Empirical Software Engineering (2025) 30:178
https://doi.org/10.1007/s10664-025-10727-w

The Influence of Code Comments on the Perceived Helpfulness of Stack Overflow Posts

Kathrin Figl[1] · Maria Kirchner[1] · Sebastian Baltes[2] · Michael Felderer[3,4]

Received: 18 August 2024 / Accepted: 26 August 2025 / Published online: 21 October 2025

Abstract
Question-and-answer platforms such as Stack Overflow are an important way for software developers to share and retrieve knowledge. However, reusing poorly understood code can lead to serious problems, such as bugs or security vulnerabilities. To better understand how code comments affect the perceived helpfulness of Stack Overflow answers, we conducted an online experiment simulating a Stack Overflow environment (n=91). The results indicate that both block and inline comments are perceived as significantly more helpful than uncommented source code. Moreover, novices rated code snippets with block comments as more helpful than those with inline comments. Interestingly, other surface features, such as the position of an answer and its answer score, were considered less important. Moreover, the content of Stack Overflow has been a major source for training large language models. AI-based coding assistants such as GitHub Copilot, which are based on these models, are changing the way Stack Overflow is used. However, our findings have implications beyond Stack Overflow. First, they may help to improve the relevance also of other community-driven platforms, which provide human advice and explanations of code solutions, complementing AI-based support for software developers. Second, since chat-based AI tools can be prompted to generate code in different ways, knowing which properties influence perceived helpfulness can lead to more targeted prompting strategies to generate readable code snippets.

Keywords Code comprehension · Code comments · Stack overflow · Question-and-answer platforms

1 Introduction

Programming is a constantly evolving process of problem solving that encompasses tasks such as feature implementation, debugging, and system maintenance. In this dynamic field, software developers often turn to a variety of information sources to achieve their goals. The

Communicated by: Yasutaka Kamei.

Extended author information available on the last page of the article advent of artificial intelligence (AI) has caused a paradigm shift in programming practices. AI-powered tools have the ability to generate code snippets from user input, streamlining the prototyping process for developers, and scan code for inefficiencies and bugs, while predictive coding and auto-completion features suggest code segments. However, the continued relevance of community-driven platforms such as Stack Overflow, even in the face of technological advances, is underscored by a modest 2.6% (Xue et al. 2023) decline in question activity on the site since the introduction of ChatGPT. It highlights the importance of platforms that provide human advice on programming issues. Stack Overflow, with its large pool of experienced developers, offers an environment in which detailed, community-validated, and context-specific advice thrives. The platform's peer-review system, manifested through user upvotes and downvotes, is built to ensure that the most accurate and helpful answers are elevated, maintaining a standard of quality and reliability. The diversity of answers to a single question, including their comments—each providing a unique perspective or solution—often yields more comprehensive insights than the solutions typically generated by AI. However, it is also worth noting that there is an active discussion in the community on if and why Stack Overflow is "fading away"[1] and whether LLMs make Stack Overflow irrelevant.[2]

Despite these discussions, code examples are a critical resource that eases the challenges of independent problem solving. Stack Overflow, with its extensive repository of more than 24 million questions, 36 million answers, and 91 million comments (Stack Exchange Inc 2025), remains the main question-and-answer (Q&A) forum for software developers around the world. As a platform for learning and sharing, Stack Overflow has revolutionized the way developers seek help and provide solutions to programming questions. It acts as a knowledge-sharing hub, making it easier for developers to share their expertise and find answers to their problems. By sharing their personal experiences and anecdotes, contributors to the platform offer practical insights into the application of programming knowledge that is more personal than AI-generated advice. The site serves as a comprehensive resource for developers seeking help beyond their local communities. Stack Overflow users include those who ask questions, those who answer them, and those who browse existing Q&A posts for help.

Stack Overflow has attracted significant research interest from both the information systems community (Aaltonen and Wattal 2020; Bornfeld and Rafaeli 2019; Lee et al. 2019; Sha et al. 2022) and the empirical software engineering community. Many of the previous studies focused on different properties of the content on Stack Overflow such as text, code, and metadata (Baltes et al. 2018; Treude et al. 2011; Wu et al. 2019) and its relationship to other platforms such as GitHub (Abdalkareem et al. 2017; Baltes and Diehl 2019; Vasilescu et al. 2013; Zhang et al. 2019). However, there is a lack of research that examines factors from the user's perspective when selecting a code snippet. Answer selection is critical because even if developers find a relevant post that addresses their problem, they still need to choose an answer that includes a code snippet that can solve their specific problem. Prior research, including that of Treude and Robillard (2017), has identified poor comprehensibility of Stack Overflow code snippets as a common problem that complicates their reuse. Potential risks include copying poorly understood code, which can lead to bugs (Abdalk-

[1] https://news.ycombinator.com/item?id=41364798

[2] See also: https://blog.pragmaticengineer.com/are-llms-making-stackoverflow-irrelevant/; https://meta.stackoverflow.com/q/422392/1974143

areem et al. 2017), security vulnerabilities (Van Der Linden et al. 2020), and licensing issues (Baltes and Diehl 2019).

To address this research gap, this study investigates the factors that Stack Overflow users consider when selecting code snippets and solutions to their programming problems. Specifically, it examines the role of code style features, such as the type of source code comments (block, inline, or none) and surface-level cues, including answer position and score. Improving the understanding of code snippets with comments is particularly relevant as users increasingly rely on Stack Overflow for complex questions, while simpler questions are often handled by AI tools (Xue et al. 2023). The following research questions guided the study:

RQ1 What is the impact of code comments on the perceived helpfulness of code snippets on Stack Overflow?
RQ2 Does the impact of code comments on perceived helpfulness differ between novices and experts?
RQ3 Does the position of a Stack Overflow answer within a Stack Overflow thread affect perceived helpfulness?
RQ4 Does the presence of answer scores amplify the effect of position on perceived helpfulness?

To investigate these questions, we conducted an online experiment with 91 participants in a simulated Stack Overflow environment. Participants rated code snippets that varied in comment type, answer position, and score. We also examined whether programming experience (novice vs. expert) moderates the perceived helpfulness of different types of comments and surface features. The results show that code snippets with comments were generally perceived as more helpful than those without comments, with block comments receiving the highest helpfulness ratings. This effect was particularly pronounced among novices, who rated block comments as significantly more helpful than inline comments. In contrast, surface-level cues such as answer position and answer score had no significant effect on perceived helpfulness. These findings contribute to a better understanding of how developers evaluate shared code snippets and suggest that code comments—especially block comments—play a crucial role in enhancing the perceived helpfulness of programming answers on community-driven platforms.

2 Related work and theoretical background

In this section, we summarize related work on Stack Overflow as a social Q&A platform for software developers, as well as theories and empirical studies on code reading and comprehension.

2.1 Stack Overflow as a social Q&A platform for software developers

Social media has changed the way software developers communicate online and coordinate software development activities (Storey et al. 2017). Social communities for software developers, such as GitHub, Twitter/X, or Stack Overflow, provide participants with quick access to knowledge and expertise shared by other developers. These platforms enable various activities, including searching for projects hosted by other developers, contributing to projects, hosting one's own projects, participating in discussions about programming technologies, and following repositories of interest. Together, individuals form a community around their shared interests on social developer platforms (Vasilescu et al. 2013). Technical Q&A platforms such as Stack Overflow have become increasingly important to software developers as a means of knowledge sharing. As one of the most popular Q&A sites in the world, Stack Overflow serves as a community-based platform for collaborative information sharing spanning diverse programming languages. Over the years, Stack Overflow has accumulated a vast amount of programming knowledge consisting of code snippets accompanied by natural language explanations. The popularity and significance of Q&A forums as a source of support for software developers to solve their development problems are substantial. The primary function of Stack Overflow is to provide a platform where users can ask questions and receive answers from other users. To improve the quality of both questions and answers, users have the ability to comment on and edit each other's posts. In addition, users can earn reputation points and badges for their contributions, e.g., if other users upvote their content.

Researchers have conducted studies on questions, answers, and code examples on Stack Overflow to examine various aspects related to the quality of answers, including the types of questions and comments (Nasehi et al. 2012; Treude et al. 2011; Zhang et al. 2019), their helpfulness (Nasehi et al. 2012), and the characteristics of good and bad question-answer relationships (Baltadzhieva and Chrupała 2015; Duijn et al. 2015). In addition, research has been conducted on coding style and code fragments (Bafatakis et al. 2019; Busjahn et al. 2015; Lee et al. 2013; Treude and Robillard 2017), as well as developer and user behavior observed on Stack Overflow (Aaltonen and Wattal 2020; Bornfeld and Rafaeli 2019; Lee et al. 2019; Van Der Linden et al. 2020). Furthermore, researchers have applied topic analysis and mining techniques to study the reuse of code snippets (Abdalkareem et al. 2017; Baltes and Diehl 2019; Wu et al. 2019). Moreover, studies have analyzed the characteristics of good and bad quality code snippets (Baltadzhieva and Chrupała 2015; Duijn et al. 2015). Duijn et al. (2015) examined code snippets and the code-to-text ratio on Stack Overflow, analyzing metrics to assess their impact on quality. Other studies have revealed that Stack Overflow content has been a major source for training large language models (Nasr et al. 2023).

A fundamental issue that stands out in the literature is that code snippets on Stack Overflow can be difficult to understand, making it challenging for developers to reuse parts of the code, potentially leading to security issues. Treude and Robillard (2017) conducted a study to investigate the extent to which developers perceive Stack Overflow code snippets as self-explanatory, identifying several issues such as the organization of the code, naming, incompleteness, quality, domain, clutter, and rationale. Van Der Linden et al. (2020) researched how developers use Stack Overflow as a resource and their interactions with code snippets, focusing on security aspects. They found that developers often select code snippets based on additional information, such as response score and acceptance status, without considering the security of the snippet. Reusing code snippets from Stack Overflow is common also among experienced developers. However, on average, software that utilizes such snippets has a higher percentage of bugs after code reuse (Abdalkareem et al. 2017). Baltes and Diehl (2019) found that developers frequently reuse content from Stack Overflow without the required attribution, making it difficult or even impossible to go back to the original post in case problems arise. Many Stack Overflow code snippets are incomprehensible, of low quality, or require extensive customization for effective reuse (Wu et al. 2019). Consequently, the following section examines the factors that contribute to the readability and comprehensibility of source code.

2.2 Code reading and comprehension

Source code is a set of computer instructions written in text form using a language that can be understood by humans. Developers often face time-sensitive situations that require rapid product development or bug fixes. In such cases, they rely on existing code to solve their problems (Alarcon et al. 2016). Code transparency refers to its readability and reproducibility. In a study by Alarcon et al. (2016) on transparency, programmers emphasized the importance of elements such as organization, style, and architecture. Among these elements, code style, including comments, was particularly emphasized. Code comments are added to improve readability and support future maintenance activities. The usefulness of comments depends on the programming experience of the reader (Alarcon et al. 2016). Program readability, which refers to the process by which developers understand source code, is a critical aspect of software development. The readability of code differs from that of human-readable text or natural languages due to its structured nature and the inclusion of various elements such as design, documentation, and logic (Buse and Weimer 2009). The higher the readability of the code, the faster and more accurately a programmer can extract critical information from the program text.

The literature on code comprehension has identified different approaches to reading code. Researchers have proposed several cognitive models (Fagerholm et al. 2022) of the reading process to explain how software developers approach the process of code comprehension. These models include *top-down*, *bottom-up*, *as-needed*, and *integrated comprehension* approaches (Dunsmore et al. 2000).

Brooks (1983) has proposed the *top-down theory* of code comprehension. According to this theory, software developers begin by forming general hypotheses about the purpose of the code. In the next step, developers attempt to verify these hypotheses by scanning the code for specific structures or operations, which Brooks (1983) calls "beacons" that programmers use to recognize known structures and operations in the code. According to Brooks (1983), code and its documentation can be thought of as a collection of beacons, such as comments, variables, or indentation. When these indicators are found, they confirm the presence of expected structures or operations and validate the developer's initial hypotheses. However, if no indicators are found, the programmer must examine the code more closely and may need to revise or reject their hypotheses. Understanding continues in iterative rounds, in which software developers repeatedly formulate and verify their hypotheses until they have a complete understanding of the entire code (Brooks 1983; Dunsmore et al. 2000).

The *bottom-up theory* suggests that software developers start by examining small pieces of code, gradually combining them to form higher levels of abstraction, as their initial knowledge is insufficient to identify beacons (Shneiderman 1977). In this approach, semantic relationships are grouped into chunks that serve as building blocks for higher-level chunks. The process continues as more chunks are constructed and relationships between existing chunks are recognized, leading to the formation of larger, higher-level chunks. This iterative process helps the software developer form a general hypothesis about the purpose of the program (Dunsmore et al. 2000; Shneiderman 1977).

Littman et al. (1987) proposed the *as-needed approach*, in which software developers aim to understand only the parts of the code necessary to make successful changes, thereby minimizing the amount of code they need to understand. This strategy focuses on selectively understanding specific components rather than the entire code.

The *integrated model of code comprehension* combines elements of top-down and bottom-up strategies and suggests that software developers alternate between these strategies depending on the specific context or their familiarity with the code (Von Mayrhauser and Vans 1995). When the code is familiar, developers tend to use a top-down approach. In contrast, in scenarios where the code is unfamiliar, such as when reading code snippets on Stack Overflow, the bottom-up approach is more frequently used (Dunsmore et al. 2000).

Code style is highly relevant for code readability. Several programming languages provide style guides that outline coding conventions and serve as standard libraries for writing code. For the Python programming language, *PEP 8 - Style Guide for Python Code* describes coding conventions to ensure code consistency (Van Rossum et al. 2001). Violations of the style guide have a negative impact on code readability (Lee et al. 2013). Moreover, Stack Overflow posts with Python code snippets that adhere to the coding style guide and have fewer violations per statement are preferred by the community (Bafatakis et al. 2019).

Beyond code style, code comments, which are the main focus of this paper, are particularly relevant for code comprehension. Pascarella and Bacchelli (2017) have highlighted the importance of source code comments as an information source for software developers. Comments help improve the readability, comprehensibility, and maintainability of the code (Pascarella and Bacchelli 2017). Source code and comments are the two most important artifacts for understanding a system (de Souza et al. 2005).

More recently, software engineering researchers have studied aspects such as the effect of functional decomposition on code comprehensibility, without finding a connection (Tempero et al. 2024). Sergeyuk et al. (2024) have studied whether existing code readability models are aligned with developers' comprehensibility ratings for AI-generated code, considering the models proposed by Buse and Weimer (2008), Posnett et al. (2011), Dorn (2012)[3], Mi et al. (2022), and Scalabrino et al. (2018). However, they found that readability assessments of AI-generated code differed between these models and that the correlation with human comprehensibility evaluations was low. Fakhoury et al. (2019) found that existing readability models do not capture readability improvements as documented in open source GitHub projects. Etgar et al. (2022) have studied the connection between the information contained in function and variable names and their memorability. They found that more informative names are easier to remember.

Moreover, researchers have studied "atoms of confusion", that is, small patterns of source code that might be misinterpreted, in languages such as C/C++ (Gopstein et al. 2018) and Java (Langhout and Aniche 2021). Stapleton et al. (2020) studied the comprehensibility of human-generated and machine-generated code summaries and found that their participants performed significantly better when using human-written summaries. Wiese et al. (2019) found that code written using "expert" patterns is considered more elegant, but their comprehensibility is not necessarily different from "novice" patterns.

[3] https://web.eecs.umich.edu/~weimerw/students/dorn-mcs-paper.pdf

Regarding code comments, Jabrayilzade et al. (2021) developed a taxonomy of 11 inline comment smells based on a multivocal literature review and 899 inline comments from three open-source Java projects that the authors manually labeled. The most common smells were misleading comments that did "not accurately represent what the code does" and obvious comments that "restate what the code does in an obvious manner" (Jabrayilzade et al. 2021). Based on an online survey with 102 participants, Huang et al. (2023) found that inline comments are written more frequently than block comments. Regarding perceived helpfulness, the authors were unable to find a considerable difference. With respect to automatic comment generation, they found that existing models perform better for block comments. Finally, Wang et al. (2024) found that TODO comments in open-source projects are often of low quality.

2.3 Summary

Although there is a considerable body of knowledge on different properties of Stack Overflow posts (see Section 2.1) and code comprehension in general (see Section 2.2), the impact of different ways of presenting and documenting source code in the typically short code snippets on Stack Overflow has not yet been studied. Stack Overflow's fragmented code-snippet-based knowledge sharing approach differs from traditional code comprehension, which usually spans larger code bases, potentially even large industrial software projects. For example, results from whole open-source projects regarding the differences between inline and block comments do not necessarily generalize to individual code snippets in Stack Overflow posts. In the specific context of Stack Overflow, users' assessment of the perceived helpfulness of code snippets can influence their decision to reuse. A better understanding of perceived helpfulness has implications beyond code on Stack Overflow, as AI-based assistants also offer the option to generate multiple solution proposals or re-generate solutions that users are not satisfied with.

3 Hypotheses

In the following, we describe the hypotheses that we formulated based on our research questions. These hypotheses guided the design of our study.

3.1 Code comments and inline vs. block comments (RQ1)

The importance of code comments is supported by the study by Misra et al. (2020), who analyzed the relationship between code comments in Python code and problem solving through various experiments. The study concluded that comments play an important role in problem solving within coding projects and that increasing the percentage of relevant comments, along with source code comments, can reduce the average time it takes to solve a problem. Previous studies also support the claim that code comments play a crucial role in the selection of code snippets on Stack Overflow. Nasehi et al. (2012) conducted a study on Stack Overflow, examining well-received answers and identifying characteristics of effective examples. Their findings highlighted the importance of code explanations in addition to code snippets, with comments recognized as being as important as code snippets (Nasehi et al. 2012). Moreover, Robillard (2009) conducted a survey of professional developers and found that they value well-structured and complete documentation, as well as a comprehensive set of examples. Users are likely to subjectively perceive greater helpfulness when they encounter code snippets accompanied by comments on Stack Overflow. Based on the previous findings, we formulated the following hypothesis:

> Hypothesis 1a: Code snippets with code comments are generally rated as more helpful than code snippets without code comments.

In the Python programming language, comments are preceded by a hash character (#) and a space. For the context of this study, we ignore multi-line comments and Python docstrings. We distinguish two types of comments used within code snippets: block comments and inline comments. Block comments refer to a sequence of consecutive comment lines, each of which begins with a hash sign. These block comments are similar to method comments (Rani et al. 2023; Wang et al. 2024). Inline comments, on the other hand, appear on the same line as the statement (Van Rossum et al. 2001). Although there is a large body of literature on code comments in general (Rani et al. 2023), only a few studies have specifically compared different types of code comments in terms of their impact on code comprehension and perceived helpfulness. Due to their structural and contextual advantages, block comments are better suited for providing more detailed information and longer explanations than inline comments. Typically placed at the beginning of a section of code, such as a function, class, or module, block comments can provide a comprehensive overview or summary. Their advantage is that they are not limited to the space at the end of a line of code, allowing for more elaborate explanations, including multiple paragraphs. In contrast, inline comments, appropriately positioned next to specific lines of code, are ideal for concise explanations or clarifications and contain fewer words on average (Wang et al. 2024). We suggest that block comments, with their capacity for extended detail, are generally more advantageous than the brevity offered by inline comments. We can draw on the code comprehension literature to support this claim (see, e.g., Dunsmore et al. (2000)). While providing only inline comments is likely to be effective for detailed, line-specific explanations that support bottom-up and as-needed comprehension strategies, block comments can additionally provide general overviews and support top-down comprehension. Because code snippets on Stack Overflow are typically unfamiliar to readers, the bottom-up comprehension approach is often required (Dunsmore et al. 2000); the as-needed approach is likely to be applied as well. Thus, inline comments provide immediate, line-specific explanations that help piece together the lower-level functionality that is essential to the bottom-up approach. However, for a deeper understanding of the entire proposed code solution, block comments can help explain the overarching design and purpose of code sections, not just the intentions behind specific lines or statements, which inline comments provide. In addition to code comprehension, users may perceive detailed and carefully crafted block comments as more "polite" than brief inline comments. This perception of politeness is an important factor in answer selection on Stack Overflow (Xue et al. 2023). In summary, we hypothesize that:

> Hypothesis 1b: Code snippets with block comments are generally rated as more helpful than code snippets with inline comments.

3.2 Interaction effect with expertise—experts vs. novices (RQ2)

Novices and experts exhibit differences in their code-reading behavior, so it is important to consider this distinction in expertise when formulating hypotheses about users' preferences for code comments and the specific types of comments they prefer on Stack Overflow. A study conducted by Busjahn et al. (2015) used eye-tracking to investigate code reading skills and found that novices read code in a less linear manner compared to natural language text. Additionally, experts were found to read code even less linearly than novices, indicating a difference in the reading process between code and natural language text. In examining how subjects view an algorithm, Crosby and Stelovsky (1990) found that the eye movements of experts and novices differed when viewing the English and Pascal versions of an algorithm. They found that experts spent more time looking at complex statements. They also discovered that while both groups spent a lot of time looking at comments, novices spent significantly more time looking at comments (Crosby and Stelovsky 1990), which implies that comments seem to be more important to novices. Learning a programming language involves the acquisition of knowledge structures to apply problem-solving skills (Gilmore 1990). Programming expertise involves different cognitive processes that, combined with changes in knowledge, can lead to the adoption of different methods for solving specific programming problems. These different approaches can be described as strategies, and experienced programmers have a more extensive repertoire of strategies than novices when programming (Gilmore 1990). An exploratory study conducted by Chatterjee et al. (2020) aimed to understand how novice software engineers focus on the information presented in responses to questions on Stack Overflow. The results showed that novice software engineers only pay attention to 27% of the code and 15 to 21% of the text in a Stack Overflow post as they try to understand the relevant information and determine how to apply it to their specific context. As Pascarella and Bacchelli (2017) discuss, code comments serve as a valuable source of information. Gilmore (1990) argues that the choice of a code snippet is influenced by the developer's knowledge. Based on the existing literature, which suggests that novice software developers benefit from more explanatory comments, while experienced programmers tend to prefer fewer or no source code comments, we propose the following hypothesis:

> Hypothesis 2a: Individuals with less programming experience perceive code snippets with comments as more helpful than those without comments.

Furthermore, we argue that block source code comments have the potential to provide novice software developers with the additional information they need compared to shorter inline comments. In contrast, experienced programmers tend to prefer inline source comments, which are typically shorter and cause less disruption to the code-reading process. Based on these considerations, we formulate the following hypothesis:

> Hypothesis 2b: Individuals with less programming experience perceive code snippets with block comments as more helpful than those with inline comments.

3.3 Ordering effects (RQ3)

The order of answers within a Stack Overflow thread can influence their perceived helpfulness. Typically, Stack Overflow sorts answers in descending order by answer score. Murphy et al. (2006) observed that an item's position affects its memorability, with top items enjoying a memory advantage due to early processing and less competition from subsequent items, a phenomenon known as the primacy effect. In support of this, Galesic et al. (2008) argued that individuals tend to process horizontal lists from top to bottom, resulting in longer memory retention for initial options. Their studies showed that participants looked longer at the first option on a list, especially for longer lists. This suggests that due to cognitive and behavioral biases, code options at the beginning of a list are more likely to be selected, regardless of code quality or comment type. This leads to the hypothesis that:

> Hypothesis 3: Regardless of the content, answers positioned earlier in the thread are generally perceived as more helpful by users.

3.4 Answer score (RQ4)

The Stack Overflow platform uses a scoring system for both questions and answers, allowing users to vote on the helpfulness of a question or answer. Several studies have looked at different aspects of the Stack Overflow community and platform, such as the accepted answer score, the scoring system, and the user reputation. For example, Gantayat et al. (2015) studied Stack Overflow posts and explored the interaction between accepted answers and upvotes. Their results showed that accepting an answer led the community to vote for that answer instead of an alternative answer with a similar or higher score prior to acceptance. Van Der Linden et al. (2020) conducted the first large-scale between-groups experiment and observed that answer details, called surface features, have a significant impact on the selection of both secure and insecure code snippets. The observed positive effect of the answer score on the acceptance and selection of code snippets also has its drawbacks. Due to Stack Overflow's decentralized structure, its voting and reputation system may be susceptible to manipulation (Chen et al. 2023). Answer details, such as the 'accepted answer' mark, user reputation scores, and answer scores, have also been found to mislead users into accepting insecure coding advice, inadvertently promoting vulnerable code (Meng et al. 2018). Building on previous studies that highlight the importance of answer scores in code snippet selection, the following hypothesis was formulated regarding the perceived helpfulness of code snippets:

> Hypothesis 4: Users perceive answers as more helpful when they not only appear earlier in the thread, but also have a higher answer score, regardless of their content.

4 Method

Our empirical study utilized an online experiment that simulated the Stack Overflow environment, followed by a post-survey. Participants were asked to rate answers based on their perceived helpfulness within a simulated Stack Overflow environment. The *answer position* and *source code comment type* (block comment, inline comment, no comment) were used as within-subjects factors in the experimental design. In addition to *answer position*, the experiment used the *answer score* as a between-subjects factor, with one group receiving the score treatment, displayed on the left side of the Stack Overflow answers, while the control group received the answers in the same order, but without scores. Participants were randomly assigned to either the treatment or control group. This experimental setup allowed us to isolate the effect of scores from content-based factors such as code comments, enabling a controlled comparison of perceived helpfulness across conditions. For each scenario, that is, for each Stack Overflow question, the authors developed three different code snippets, inspired by real Stack Overflow posts related to the question. Each answer consisted of a short explanation text and a source code snippet. We counterbalanced the comment types with source code snippets. Since each participant was presented with three code snippets for each question, this resulted in six possible combinations that were randomly selected to be presented to a participant for each scenario. To vary the order of the answers, we presented the three answers to the participants in random order.

4.1 Experimental materials: Stack Overflow simulation

In the experiment, participants were instructed to imagine searching for answers on Stack Overflow: *Suppose that you are developing a Python program and run into a specific programming problem. While searching for your problem on the web, you find an answered thread on Stack Overflow that addresses your problem.* Participants were instructed to read through all available answers, along with accompanying code snippets, and then rate each answer based on its helpfulness.

The *Stack Overflow simulation* consisted of three Stack Overflow thread scenarios, each with the same structure and layout (as depicted in Fig. 1), inspired by existing posts on the platform. Rather than randomly sampling threads, we constructed three Python-related scenarios based on real Stack Overflow discussions, but adapted them to ensure a uniform style and a manageable level of complexity. Each set of code snippets was designed to compile and produce similar outcomes, employing various functions, variables, or packages. Each scenario began with a question posed to the forum by a fictitious Stack Overflow user,

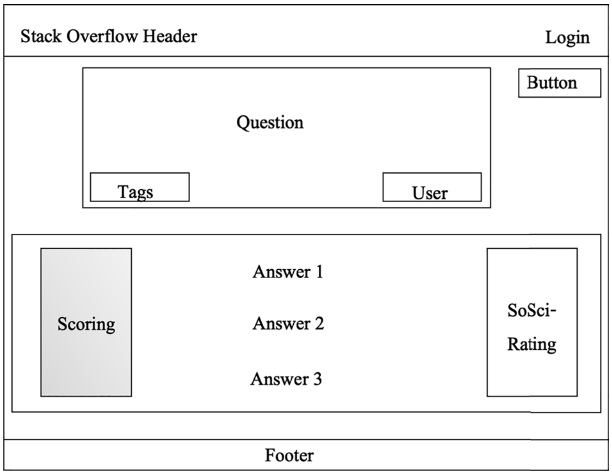
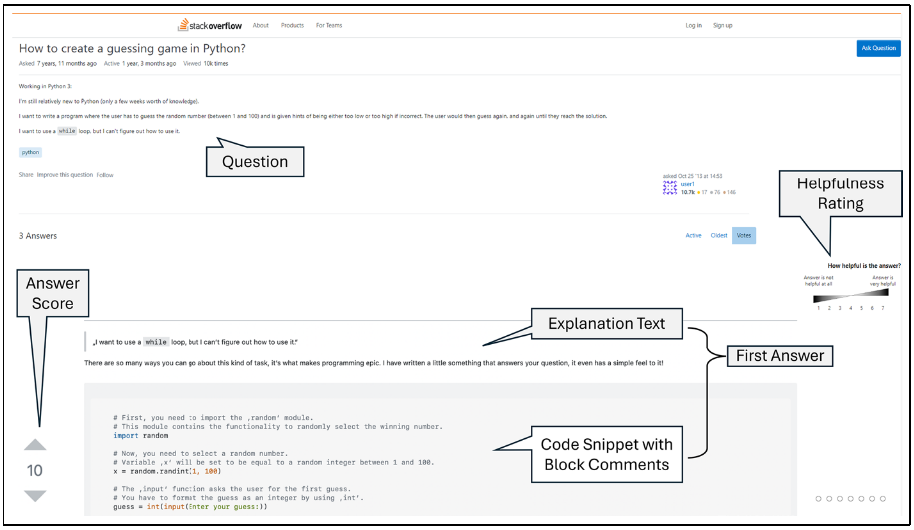


Fig. 1 Simulated Stack Overflow design (abstract structure)



How to create a guessing game in Python?

Asked 7 years, 11 months ago Active 1 year, 3 months ago Viewed 10k times

Working in Python 3:

I'm still relatively new to Python (only a few weeks worth of knowledge).

I want to write a program where the user has to guess the random number (between 1 and 100) and is given hints of being either too low or too high if incorrect. The user would then guess again, and again until they reach the solution.

I want to use a `while` loop, but I can't figure out how to use it.

python

Share Improve this question Follow

asked Oct 25 '13 at 14:53
user1
10.7k ● 17 ● 76 ● 146

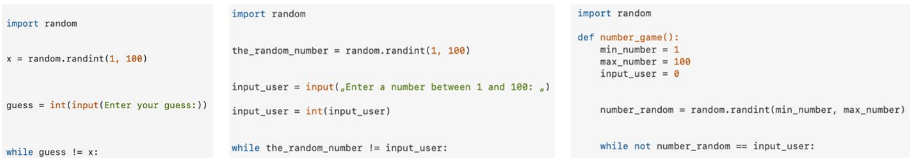
3 Answers

„I want to use a `while` loop, but I can't figure out how to use it."

There are so many ways you can go about this kind of task, it's what makes programming epic. I have written a little something that answers your question, it even has a simple feel to it!

```
# First, you need to import the ‚random' module.
# This module contains the functionality to randomly select the winning number.
import random

# Now, you need to select a random number.
# Variable ‚x' will be set to be equal to a random integer between 1 and 100.
x = random.randint(1, 100)

# The ‚input' function asks the user for the first guess.
# You have to format the guess as an integer by using ‚int'.
guess = int(input(Enter your guess:))
```

Fig. 2 Simulated Stack Overflow design (labeled screenshot)



```
import random

x = random.randint(1, 100)

guess = int(input(Enter your guess:))

while guess != x:
```

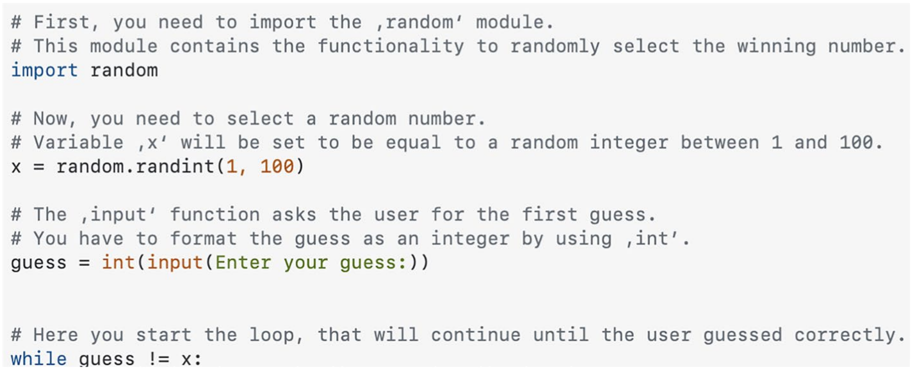
```
import random

the_random_number = random.randint(1, 100)

input_user = input(„Enter a number between 1 and 100: „)

input_user = int(input_user)

while the_random_number != input_user:
```

```
import random

def number_game():
    min_number = 1
    max_number = 100
    input_user = 0

    number_random = random.randint(min_number, max_number)

    while not number_random == input_user:
```

Fig. 3 Excerpts from three answers without source code comments

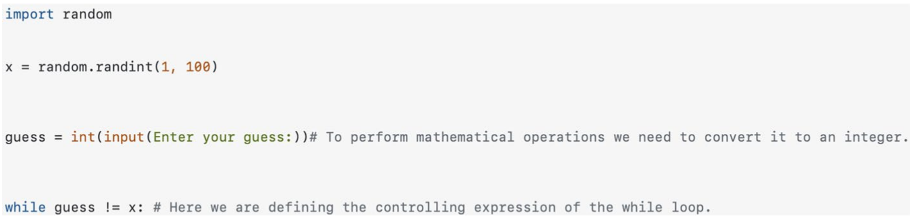
representing a specific problem. Figure 2 shows the first Stack Overflow scenario with the question *"How to create a guessing game in Python?"* and the corresponding problem of the Stack Overflow user identified as *"user1"*, along with the first answer. The experiment included three different Stack Overflow scenarios, one asking how to print the fourth most common word from a *"Romeo and Juliet"* excerpt using Python, and another asking for guidance on how to handle exceptions in Python. We developed questions and answers for the threads using Python because it is among the most popular programming languages on Stack Overflow (Stack Exchange Inc 2025). The code snippets were developed in a way that each code snippet yields the same or very similar results. The question was followed by three answers that provided a solution to the problem posed. Figure 3 shows snippets from three answers from the control group without source code comments for the first Stack Overflow scenario to the question *"How to create a guessing game in Python?"*. Figure 4 provides an excerpt from one of the answers with block source code comments, and Fig. 5 shows an excerpt from one of the answers with inline source comments.

For the treatment group with answer scores, scoring buttons were displayed on the left side (see Fig. 2). The answers were displayed to the participant in descending order of the scores. Starting with 10, the highest score, followed by a score of 3, and finally a score of 0. These scoring numbers are realistic because the average score for an answer is 1.8 (Nasehi et al. 2012).



```
# First, you need to import the ‚random‘ module.
# This module contains the functionality to randomly select the winning number.
import random

# Now, you need to select a random number.
# Variable ‚x‘ will be set to be equal to a random integer between 1 and 100.
x = random.randint(1, 100)

# The ‚input‘ function asks the user for the first guess.
# You have to format the guess as an integer by using ‚int‘.
guess = int(input(Enter your guess:))


# Here you start the loop, that will continue until the user guessed correctly.
while guess != x:
```

Fig. 4 An excerpt from an answer with block source code comments



```
import random


x = random.randint(1, 100)



guess = int(input(Enter your guess:))# To perform mathematical operations we need to convert it to an integer.



while guess != x: # Here we are defining the controlling expression of the while loop.
```

Fig. 5 An excerpt from an answer with inline source code comments

After reading the scenario and answer options, participants were asked to rate the helpfulness of each answer option. This was done using a 7-point Likert scale item ranging from “Answer is not helpful at all” to “Answer is very helpful.” The scale was prominently displayed to the right of each answer code snippet (see Fig. 2).

5 Experimental procedure and post-survey

The online experiment was implemented using the questionnaire platform SoSci[4] and was divided into three main parts: the introduction, the Stack Overflow simulation, and the post-survey. In the first part, the introduction, the participant was presented with a consent form that explicitly stated that participation was voluntary and anonymous, and that all data collected would be kept confidential. In addition, participants were given technical instructions, such as the requirement to use a PC or laptop (not a mobile device) and to ensure that the browser zoom was set to 100%. After the Stack Overflow simulation, the third part was a post-survey divided into three sections: (a) Stack Overflow questions, (b) general control questions related to expertise, and (c) demographic information. In the first part of the post-survey, participants were asked to rank Stack Overflow elements displayed as cards in order of importance when evaluating an answer. They used drag-and-drop to position each item on a scale from least to most important. The question was asked twice: First, we asked about the elements used in the experiment, that is, the answer score, blank code (i.e.,

[4] https://www.soscisurvey.de/

code without comments), explanation text, inline source comments, and block source comments. By “explanation text,” we mean a description of the code placed at the beginning of an answer that is not formatted as an inline or block comment. Second, we asked about the elements of Stack Overflow in general, which included the same elements as before, along two additional ones: the accepted answer marked as the best answer by the questioner and the answerer’s reputation score. We asked several questions about the user’s prior experience and expertise. Participants were asked about their active or passive experience with Stack Overflow, frequency of use, whether they had used a code snippet from the Q&A page, and their reasons for using Stack Overflow. They also provided information about their programming skills, including years of experience. Experience with different programming languages was rated on a 6-point Likert scale item from “never”, “rarely”, “seldom”, “sometimes”, “occasionally” to “most of the time.” An open text field was provided to list additional programming languages not included. In the last section of the questionnaire, participants provided demographic information such as gender, age, country of origin, education, and occupation or field of study.

5.1 Data-collection, sample, and data preparation

In order to collect empirical data from software development students and professionals, we chose several channels to recruit participants for this online study. First, the mail delivery service of a large European university was used. Specific faculties and study programs—such as Mathematics and Computer Science, Physics, and Information Systems—were selected as recipients of the invitation email. An email was sent to subscribers of the survey mailing list who belonged to the selected student groups, including a short explanation of the study and a link to the Stack Overflow online experiment.

The study was also posted on a local hackerspace website and on LinkedIn. In addition, individuals with programming backgrounds from the authors’ personal networks—including friends, work colleagues, and family members—were invited to participate in the study to ensure a diverse mix of novice and professional software developers.

Only fully completed questionnaires were considered for further analysis. The sample consisted of 91 participants with a mean age of 26.22 years ($SD = 7.58$), of whom 79.1% identified as men, 18.7% as women, and 2.2% preferred not to disclose their gender.

To prepare for the analyses and test the hypotheses, we first divided the participants into novices and experts based on their programming skills, as determined by their responses to the post-survey. Novices were those with none, less than one year, or one year of programming experience, while experts had two to ten or more years of experience. Of the 91 participants, 61 (67%) were classified as experts and 30 (33%) as novices.

This imbalance can be attributed to the data collection process, as most respondents were recruited through the university mailing list—where many computer science students already have considerable programming experience—and through LinkedIn or colleagues, where we aimed to recruit professional software developers or those with similar backgrounds.

6 Results

In the following, we summarize our results on the helpfulness of different types of source code comments in Stack Overflow posts, as well as the perceived importance of various Stack Overflow elements.

6.1 Helpfulness of source code comment types (RQ1 and RQ2)

The analyses are based on the cumulative sample of three answer comments across three scenarios per participant, resulting in a total of $n_{\text{accumulated}} = 819$ observations of answer comment helpfulness ratings ($n = 91 \times 3 \times 3$).

Figure 6 shows how programming novices and experts rated the helpfulness of different types of source code comments. Novices rated block comments as slightly more helpful ($M = 5.60$, $SD = 1.66$) than experts ($M = 5.48$, $SD = 1.50$), while the average ratings for inline comments and no comments were lower for novices than for experts. Specifically, experts rated inline comments ($M = 5.10$, $SD = 1.42$) as more helpful than novices ($M = 4.70$, $SD = 1.62$). Experts also rated uncommented code as more helpful on average ($M = 4.39$, $SD = 1.63$) than novices ($M = 4.07$, $SD = 1.62$). Overall, both groups rated commented code as more helpful than uncommented code snippets.

We used linear mixed models provided by the `lme4`[5] and `lmerTest`[6] packages in R[7] to analyze the fixed factors: comment type (baseline = no comments, block comments, inline comments), expertise (experts vs. novices), position (first, second, third answer), and the presence of answer scores (no answer score vs. presence of answer score; first answer: score 10, second answer: score 3, and third answer: score 0). The dependent variable was the

[5] https://cran.r-project.org/web/packages/lme4/

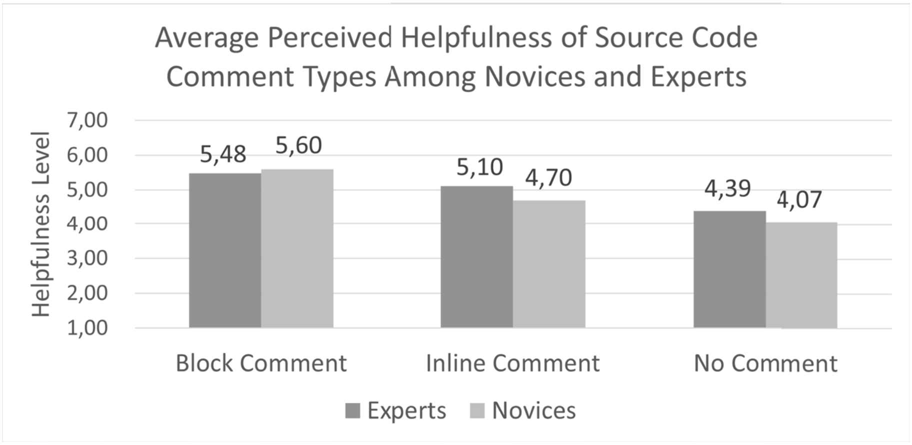


Fig. 6 Comparison of the mean helpfulness of different types of source code comments

[6] https://cran.r-project.org/web/packages/lmerTest/

[7] https://www.r-project.org

perceived helpfulness of an answer. We also examined interaction effects between comment type and expertise, as well as between position and score.

A second model ('Model 2' in Table 1) was calculated to allow for a more direct comparison of block and inline comment types. This model excluded the "no comment" condition and used block comments as the baseline. Both models included two random intercepts, one for participant and one for code snippet. The intra-class correlation (ICC) was 0.23 in the first model and 0.24 in the second model, indicating that a substantial proportion of the

Table 1 Linear mixed models for perceived helpfulness

	Model 1		Model 2	
	comparing block and inline comments with no comments		comparing block with inline comments	
Predictors	*Estimates*	*p*	*Estimates*	*p*
(Intercept)	4.08	<0.001	5.57	<0.001
Comment Type: Block Comments vs. No Comments	1.51	**<0.001**	*not included in model 2*	
Comment Type: Inline Comments vs. No Comments	0.62	**0.003**	*not included in model 2*	
Comment Type: Inline Comments vs. Block Comments	not included in model 1		−0.88	**<0.001**
Expertise (Experts vs. Novices)	0.32	0.172	−0.1	0.674
Position	0.02	0.777	0.05	0.524
Presence of User Ratings	0	0.997	0.11	0.551
Comment Type: Block Comments vs. No Comments * Expertise (Experts vs. Novices)	−0.43	**0.09**	*not included in model 2*	
Comment Type: Inline Comments vs. No Comments * Expertise (Experts vs. Novices)	0.09	0.723	*not included in model 2*	
Comment Type: Inline Comments vs. Block Comments * Expertise (Experts vs. Novices)	*not included in model 1*		0.5	**0.042**
Position * Presence of User Ratings	-0.03	0.793	−0.02	0.897
Random Effects				
σ^2	1.9		1.77	
τ 00 Participant	0.46		0.49	
τ 00 Code Snippet	0.1		0.06	
ICC	0.23		0.24	
N	91 participants		91 participants	
	9 code snippets		9 code snippets	
Observations	811		539	
Marginal R^2/Conditional R^2	0.099/0.302		0.040/0.269	

variance in helpfulness ratings is explained by differences between participants and code snippets. The conditional R^2, reflecting the variance explained by both fixed and random effects, was 0.302 for the first model and 0.269 for the second, indicating that approximately 30% and 27% of the variance, respectively, could be explained by the model.

The position of the answer, the presence of an answer score, and their interaction did not have a significant effect on the perceived helpfulness of a code snippet in either model. Expertise was also not a significant factor; thus, novices did not generally rate the helpfulness of comments higher than experts.

Furthermore, the position of the comment (first, second, or third answer) had no significant influence on helpfulness ratings.

The results of Model 1 suggest that including block comments (Estimate = 1.51, $p < 0.001$) or inline comments (Estimate = 0.62, $p = 0.003$) significantly increases the perceived helpfulness of answers compared to uncommented source code.

Notably, novices perceive block comments as particularly helpful in comparison to no comments—even more so than experts—as indicated by a marginally significant interaction effect (Estimate = -0.43, $p < 0.09$), as shown in Fig. 6. This suggests that the effect of block comments on perceived helpfulness depends on the expertise level of the user.

In contrast, the interaction between expertise and inline comments vs. no comments was not significant (Estimate = 0.09, $p = 0.723$). Model 2 further shows that, when comparing block comments to inline comments, block comments are rated as significantly more helpful (Estimate = −0.88, $p < 0.001$). Furthermore, the interaction between expertise and inline comments vs. block comments shows a significant positive effect on perceived helpfulness (estimate = 0.50, $p = 0.042$). Experts and novices rated block comments as more helpful than inline comments, with the difference being more pronounced for novices (see Fig. 6).

Table 1 summarizes the results of the linear mixed-effects models for perceived helpfulness.

6.2 Perceived importance of Stack Overflow elements (RQ3 and RQ4)

Next, we present the results of the post-survey, which assessed participants' perceptions of the importance of various information elements used in Stack Overflow threads. Importance ratings were collected using a fine-grained scale ranging from 0 to 100. Participants were first asked to rate how important each element used in the experiment was when choosing an answer. A subset of the sample, the participants who had prior experience with Stack Overflow either as active contributors or passive users (85 participants, 93.4%), were asked to rate these elements again, this time in the context of the real Stack Overflow platform. In this second assessment, participants were presented with two additional elements that are present on the actual platform but were not included in the experiment: the answerer's reputation score and the accepted answer (i.e., the answer marked as the best by the questioner).

As shown in Fig. 7, the five elements common to both the simulated experimental platform and the real Stack Overflow platform were ranked similarly in terms of perceived importance. Based on descriptive mean values, *explanation text* was rated as the most important element, followed by *block source code comments*. *Inline code comments* ranked third, and the *answer score* ranked fourth. The element rated lowest was *blank code*. Figure 7 also includes ratings for two informational elements that are exclusive to the real Stack Overflow platform and were not part of the experimental interface. Among these,

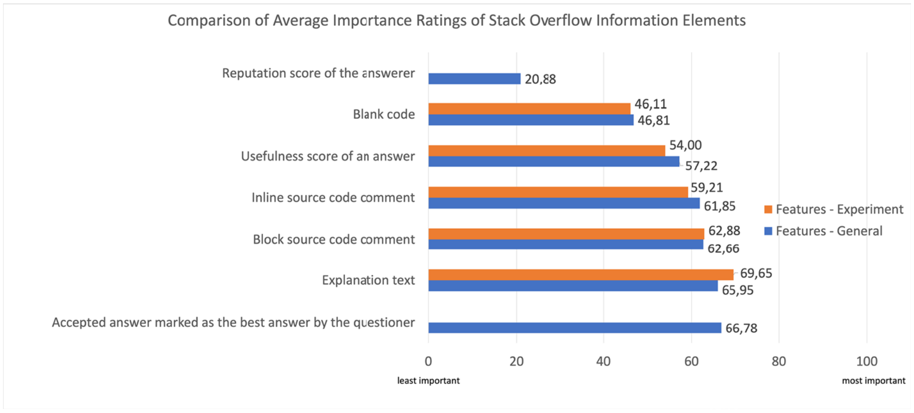


Fig. 7 Comparison of average importance ratings of Stack Overflow information elements

the *accepted answer* was rated as the most important feature for evaluating an answer ($M = 66.78$, $SD = 27.06$), whereas the *reputation score of the answerer* was rated as the least important ($M = 20.87$, $SD = 24.46$).

To statistically assess these differences, a repeated-measures General Linear Model (GLM) was conducted in SPSS on the seven informational elements as perceived in the context of the real Stack Overflow platform. Multivariate tests revealed a statistically significant main effect of the within-subject factor (Pillai's Trace =.779, F(6, 79) = 46.54, $p < .001$), indicating substantial differences across the features.

As Mauchly's test indicated a violation of sphericity ($W = .208$, $p < .001$), Greenhouse–Geisser correction was applied. The within-subject effect remained statistically significant after correction, F(3.93, 330.10) = 31.29, $p < .001$.

Post-hoc Bonferroni-adjusted comparisons showed that the *reputation score* was rated significantly lower than all other features ($p < .001$). Similarly, *blank code* was rated significantly lower than *accepted answers* ($p < .001$), *explanation text* ($p < .001$), and both *inline* ($p = .001$) and *block code comments* ($p = .005$). In contrast, *answer scores*, *accepted answers*, *explanation texts*, and *code comments* (inline and block) did not differ significantly from one another, forming a cluster of consistently high-valued elements. No significant difference was found between inline and block code comments.

A separate repeated-measures GLM was conducted to assess the perceived importance across the five elements used in the experimental platform. This model also revealed a significant main effect of informational element type (Pillai's Trace =.252, F(4, 87) = 7.32, $p < .001$). Estimated marginal means indicated that *explanation text* received the highest ratings ($M = 69.6$), significantly outperforming both *answer scores* and *blank code*. Inline and block code comments were again rated positively and did not differ significantly from each other.

Comparing both analyses, the model based on the full feature set (real Stack Overflow platform) yielded stronger statistical effects (Pillai's Trace =.779 vs. .252) and a greater number of significant contrasts. This model highlighted particularly negative perceptions of *low reputation* and *blank code*, while further reinforcing the positive evaluations of *explanation text* and *code comments*. Across both analyses, explanations and code comments

Table 2 Summary of research questions, hypotheses, and acceptance

Research Question	H	Description	Acceptance
RQ1: What is the impact of code comments on the perceived helpfulness of code snippets on Stack Overflow?	**H1a**	Code snippets with code comments are generally rated as more helpful than code snippets without code comments.	accepted
	H1b	Code snippets with block comments are generally rated as more helpful than code snippets with inline comments.	accepted
RQ2: Does the impact of code comments on perceived helpfulness differ between novices and experts?	**H2a**	Individuals with less programming experience perceive code snippets with comments as more helpful than those without comments.	rejected
	H2b	Individuals with less programming experience perceive code snippets with block comments as more helpful than those with inline comments.	accepted
RQ3: Does the position of a Stack Overflow answer within a Stack Overflow thread affect perceived helpfulness?	**H3**	Regardless of the content, answers positioned earlier in the thread are generally perceived as more helpful by users.	rejected
RQ4: Does the presence of answer scores amplify the effect of position on perceived helpfulness?	**H4**	Users perceive answers as more helpful when they not only appear earlier in the thread, but also have a higher answer score, regardless of their content.	rejected

consistently emerged as the most valued elements in evaluating the quality of answers on Stack Overflow.

6.3 Interpretation

Table 2 shows which hypotheses were supported and how they relate to the research questions.

Research Question 1 asked whether code comments influence the perceived helpfulness of code snippets on Stack Overflow. The results of the study supported **Hypothesis 1a**, which proposed that code snippets with source code comments were generally rated as more helpful than those without comments. In our experiment, code snippets with block and inline comments received higher helpfulness ratings than those without comments. This finding is consistent with previous research highlighting the importance of source code comments for software developers (Misra et al. 2020; Pascarella and Bacchelli 2017).

Regarding different comment types, our study showed that block comments were generally rated as more helpful than inline comments, supporting **Hypothesis 1b**. This supports our argument that block comments—by providing a broader overview essential for understanding the overall design and purpose of a code snippet, especially when encountering unfamiliar code on platforms like Stack Overflow—are more beneficial than shorter inline comments. They also facilitate top-down comprehension of code (Dunsmore et al. 2000).

The ranking of information elements deemed important for choosing an answer was in part consistent with the direct helpfulness ratings observed during the experiment. The mean rating for commented code was higher than that for blank code without comments. However, the results did not support our expectation that block comments would be rated as more important than shorter inline comments when evaluating an answer, as both were evaluated similarly.

Research Question 2 explored whether expertise (novices vs. experts) moderates the impact of comments on perceived helpfulness. This was tested through Hypotheses 2a and 2b. Although novices did not generally rate source comments as more helpful than experts—leading to a rejection of **Hypothesis 2a**, they did rate block comments as more helpful than inline comments, supporting **Hypothesis 2b**. This finding confirms that novices may indeed require more detailed information from source comments than experts.

Research Question 3 examined whether the position of an answer in a Stack Overflow thread influences its perceived helpfulness when no additional cues such as answer scores are shown. The related **Hypothesis 3**, which proposed that earlier answer positions would increase the perceived helpfulness of answers, was rejected. The rejection of Hypothesis 3 contradicts research on the primacy effect, which has shown that options placed at the top are more likely to be selected, regardless of their content (Galesic et al. 2008). One possible interpretation is that both primacy and recency effects (Murphy et al. 2006), which are commonly observed in psychology, may have influenced user ratings. Another plausible explanation for the lack of effects of position and score could be the small number of response options (only three), which allowed participants to thoroughly read the texts, code, and code comments and incorporate them directly into their helpfulness ratings without relying on heuristics.

Research Question 4 asked whether the presence of answer scores would amplify a potential position effect. The related Hypothesis 4, which predicted that highly rated answers in higher positions would be perceived as more helpful, was also rejected. Furthermore, the answer score was rated as relatively less important in the post-survey ranking of information elements, which was consistent with its lack of influence on the observed helpfulness ratings in the experimental data. These results contradict those of Van Der Linden et al. (2020), who demonstrated in an empirical study that surface features such as upvotes and other factors influence the selection of a code snippet.

The rejection of Hypothesis 3 and 4 is insightful in that it indicates that users prioritize the quality of content over answer scores and position on Stack Overflow. Participants in our study based their judgments primarily on the content of the code and accompanying explanations, rather than on surface-level cues such as answer order or answer scores. This critical approach helps mitigate risks such as manipulated scores (Chen et al. 2023) or the acceptance of unsafe coding advice associated with misleadingly high answer scores (Meng et al. 2018).

7 Limitations and threats to validity

Although this study provides valuable insights into how code comments affect the perceived helpfulness of Stack Overflow answers, several limitations need to be acknowledged. We organize these limitations along the standard validity categories of external, internal, construct, and conclusion validity.

7.1 External validity

First, the experiment was conducted in a simulated Stack Overflow environment. Although care was taken to closely mirror the platform's design and functionality, participant behavior might differ from real-world use where additional contextual factors apply. Our design allowed us to isolate the effect of comment types and surface features such as position and answer score. However, it may have introduced artificiality, for example, by fixing answer scores (10/3/0), omitting answer scores in the control group, or not including other influential features such as the accepted answer badge or common sorting mechanisms (e.g., newest first). As a result, the study does not fully capture the broader dimensions of perceived helpfulness of code comments and user behavior present on the real Stack Overflow platform.

Second, the study relied on a convenience sample recruited through university mailing lists, a local hackerspace, LinkedIn, and the authors' personal networks. Consequently, the exact response rate is unknown, and the sample may not be representative of the broader population of Stack Overflow users. Although we aimed for heterogeneity by including both students and professionals, the majority of participants were male, which may introduce a bias in the generalizability of the findings.

Third, all code snippets were constructed to emphasize a dominant comment type (block, inline, or none), and we avoided mixing comment types in the same snippet. Although this allowed us to examine the effect of distinct styles, it may not reflect real-world practices where mixed comment styles are common.

Moreover, the experiment focused exclusively on Python code snippets and a limited number of programming tasks. Findings may therefore not generalize across different programming languages, problem types, or technical domains.

7.2 Internal validity

Although the experiment tested various comment types and surface cues in a controlled setting, the artificial nature of the environment may have influenced participant judgments. For example, answer scores were fixed or omitted, and other influential elements such as accepted answer badges or user reputation were not shown. These simplifications, while necessary for experimental control, may have altered the natural evaluation behavior of the participants, introducing potential confounding factors.

7.3 Construct validity

Helpfulness ratings reflect subjective perceptions rather than actual reuse behavior or code comprehension. Although perceived helpfulness is a meaningful metric, it may not fully capture actual code comprehension or reuse behavior. Future studies could incorporate complementary methods such as eye-tracking, think-aloud protocols, or comprehension tests to gain more direct insights into how code comments influence understanding and decision-making.

7.4 Conclusion validity

The use of a convenience sample and the absence of a known response rate limit our ability to draw generalizable statistical inferences. Although we employed linear mixed-effects models to account for individual- and item-level variance, the moderate sample size and the imbalance between experts and novices may have limited the detection of smaller interaction effects. Future work should aim to replicate these results in larger and more diverse samples.

8 Discussion and implications

Overall, this study provides valuable insights for both practitioners and researchers, highlighting the importance of code comments and their potential to improve the helpfulness and usability of code snippets in online programming communities.

From a research perspective, as motivated in Section 2.3, the impact of different ways of presenting and documenting the usually short code snippets on Stack Overflow has not yet been adequately studied. Findings from larger open-source projects do not necessarily generalize to the specific context of shorter and isolated snippets. For example, Huang et al. (2023) found no difference between the perceived helpfulness of inline and block comments, whereas we found that block comments were perceived as more helpful.

Since users' assessment of the perceived helpfulness of code snippets can influence their decision to reuse them, our results have implications beyond Stack Overflow. Our study points to the context-dependence of perceived helpfulness and code comprehension, highlighting the need for further studies to understand how users decide to reuse code—from Stack Overflow answers or from outputs of AI-based assistants offering alternative solutions.

From a practical perspective, the findings suggest that developers and contributors on platforms such as Stack Overflow should prioritize adding code comments, with a particular emphasis on block comments. These types of comments were perceived by users as more helpful, indicating their potential to increase the overall helpfulness and understandability of code snippets. The guidelines provided by Stack Overflow on writing good answers[8] do not mention code comments at all (as of July 2025). Our suggestion is to extend these instructions based on our findings.

Our study has implications beyond code reading and comprehension. With the widespread adoption of generative AI tools in software development (Anh Nguyen-Duc 2024), the question arises of how to prompt these tools to generate the most helpful solutions

[8] https://stackoverflow.com/help/how-to-answer

matching users' needs. An interesting question is whether prompting generative AI tools to include block comments generally leads to the generation of more helpful code snippets than not prompting them to include these comments.

The study presented in this article lays the groundwork for different future research directions. We have investigated the impact of comment types and surface features on the perceived helpfulness of Stack Overflow posts. However, other features such as comment length, variable naming conventions, or other coding style elements, may also influence perceived helpfulness. Experiments similar to ours might help to investigate the respective features. Furthermore, replications of our study, as well as additional qualitative studies, can help refine our results and their interpretation.

On a meta-level, with the advance of generative AI in software engineering, a solid theoretical embedding of activities such as reading and comprehending code becomes even more important for software engineering research. Without solid theories (Lorey et al. 2022) of how users carry out central software engineering tasks, it will be difficult to compare the performance of AI-based tools and human developers and to develop suitable benchmarks.

9 Conclusion

The goal of this study was to investigate the factors that influence how Stack Overflow users assess the helpfulness of code snippets and solutions. To this end, we developed a simulated Stack Overflow environment that closely resembled the platform's layout and functionality. The primary focus was on the role of source code comments—specifically how different comment types (block, inline, or none) affect perceived helpfulness. In addition, we examined whether surface-level features on the platform—namely answer position and score—affect users' helpfulness assessment. We conducted an online experiment with 91 participants using this simulated environment. The results showed that code snippets with block comments were perceived as more helpful than those with inline comments, and both types were rated as more helpful than uncommented code. In contrast, answer order and scores had no significant effect on perceived helpfulness.

Our findings underscore the importance of code comments in influencing how developers judge the helpfulness of code snippets. These insights have practical implications for improving user guidance on platforms like Stack Overflow and, as motivated above, for enhancing the design of AI-based coding assistants that generate or recommend code snippets.

Directions for future research include conducting new experiments that incorporate additional features that may influence perceived helpfulness, as well as evaluating how prompting GenAI tools to include line or block comments impacts the perceived helpfulness of generated code snippets. Furthermore, replications and additional qualitative studies can help refine our results, working towards detailed theories of how users read, evaluate, and reuse code snippets.

Author contributions Kathrin Figl: Conceptualization, Data curation, Formal analysis, Investigation, Resources, Visualization, Writing (original draft), Writing (review & editing).
Maria Kirchner: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Resources, Visualization, Writing (original draft), Writing (review & editing).
Sebastian Baltes: Conceptualization, Writing (original draft), Writing (review & editing).
Michael Felderer: Conceptualization, Writing (review & editing).

Funding Open Access funding enabled and organized by Projekt DEAL. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Data availability The questionnaire used in our online experiment, along with the raw data analyzed using SPSS, is available as supplementary material (Figl et al. 2024). To protect participant anonymity, we modified any rows that could reveal their identities, including the removal of exact job descriptions and the generalization of participants' exact ages into age ranges.

Declarations

Ethical approval This study did not require formal approval from an ethics committee. The research was conducted in accordance with the principles of good scientific practice.

Informed consent All participants were informed about the purpose of the study and provided their consent prior to participation through the online questionnaire.

Consent to participate All participants gave informed consent before taking part in the study via the online experiment interface.

Conflicts of interest The authors declare that Sebastian Baltes is a member of the Empirical Software Engineering editorial board.

Clinical trial number Not applicable.

References

Aaltonen A, Wattal S (2020) Rejecting and Retaining New Contributors in Open Knowledge Collaboration: A Natural Experiment in Stack Overflow Q &A Service. In: 28th European Conference on Information Systems (ECIS 2020)

Abdalkareem R, Shihab E, Rilling J (2017) On code reuse from StackOverflow: an exploratory study on Android apps. Inf Softw Technol 88:148–158

Alarcon GM, Militello LG, Ryan P, Jessup SA, Calhoun CS, Lyons JB (2016) A descriptive model of computer code trustworthiness. J Cogn Eng Decis Mak 11(2):107–121

Anh Nguyen-Duc FK Pekka Abrahamsson (2024) Generative AI for Effective Software Development. Springer Cham

Bafatakis N, Boecker N, Boon W, Salazar MC, Krinke J, Oznacar G, White R (2019) Python Coding Style Compliance on Stack Overflow. In: Proceedings of the 19th International Conference on Mining Software Repositories (MSR), IEEE, pp 210–214

Baltadzhieva A, Chrupała G (2015) Predicting the Quality of Questions on StackOverflow. In: Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP 2015), pp 32–40

Baltes S, Diehl S (2019) Usage and attribution of Stack Overflow code snippets in GitHub projects. Empir Softw Eng 24(3):1259–1295

Baltes S, Dumani L, Treude C, Diehl S (2018) SOTorrent: Reconstructing and Analyzing the Evolution of Stack Overflow Posts. In: Proceedings of the 15th International Conference on Mining Software Repositories (MSR 2018), ACM, pp 319–330

Bornfeld B, Rafaeli S (2019) When Interaction is Valuable: Feedback, Churn and Survival on Community Question and Answer Sites: The Case of Stack Exchange. In: 52nd Hawaii International Conference on System Sciences (HICSS 2019)

Brooks R (1983) Towards a theory of the comprehension of computer programs. Int J Man-Mach Stud 18(6):543–554

Buse RP, Weimer WR (2009) Learning a metric for code readability. IEEE Trans Softw Eng 36(4):546–558

Buse RPL, Weimer W (2008) A metric for software readability. In: Ryder BG, Zeller A (eds) Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2008, Seattle, WA, USA, July 20-24, 2008, ACM, pp 121–130, https://doi.org/10.1145/1390630.1390647

Busjahn T, Bednarik R, Begel A, Crosby M, Paterson JH, Schulte C, Sharif B, Tamm S (2015) Eye movements in code reading: Relaxing the linear order. In: Proceedings of the 23rd International Conference on Program Comprehension (ICPC 2015), IEEE, pp 255–265

Chatterjee P, Kong M, Pollock L (2020) Finding help with programming errors: an exploratory study of novice software engineers' focus in Stack Overflow posts. J Syst Softw 159:110454

Chen T, Ouh EL, Tan KW, Lo SL (2023) Machine-Learning Approach to Automated Doubt Identification on Stack Overflow Comments to Guide Programming Learners. In: 27th Pacific Asia Conference on Information Systems (PACIS 2023)

Crosby ME, Stelovsky J (1990) How do we read algorithms? A case study. Computer 23(1):25–35

de Souza SCB, Anquetil N, de Oliveira KM (2005) A Study of the Documentation Essential to Software Maintenance. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information (SIGDOC 2005), pp 68–75

Duijn M, Kucera A, Bacchelli A (2015) Quality Questions Need Quality Code: Classifying Code Fragments on Stack Overflow. In: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR 2015), IEEE, pp 410–413

Dunsmore A, Roper M, Wood M (2000) The role of comprehension in software inspection. J Syst Softw 52(2–3):121–129

Etgar A, Friedman R, Haiman S, Perez D, Feitelson DG (2022) The effect of information content and length on name recollection. In: Rastogi A, Tufano R, Bavota G, Arnaoudova V, Haiduc S (eds) Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022, ACM, pp 141–151, https://doi.org/10.1145/3524610.3529159

Fagerholm F, Felderer M, Fucci D, Unterkalmsteiner M, Marculescu B, Martini M, Tengberg LGW, Feldt R, Lehtelä B, Nagyváradi B et al (2022) Cognition in software engineering: a taxonomy and survey of a half-century of research. ACM Comput Surv 54(11s):1–36

Fakhoury S, Roy D, Hassan SA, Arnaoudova V (2019) Improving source code readability: theory and practice. In: Guéhéneuc Y, Khomh F, Sarro F (eds) Proceedings of the 27th International Conference on Program Comprehension, ICPC 2019, Montreal, QC, Canada, May 25-31, 2019, IEEE / ACM, pp 2–12, https://doi.org/10.1109/ICPC.2019.00014

Figl K, Maria K, Baltes S, Felderer M (2024). The Influence of Code Comments on the Perceived Helpfulness of Stack Overflow Posts (Supplementary Material). https://doi.org/10.5281/zenodo.13319936

Galesic M, Tourangeau R, Couper MP, Conrad FG (2008) Eye-tracking data: new insights on response order effects and other cognitive shortcuts in survey responding. Public Opin Q 72(5):892–913

Gantayat N, Dhoolia P, Padhye R, Mani S, Sinha VS (2015) The Synergy Between Voting and Acceptance of Answers on StackOverflow – Or the Lack Thereof. In: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR 2015), IEEE, pp 406–409

Gilmore DJ (1990) Expert Programming Knowledge: A Strategic Approach, Elsevier, pp 223–234

Gopstein D, Zhou HH, Frankl PG, Cappos J (2018) Prevalence of confusing code in software projects: atoms of confusion in the wild. In: Zaidman A, Kamei Y, Hill E (eds) Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, ACM, pp 281–291, https://doi.org/10.1145/3196398.3196432

Huang Y, Guo H, Ding X, Shu J, Chen X, Luo X, Zheng Z, Zhou X (2023) A comparative study on method comment and inline comment. ACM Trans Softw Eng Methodol 32(5):126:1–126:26, https://doi.org/10.1145/3582570

Jabrayilzade E, Gürkan O, Tüzün E (2021) Towards a taxonomy of inline code comment smells. In: 21st IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2021, Luxembourg, September 27-28, 2021, IEEE, pp 131–135, https://doi.org/10.1109/SCAM52516.2021.00024

Langhout C, Aniche M (2021) Atoms of confusion in java. In: 29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021, IEEE, pp 25–35, https://doi.org/10.1109/ICPC52881.2021.00012

Lee SY, Rui H, Whinston AB (2019) Is best answer really the best answer? The politeness bias. MIS Q 43(2):579–600

Lee T, Lee JB, In HP (2013) A study of different coding styles affecting code readability. Int J Softw Eng Appl 7(5):413–422

Littman DC, Pinto J, Letovsky S, Soloway E (1987) Mental models and software maintenance. J Syst Softw 7(4):341–355

Lorey T, Ralph P, Felderer M (2022) Social science theories in software engineering research. In: Proceedings of the 44th International Conference on Software Engineering, pp 1994–2005

Meng N, Nagy S, Yao D, Zhuang W, Argoty GA (2018) Secure Coding Practices in Java: Challenges and Vulnerabilities. In: Proceedings of the 40th International Conference on Software Engineering (ICSE 2018), pp 372–383

Mi Q, Hao Y, Ou L, Ma W (2022) Towards using visual, semantic and structural features to improve code readability classification. J Syst Softw 193:111454. https://doi.org/10.1016/j.jss.2022.111454

Misra V, Reddy JSK, Chimalakonda S (2020) Is there a correlation between code comments and issues? An exploratory study. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC 2020), pp 110–117

Murphy J, Hofacker CF, Mizerski R (2006) Primacy and recency effects on clicking behavior. J Comput-Mediat Commun 11(2):522–535

Nasehi SM, Sillito J, Maurer F, Burns C (2012) What makes a good code example?: A study of programming Q &A in StackOverflow. In: IEEE International Conference on Software Maintenance (ICSM 2012), IEEE, pp 25–34

Nasr M, Carlini N, Hayase J, Jagielski M, Cooper AF, Ippolito D, Choquette-Choo CA, Wallace E, Tramèr F, Lee K (2023) Scalable extraction of training data from (production) language models. 2311.17035

Pascarella L, Bacchelli A (2017) Classifying code comments in Java open-source software systems. In: Proceedings of the 14th International Conference on Mining Software Repositories (MSR 2017), IEEE, pp 227–237

Posnett D, Hindle A, Devanbu PT (2011) A simpler model of software readability. In: van Deursen A, Xie T, Zimmermann T (eds) Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings, ACM, pp 73–82, https://doi.org/10.1145/1985441.1985454

Rani P, Blasi A, Stulova N, Panichella S, Gorla A, Nierstrasz O (2023) A decade of code comment quality assessment: a systematic literature review. J Syst Softw 195:111515

Robillard MP (2009) What makes APIs hard to learn? Answers from developers. IEEE Softw 26(6):27–34

Scalabrino S, Linares-Vásquez M, Oliveto R, Poshyvanyk D (2018) A comprehensive model for code readability. J Softw Evol Process. https://doi.org/10.1002/smr.1958

Sergeyuk A, Lvova O, Titov S, Serova A, Bagirov F, Kirillova E, Bryksin T (2024) Reassessing java code readability models with a human-centered approach. In: Steinmacher I, Linares-Vásquez M, Moran KP, Baysal O (eds) Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ICPC 2024, Lisbon, Portugal, April 15-16, 2024, ACM, pp 225–235, https://doi.org/10.1145/3643916.3644435

Sha AS, Haller A, Shi Y (2022) Effects of Label Usage on Question Lifecycle in Q &A Community. In: European Conference on Information Systems (ECIS 2022)

Shneiderman B (1977) Measuring computer program quality and comprehension. Int J Man Mach Stud 9(4):465–478

Stack Exchange Inc (2025) Stack Exchange Data Explorer

Stapleton S, Gambhir Y, LeClair A, Eberhart Z, Weimer W, Leach K, Huang Y (2020) A human study of comprehension and code summarization. In: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020, ACM, pp 2–13, https://doi.org/10.1145/3387904.3389258

Storey MD, Zagalsky A, Filho FMF, Singer L, Germán DM (2017) How social and communication channels shape and challenge a participatory culture in software development. IEEE Trans Software Eng 43(2):185–204

Tempero ED, Denny P, Finnie-Ansley J, Luxton-Reilly A, Kirk D, Leinonen J, Shakil A, Sheehan RJ, Tizard J, Tu Y, Wuensche B (2024) On the comprehensibility of functional decomposition: An empirical study. In: Steinmacher I, Linares-Vásquez M, Moran KP, Baysal O (eds) Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension, ICPC 2024, Lisbon, Portugal, April 15-16, 2024, ACM, pp 214–224, https://doi.org/10.1145/3643916.3644432

Treude C, Robillard MP (2017) Understanding Stack Overflow Code Fragments. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME 2017), IEEE, pp 509–513

Treude C, Barzilay O, Storey MD (2011) How do programmers ask and answer questions on the web? In: Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011), ACM, pp 804–807

Van Der Linden D, Williams E, Hallett J, Rashid A (2020) The impact of surface features on choice of (in) secure answers by Stackoverflow readers. IEEE Trans Software Eng 48(2):364–376

Van Rossum G, Warsaw B, Coghlan N (2001) PEP 8 – Style Guide for Python Code. https://peps.python.org/pep-0008/

Vasilescu B, Filkov V, Serebrenik A (2013) Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: 2013 International Conference on Social Computing (SocialCom 2013), IEEE Computer Society, pp 188–195

Von Mayrhauser A, Vans AM (1995) Program comprehension during software maintenance and evolution. Computer 28(8):44–55

Wang H, Gao Z, Bi T, Grundy JC, Wang X, Wu M, Yang X (2024) What makes a good TODO comment? ACM Trans Softw Eng Methodol 33(6):165. https://doi.org/10.1145/3664811

Wiese ES, Rafferty AN, Fox A (2019) Linking code readability, structure, and comprehension among novices: it's complicated. In: Beecham S, Damian DE (eds) Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2019, Montreal, QC, Canada, May 25-31, 2019, IEEE / ACM, pp 84–94, https://doi.org/10.1109/ICSE-SEET.2019.00017

Wu Y, Wang S, Bezemer C, Inoue K (2019) How do developers utilize source code from Stack Overflow? Empir Softw Eng 24(2):637–673

Xue J, Wang L, Zheng J, Li Y, Tan Y (2023) Can ChatGPT Kill User-Generated Q &A Platforms? In: Proceedings of the 44th International Conference on Information Systems (ICIS 2023)

Zhang H, Wang S, Chen TH, Hassan AE (2019) Reading answers on Stack Overflow: not enough! IEEE Trans Software Eng 47(11):2520–2533

Authors and Affiliations

Kathrin Figl[1] · Maria Kirchner[1] · Sebastian Baltes[2] · Michael Felderer[3,4]

✉ Sebastian Baltes
sebastian.baltes@uni-bayreuth.de

Kathrin Figl
kathrin.figl@uibk.ac.at

Maria Kirchner
kirchnermaria7@gmail.com

Michael Felderer
michael.felderer@dlr.de

[1] University of Innsbruck, Innsbruck, Austria

[2] University of Bayreuth, Bayreuth, Germany

[3] German Aerospace Center (DLR), Cologne, Germany

[4] University of Cologne, Cologne, Germany