brien, M., and Smyth, B. (2008). Are people biased in their use of search engines? *Commun. ACM*, 51(2):49–52.

Kim, J., Lee, S., Hwang, S.-w., and Kim, S. (2010). Towards an intelligent code search engine. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 1358–1363. AAAI Press.

Kuttal, S. K., Kim, S. Y., Martos, C., and Bejarano, A. (2021). How end-user programmers forage in online repositories? an information foraging perspective. *Journal of Computer Languages*, 62:101010.

Mondal, S., Bappon, S. D., and Roy, C. (2024). Enhancing user interaction in ChatGPT: Characterizing and consolidating multiple prompts for issue resolution. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

Nasehi, S. M., Sillito, J., Maurer, F., and Burns, C. (2012). What Makes a Good Code Example? A Study of Programming Q&A in StackOverflow. In *Proc. of the IEEE Intl. Conf. on Software Maintenance (ICSM'12)*, pages 25–34, Washington, DC, USA.

Niu, H., Keivanloo, I., and Zou, Y. (2017). Learning to rank code examples for code search engines. *Empirical Softw. Engg.*, 22(1):259–291.

Nykaza, J., Messinger, R., Boehme, F., Norman, C. L., Mace, M., and Gordon, M. (2002). What programmers really want: Results of a needs assessment for SDK documentation. In *Proceedings of the 20th Annual International Conference on Computer Documentation*, SIGDOC '02, page 133–141, New York, NY, USA. Association for Computing Machinery.

Rahman, M. M., Barson, J., Paul, S., Kayani, J., Lois, F. A., Quezada, S. F., Parnin, C., Stolee, K. T., and Ray, B. (2018). Evaluating how developers use general-purpose web-search for code retrieval. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, page 465–475, New York, NY, USA. Association for Computing Machinery.

Robillard, M. P. (2009). What makes APIs hard to learn? answers from developers. *IEEE Softw.*, 26(6):27–34.

Rocha, A. M. and Maia, M. A. (2023). Mining relevant solutions for programming tasks from search engine results. *IET Software*, 17(4):455–471.

Sadowski, C., Stolee, K. T., and Elbaum, S. (2015). How developers search for code: A case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 191–201, New York, NY, USA. Association for Computing Machinery.

Sim, S. E., Umarji, M., Ratanotayanon, S., and Lopes, C. V. (2011). How well do search engines support code retrieval on the web? *ACM Trans. Softw. Eng. Methodol.*, 21(1).

Stolee, K. T., Elbaum, S., and Dobos, D. (2014). Solving the search for source code. *ACM Trans. Softw. Eng. Methodol.*, 23(3).

Tufano, R., Mastropaolo, A., Pepe, F., Dabić, O., Penta, M. D., and Bavota, G. (2024). Unveiling ChatGPT's usage in open source projects: A mining-based study.

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., and Xing, Z. (2017). What do developers search for on the web? *Empirical Softw. Engg.*, 22(6):3149–3185.

Xiao, T., Treude, C., Hata, H., and Matsumoto, K. (2024). DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

Zha, Z.-J., Yang, L., Mei, T., Wang, M., Wang, Z., Chua, T.-S., and Hua, X.-S. (2010). Visual query suggestion: Towards capturing user intent in internet image search. *ACM Trans. Multimedia Comput. Commun. Appl.*, 6(3).

Zuccon, G., Koopman, B., and Shaik, R. (2023). Chatgpt hallucinates when attributing answers. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, SIGIR-AP '23, page 46–51, New York, NY, USA. Association for Computing Machinery.