Dear Diary, so far so good—The application works perfectly

Int. J. Human–Computer Studies 218 (2027) 103943

Contents lists available at ScienceDirect

International Journal of Human - Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

Dear Diary, so far so good—The application works perfectly

Juan Pablo Sáenz*, Luigi De Russis

Department of Control and Computer Engineering, Politecnico di Torino, Torino, 10129, Italy

HIGHLIGHTS

- We designed and implemented Dear Diary, an IDE-integrated tool for developers.
- It captures code and non-code artifacts to support a holistic development journey.
- A lab study with 18 participants confirmed its usability, relevance, and usefulness.

ARTICLE INFO

Keywords:
Development environment
Development framework
Non-experienced programmers

ABSTRACT

Approaching a development framework for the first time is like embarking on a journey, marked by mistakes, solutions, design choices, and key considerations. Tracking these aspects, much like keeping a diary, helps programmers understand implementations, revisit errors, and plan future strategies. Inspired by this analogy, we present Dear Diary, a tool integrated into the IDE that enables programmers to document their development journey. The tool allows developers to capture multiple versions of a project, functional or not, and to add rich explanations in their own words. These explanations can refer to all the components of a project, encompassing files, code, terminal commands, and dependencies, to create a holistic approach that provides richer context for understanding and learning from project evolution. A user study with 18 participants examined how Dear Diary was used during a framework-learning task, focusing on the capture phase of the development journey. Results showed that participants used the tool throughout the programming task to capture diverse forms of contextual knowledge linked to their development activity, while reporting positive perceptions of its usability, relevance, and potential usefulness.

1. Introduction and background

Learning a programming language or becoming proficient with a new development framework is far from being a linear and straightforward process that exclusively concerns code writing. It is a process with ups and downs, during which many things are learned along the way through trial and error. Contrary to the common perception, writing code represents only one of the many activities the developer must carry out. In their development journey, programmers encounter various tasks, including setting up the development and deployment environments (typically done by executing terminal commands), continuously seeking guidance from online tutorials, forums, or source code examples (Ichinco and Kelleher, 2015; Hartmann et al., 2010; Guo, 2017), making numerous design and implementation decisions (Liu et al., 2019), trying out and selecting the most suitable libraries, and interpreting and resolving novel execution errors, often arising from missing dependencies or incompatibilities. Unfortunately, the valuable knowledge that emerges from such tasks—regarding lessons learned, design rationale, strategies to overcome errors, sources consulted, and pieces of advice to remember or share—is rarely captured and tracked over time, despite being fundamental to understanding the current codebase and guiding future implementations (Maalej et al., 2014; Liang et al., 2023; Nassif and Robillard, 2017; Liu et al., 2021).

With respect to code writing, the most suitable and widely-adopted tools programmers have for tracking project development over time are Version Control Systems (VCSs). Using them, developers can capture snapshots of the codebase at specific points over time (by making commits) and associate a short message with each commit, ideally explaining what was changed and why the change was made. These commit messages serve as an “audit trail” through which programmers can get an idea of the codebase evolution and, to some extent, infer the rationale behind the changes (Tian et al., 2022). However, given their limited length

* Corresponding author.
Email addresses: juan.saenz@polito.it (J.P. Sáenz), luigi.derussis@polito.it (L. De Russis).

https://doi.org/10.1016/j.ijhcs.2026.103943
Received 14 November 2025; Received in revised form 2 September 2026; Accepted 5 September 2026
Available online 12 September 2026

and format (preferably written in the imperative, along with an optional explanation) (Li and Ahmed, 2023), they do not allow programmers to capture freely, in their own words, valuable implicit knowledge that contributed to the current state of the application, such as lessons learned, design rationale, strategies to overcome errors, sources consulted, and advice to remember or share. Furthermore, since each commit message is associated with the entire project, programmers cannot add detailed descriptions or annotations linked to specific portions of the codebase, such as a code snippet or a file (Wang et al., 2024a).

Importantly, as noted earlier, writing code is not the only activity a developer must carry out in the development journey with a new framework. While VCSs can track changes to the codebase, they are insufficient to capture other tasks and the valuable knowledge emerging from them, such as setting up the development and deployment environment, managing dependencies, and executing the application—typically done through terminal commands—which remains uncaptured (Zhi et al., 2015; Aghajani et al., 2019). Consequently, if a programmer discovers that running a specific terminal command enables them to implement a functionality, install a dependency, or resolve an issue, a commit message or code comments would likely be insufficient to capture this knowledge effectively. This would prevent the programmer from capturing crucial details that could guide them, or someone else, facing similar challenges in the future. For instance: Where did they find the solution (if online, on which specific website)? Which parameters should be included when executing the terminal command? What do these parameters represent? What should be the expected outcome, and how can it be verified?

In this scenario, using external tools such as wikis or platforms like Notion[1] or even plain text files saved locally for note-taking is a common practice (Al Safwan et al., 2022; Raglianti, 2022; Tantisuwankul et al., 2019; Storey et al., 2017). Regarding code writing, when used together with VCSs, this approach can enhance the expressivity and detail of commit messages by linking them to these external resources or by including them in reports or pull requests (Li and Ahmed, 2023). Similarly, this practice offers a feasible way to capture valuable knowledge not necessarily tied to code writing (*non-code-centric artifacts*), such as executed terminal commands, their corresponding outputs, and error messages. Nevertheless, maintaining such documentation requires additional effort, and the quality can suffer from outdated links and erroneous updates, resulting in irrelevant or decontextualized content (Dagenais and Robillard, 2010; Le et al., 2015). It also requires programmers to frequently switch between different applications and layouts, which increases interruptions and distractions (Parnin and Rugaber, 2011; Parnin and DeLine, 2010; Happel and Maalej, 2009). Additionally, programmers are required to manually define ad-hoc conventions and guidelines for organizing information, which becomes particularly challenging when considering the various versions of the development project over time. Moreover, this approach may not be practical for searches or for ensuring clarity and usability for future consultation or sharing with other programmers. In practice, if existing documentation cannot be found, it conceptually does not exist (Aghajani et al., 2019).

Furthermore, keeping track of errors and adding self-explanations to their corresponding solutions is crucial, as it enables developers to retain and replicate successful strategies across different implementations (Vieira et al., 2017; Margulieux et al., 2021). This practice not only helps programmers remember individual solution approaches but also supports the sharing and adoption of effective strategies within a broader community. By contrast, within VCS best practices, non-working versions are typically discouraged—for instance, through the recommendation to keep work-in-progress commits in separate branches to avoid affecting the main branch—making it difficult to capture the valuable insights that can emerge from errors and their resolution. While this guideline makes perfect sense in the context of professional software development to prevent the deployment of flawed code, it can be a hindrance for programmers who are becoming proficient with a new development framework. A similar issue may arise with non-code-centric artifacts, as capturing erroneous executions can be difficult in the absence of an automated mechanism.

To enable programmers who are becoming proficient with a new development framework to holistically track their development journey, we propose Dear Diary. This tool consists of a Visual Studio Code extension that seamlessly interacts with the IDE, allowing programmers to explain in their own words (much like in a diary), and in relation to the various components of a development project (both code-centric and non-code-centric artifacts), how they achieved working versions and overcame development and deployment errors. Specifically, Dear Diary enables developers to capture multiple project versions, called 'snapshots'. Each snapshot represents a project version, working or not, and can include rich explanations provided by the developers and associated with its components, namely code snippets, files, terminal commands, and dependencies. This holistic approach is intended to provide a richer context for understanding and learning from the evolution of a project, supported by the following features:

- **Create snapshots** to capture the current project state. In the context of Dear Diary, each snapshot includes the files as they existed at the time of the snapshot, the terminal commands executed since the previous snapshot up to the creation of the new one, and the development and deployment dependencies present in the project at that point.
- **Integrate detailed explanations linked to project files**, granting users the freedom to annotate or elucidate specific files in their own words.
- **Automatically gather and display all terminal command activity**, including the commands executed and their corresponding outputs. Allow programmers to add a description for each command, indicate its success or failure, and include tags.
- **Collect and display the development and deployment dependencies** present in the project. Enable programmers to annotate, tag, and access official documentation for each dependency.
- **Add detailed explanations tied explicitly to relevant code snippets**, outlining the reasoning behind the implementation, links to the consulted sources, and observations to remember.
- **Navigate through the snapshots**, displaying their corresponding files, terminal commands, and dependencies. Allow programmers to freely explore and visualize associated information, including descriptions, tags, or annotations made by them.
- **Provide a consolidated timeline-style visualization of each snapshot**, enabling the programmers to **conduct integrated searches** across code snippets, files, terminal commands, and dependencies, leveraging the annotations and tags they have added.

Finally, in this work, we present the following main contributions:

- Dear Diary, is an IDE-integrated tool that allows programmers to straightforwardly capture the knowledge that emerges from working on various development tasks and associate it with specific project elements, extending across multiple snapshots of the development project, encompassing both working and non-working versions (Section 3).
- A **user study** with 18 participants examining how Dear Diary was used during a framework-learning task (Section 5). The study focused on the capture phase of the development journey and provided initial evidence that:
  - Participants were able to integrate Dear Diary into the programming task and use it to create situated records linked to snapshots, files, terminal commands, dependencies, and code snippets;

[1] "Your connected workspace for wiki, docs & projects | Notion", https://www.notion.so, last accessed November 11, 2025.

– The recorded content covered diverse forms of contextual knowledge, including actions taken, exploratory questions, consulted resources, instructions, reminders, encountered issues, errors, and TODOs;
– Participants reported positive perceptions of Dear Diary's usability and relevance, and identified academic and collaborative scenarios in which the tool could be useful.

2. Related work

Our research builds upon prior work that seeks to **capture design rationale** throughout the development process—particularly when getting started with a framework—by using **annotations** and adopting a **storytelling** approach.

2.1. Capturing the design rationale

Understanding the design rationale behind existing implementations is one of the most common challenges developers face (LaToza, 2020). Good documentation has long been argued to be vital in helping developers write code more quickly and consistently with design decisions (Mehrpour et al., 2019), and documenting design decisions, in particular, is considered essential to maintaining the comprehensibility of code (Clements et al., 2003). However, traditional documentation practices are inefficient. They require much manual work during their creation, and there is a gap between the creators and consumers (Robillard et al., 2017; Nasehi et al., 2012). Indeed, producing such documentation is time-consuming, and its cost can outweigh its benefits (Lethbridge et al., 2003). As a result, many design decisions remain uncaptured, undocumented, and tacit in the minds of decision-makers (Jansen et al., 2008; van der Ven et al., 2006), leading to knowledge evaporation (Capilla et al., 2016).

While various studies aim to facilitate the understanding and integration of online code examples (Oney and Brandt, 2012; Head et al., 2018a; Liu et al., 2019), and to automatically generate documentation based on the available code (Aghajani et al., 2021; Zhang et al., 2020; McBurney and McMillan, 2016; LeClair et al., 2019; Rai et al., 2022; Wu et al., 2021), fewer research efforts have focused on helping programmers track their development process in their own words, extending beyond a purely code-focused perspective. Our approach aims to **streamline the process of capturing design rationale** by integrating directly into the IDE, minimizing the need for developers to switch between applications. It encourages developers to document their rationale 'on the go' as they progress through their development journey and supports the addition of high-level descriptions that may not be directly tied to technical aspects.

2.2. Tools for documenting and managing the design rationale

In general terms, our proposal falls within the realm of tools designed to support developers in managing design decisions. In this context, Mehrpour et al. (Mehrpour and Latoza, 2023) surveyed tool support for developers managing design decisions in code, focusing on activities such as identifying, choosing, documenting, and understanding these decisions. They introduced a taxonomy of tools that assist developers in this process. Dear Diary fits into this taxonomy's 'documentation tool' category, as it captures design rationales by explaining choices in design decision descriptions—often in a less formal, unstructured format. While its primary purpose is to document design decisions, tools like ours also support updating them over time and linking them to the code. This aligns with the need highlighted in Alexeeva et al. (Alexeeva et al., 2016) systematic literature review, which calls for better support in working with design rationale—making it more explicit, better connected to the code, and linked to high-level and contextual information (Head et al., 2018b).

Various tools and methods have been developed within this taxonomy category to actively document design decisions and capture the rationale behind development choices. ActiveDocumentation (Mehrpour et al., 2019) documents design decisions as design rules, which are checked against the code. After making code changes, developers receive immediate feedback on which rules are satisfied or violated. Along the same lines, CODES is a tool that extracts candidate method documentation from Stack Overflow discussions and generates Javadoc descriptions from it (Vassallo et al., 2014). Furthermore, Tutorons (Head et al., 2015) are language-specific routines that automatically provide context-relevant, on-demand micro-explanations and demonstrations of code snippets on web pages, helping programmers understand the code without consulting external documentation. Colaroid (Wang et al., 2023) is a literate programming approach for creating explorable multi-stage tutorials. It uses a Visual Studio Code extension that lets web developers write tutorials while building a front-end HTML application, interleaving textual explanations with executable code in an $\langle iframe \rangle$.

Beyond tools explicitly designed to capture design rationale, another relevant family of environments is computational notebooks, most prominently Jupyter Notebooks. Notebooks allow users to interleave executable code, narrative explanations, visualizations, and outputs in a single document, and they have been widely used for exploratory programming, data analysis, teaching, and computational narratives (Rule et al., 2018; Kery et al., 2018). Prior work has examined their benefits for sensemaking and communication (Rule et al., 2018; Kery et al., 2018), as well as challenges related to maintenance, messiness, reproducibility, and collaboration (Head et al., 2019; Pimentel et al., 2019). Dear Diary shares with notebooks the goal of connecting implementation artifacts with explanatory narrative. However, its design targets a different setting: IDE-based software projects composed of multiple files, terminal commands, dependencies, and evolving project states. Rather than organizing work as a linear sequence of executable cells, Dear Diary captures snapshots of a project and allows programmers to annotate heterogeneous artifacts, including non-working states and terminal outputs, as part of a broader development journey.

Similarly, DecDoc (Hesse et al., 2016) lets developers capture and collaborate on decision-related knowledge, linking design decisions to artifacts like requirements, design diagrams, and code. It documents decisions through UML diagrams and code annotations. REACT (Alkadhi et al., 2017) is a lightweight approach for manually annotating chat messages to capture the rationale behind team development decisions. It focuses on annotating chat messages with five rationale elements: issues, alternatives, pro-arguments, con-arguments, and decisions. Finally, CodeLink (Zaychik and Regli, 2003) is a design rationale support tool that integrates email-based collaboration with the software development process. It allows development teams to associate specific code elements with email messages automatically.

In this regard, Dear Diary aims to **go beyond capturing design rationale solely tied to code**. By integrating with the IDE to automatically gather contextual information, our approach enables the documentation of rationale related to files, terminal commands, and development or execution dependencies. This aspect is crucial when working not with standalone code files but with frameworks where setup and execution are more complex, and design decisions often span across heterogeneous artifacts—such as the choice of a specific dependency, the execution of a deployment command with particular parameters, or the creation of multiple JSON-format setup files for a given purpose. By capturing this broader context and linking informal descriptions to these artifacts, the tool supports reproducibility and helps less experienced programmers understand how implementations evolve and artifacts are interconnected.

2.3. Annotations and storytelling in software development

Moreover, from a more detailed perspective, our proposal concerns annotations and storytelling. In software development, annotations aid developers in externalizing and tracking information during code sensemaking tasks (Horvath et al., 2022a). They help developers recall and locate semantic information during software maintenance (Storey et al.,

2009). Additionally, task annotations within source code are vital for managing personal and team tasks, serving as a bridge between formal issue tracking systems and informal code comments (Storey et al., 2008). Indeed, research has shown that annotations on program documentation and code can significantly enhance programmers' understanding and productivity (Sulír et al., 2016; Horvath et al., 2022b).

Storytelling, for its part, has been recognized as a creative process with similarities to software development (Ciancarini et al., 2023), and it has been investigated as a strategy to assist developers in code comprehension (Wuilmart et al., 2023). Allen and Kelleher (Allen and Kelleher, 2024), for instance, explored the use of AI-generated narratives to present software development histories. They used GPT-4 to generate a narrative story view from code changes and web searches. In a study with 16 programmers, these narratives improved recall, boosted confidence, and reduced mental effort. Participants appreciated how the stories connected changes and fostered empathy for the original developer, viewing them as a complement to traditional versioning systems. Compared to this work, Dear Diary's approach is not to automate storytelling but to **empower programmers who are engaging with a particular framework for the first time to track their development journey on their own terms, in a self-reflective manner**, in relation to the diverse artifacts they deem relevant as they explore, manipulate, and make sense of them.

Most relevant to our work are Storyteller (Mahoney, 2023), Sodalite (Horvath et al., 2023), and Catseye (Horvath et al., 2022a). Storyteller is a tool that helps instructors guide learners through code examples by animating and annotating coding sessions. It captures file changes via a Visual Studio Code extension. These 'playbacks' can be replayed on a web interface, where instructors annotate the code's evolution and explain key decisions with text, highlights, media, and self-grading questions. For its part, Sodalite is a Visual Studio Code extension that helps developers create code stories. It leverages the IDE context to structure documentation and link the story text to relevant code. Specifically, starting with a code story template, the user manually connects the text to code definitions, references, or snippets, and the tool suggests additional connections based on frequently edited identifiers from the Git commit history. Finally, Catseye is a Visual Studio Code extension that helps experienced developers understand complex, large-scale codebases. It lets users add annotations such as tasks, questions, hypotheses, and answers directly to code snippets without modifying the code. Catseye also supports threaded discussions, links notes to different code segments, and maintains consistent syntax across languages.

In addition to the distinctive characteristics outlined so far, Dear Diary integrates annotations and storytelling into **tracking and reviewing erroneous executions** and their resolution strategies. Programmers can capture faulty versions, annotate terminal errors and outputs, and explain how they address the issues. While existing tools focus on documenting working code and its rationale, Dear Diary emphasizes the value of reflecting on mistakes—helping novices retain and reuse their problem-solving strategies (Vieira et al., 2017; Margulieux et al., 2021).

3. Usage scenario

To illustrate how Dear Diary looks and works, we present a usage scenario framed in the context of a developer creating her own first web project using the React library, and we indicate how the various steps are completed using Figs. 1 and 2 (a video demonstrating the tool and its features is also provided as Supplementary Material). In a real-world scenario, these steps unfold sequentially, with graphical elements appearing as the programmer progresses through each action. In both figures, various graphical components are presented simultaneously for illustrative purposes, although they may not originate from the same snapshot. However, it is important to note that since the implementation of the Dear Diary extension adhered meticulously to all official Visual Studio Code user experience guidelines, users can display these components in any layout provided by the IDE.

(a) Isabel is an inexperienced web developer implementing her first React application. She uses Visual Studio Code, her preferred IDE, while following a React "getting started" guide that she found

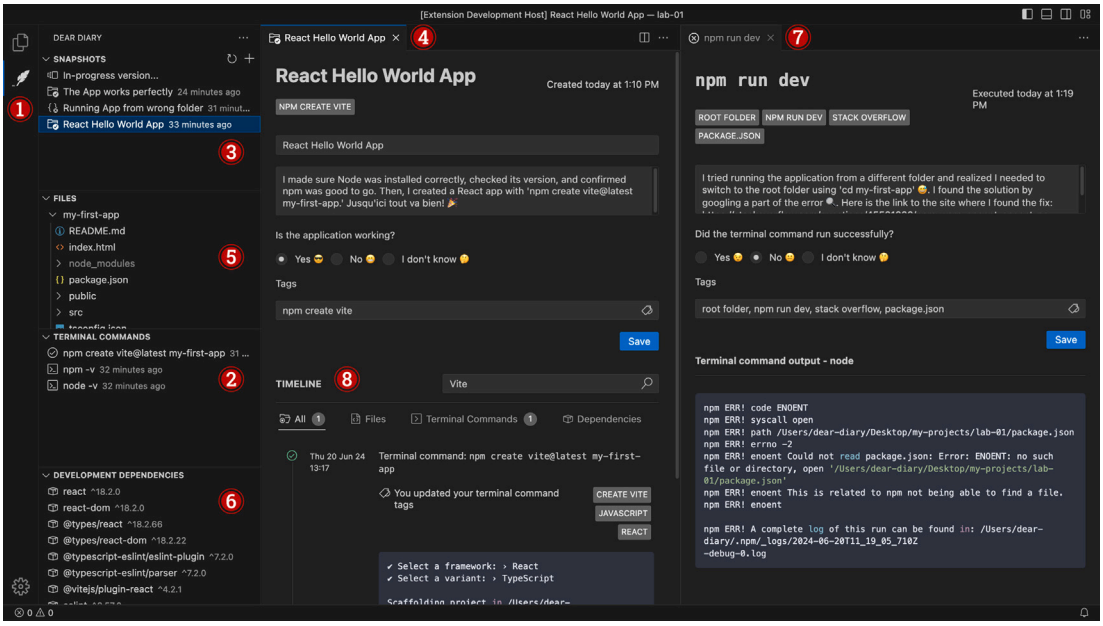

Fig. 1. Dear Diary is a Visual Studio Code extension designed to assist inexperienced programmers in their development process. It enables users to (a) capture the current project state with snapshots, encompassing code snippets, files, terminal commands, and dependencies, whether it represents a working version or not; (b) add self-explanatory comments linked to the snapshots, specific files and code snippets within them, terminal commands and their corresponding outputs, and development and deployment dependencies; (c) navigate seamlessly through snapshots, files, and terminal commands; and (d) conduct cross-cutting searches across all snapshot resources (code snippets, files, terminal commands, and dependencies) to unveil the reasoning behind implementation decisions or strategies addressing potential errors or issues.

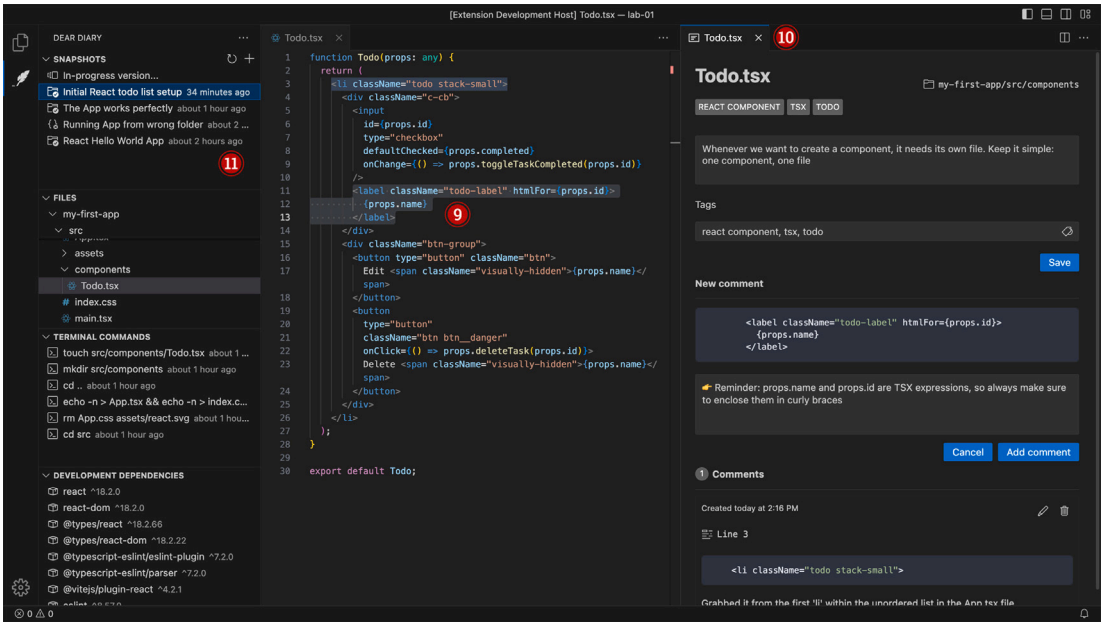

Fig. 2. As Isabel continues with her implementation, she documents the commands she executes (indicating whether they work or not), the dependencies she uses, and furthermore, as shown in the screenshot, Dear Diary allows her to add detailed explanations to code snippets, in which she explains why she implemented certain features, shares links to the sources she consulted, and includes important reminders.

online. Namely, she followed the React Getting Started tutorial in the MDN Web Docs.[2]

(b) As in many of the widely adopted available web frameworks, in React, the first steps correspond to installing and setting up the execution environment and its dependencies (Node and `npm`, in this case). These installation and setup steps are commonly completed by downloading and running installers and executing various terminal commands with diverse parameters.

(c) Naturally, these terminal commands are new to Isabel. After executing them to the letter as indicated in the getting started guide, she manages to create her first React application. At that point, she notices that she has achieved a stable checkpoint, notwithstanding its simplicity. Therefore, she wants to capture and document it. To do so, she opens the Dear Diary extension view directly from Visual Studio Code by clicking on the pen icon in the Activity Bar (**1**).

(d) Within the Dear Diary interface, Isabel creates a new snapshot by clicking the '+' button in the top right corner of the 'Snapshots' view (**3**). This action brings up a panel in the Visual Studio Code editor area (**4**). From there, she can set the snapshot's title, add a description, indicate the current status of the application (whether it is working or not), and add tags. Additionally, if she is unsure or does not know whether the application works, she can choose the 'I don't know' option.

Isabel inserts "React Hello World Application" as the title and provides the description "I made sure Node was installed correctly, checked its version, and confirmed npm was good to go. Then, I created a React app with '`npm create vite@latest my-first-app`.' Jusqu'ici tout va bien!" Additionally, she indicates that the application is working and adds the tag "npm create vite".

(e) Isabel notices the successful creation of the snapshot as its title ("React Hello World Application") appears in the 'Snapshots' view list (**3**). Furthermore, she observes that although the graphical interface of Dear Diary was not visible while she was executing the terminal commands to create the React project, the extension automatically captured them, and they are now part of the snapshot she just created. As depicted in (**2**), the 'Terminal commands' view lists the commands she has executed thus far. In the same way, the 'Files' view (**5**) displays the files and folders created during the project setup, and the 'Development dependencies' view (**6**) shows the project's dependencies.

Wrapping up, a snapshot in Dear Diary comprises:

- A copy of all the project folders and files as they were at the time of the snapshot creation (**5**).
- A list of all terminal commands executed since the last snapshot, if there is a previous snapshot, or all the terminal commands executed so far if it is the first one (**2**).
- The development dependencies present in the project at the time of snapshot creation (**6**).

(f) Later on, Isabel tries to execute her application by executing the terminal command '`npm run dev`'. However, she gets an error that she is not familiar with. Therefore, Isabel searches online and finds the cause of this error: the application had to be executed from the project's root directory. Although this is a trivial issue, Isabel has overcome her first error and wants to remember its cause, solution, and the source she consulted in case it happens again. Then, Isabel opens the corresponding terminal command by clicking it in the 'Terminal commands' view (**2**), which has been updated to show the commands executed after the first snapshot and now includes those for the 'In-progress version…', which will be linked to the new snapshot once created.

In the corresponding terminal command panel that appears in the editor area (**7**), she adds an explanation written in her own words with the URL of the Stack Overflow post that contains the solution to her problem and specifies that the terminal command did not run successfully. Additionally, in the lower part of the pane, she sees the output of the command, automatically captured by Dear Diary for this and all other executed commands.

[2] "Getting started with React - Learn web development | MDN", https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started, last accessed November 11, 2025.

(g) Moreover, throughout the entire implementation process of her React application, she wants to document the rationale and considerations guiding her implementation decisions and how they are linked to the resultant code. To achieve this, she adds several detailed explanations, tied explicitly to relevant code fragments, outlining the reasoning behind the implementation, the links to the consulted sources, and observations to remember. To do so, as shown in Fig. 2, from within Dear Diary, Isabel opens the file containing the code she wants to comment, selects the code snippet, right-clicks on it, and chooses the 'Write a comment for this code' option from the context menu (**9**). Upon doing so, a form appears in the File panel on the right side of the screen, allowing her to articulate her thoughts in her own words regarding that specific code fragment. Similarly, as with the snapshots, the terminal commands, and the development dependencies, Isabel can provide comments for the entire file and attach relevant tags to it (**10**).

(h) From then on, several snapshots can be created by Isabel, each with their associated files, terminal commands, and development and execution dependencies (**11**).

(i) Naturally, as the development process goes on, Isabel encounters familiar error messages that she has successfully resolved in the past, or she needs to write similar code for new components. Seeking solutions, she aims to explore past snapshots without the necessity of navigating to each one individually, opening the files, the terminal commands, or the development dependencies, and reading all their associated descriptions, tags, or code fragments-related comments.

To that end, rather than navigating through each snapshot individually, she opens them. For each snapshot, she accesses a timeline-style visualization (**8**) containing all commented files—whether annotated with general descriptions, tags, or commented code fragments; all the terminal commands executed since the last snapshot; and the development dependencies with added descriptions or tags. Most importantly, Dear Diary offers a search feature through which Isabel can easily filter all snapshot contents based on various criteria, such as descriptions, tags, comments on code fragments, file names, or terminal commands. In (**8**), for instance, it can be observed that upon entering the search term 'Vite' into the search field, the timeline items were filtered to display only one terminal command containing the specified word, whether within a tag or in its description.

This timeline visualization, in conjunction with the search feature, enables Isabel to quickly identify files, terminal commands, and dependencies that, according to her reminders, explanations, or tags, are somehow linked or share common characteristics. This capability facilitates comprehensive and transversal exploration of her project and, in a broader sense, her development process.

4. Design considerations

We now revisit Dear Diary's features, focusing on design motivations and implementation details. Specifically, our design and implementation aimed to achieve the following goals:

Tracking the development journey 'on the go' directly from the IDE: By integrating Dear Diary directly into the IDE, we aimed to address the well-known challenge of incentivizing developers to consistently track their design decisions (Le et al., 2015; Robillard et al., 2017) by focusing on two key aspects. First, we sought to minimize interruptions caused by switching between different applications or windows to document the development process. Currently, developers must manually copy and paste data from the IDE into other applications to externalize their knowledge, which is a significant obstacle to tracking their development process. Interruptions, even during brief tasks, are known to require developers to spend time and energy regaining focus (Meyer et al., 2022; Parnin and DeLine, 2010). This goal involves integrating the functionalities of our tool as organically as possible by aligning with the IDE's visual identity and adapting to its most common interactions.

Secondly, we aimed to accurately associate programmers' annotations with the artifacts relevant to the development process to create a rich context that could effectively enhance their understanding (Ko et al., 2006). For this context to truly represent the typical complexity of development projects, it must extend beyond source code. This requires including various elements that can only be gathered directly from the IDE, such as files that are not necessarily source code, dependencies that define the development and execution environment, terminal commands with their parameters and outputs, and the relationships among all these elements.

Automatically gathering context on development and execution environments: In line with the previous consideration and the need to go beyond a code-centric approach and to lower barriers that discourage developers from documenting their work, it is essential to automatically capture information related to the development and execution environment, as well as terminal command executions, without requiring manual actions from the programmer. Moreover, this information should be readily available for annotation and inclusion in the traceability of the development process. This approach enhances project reproducibility and provides non-experienced programmers with a comprehensive view of how successful implementations were achieved, how errors were resolved, and whether issues stemmed from code implementation or from missing or incompatible dependencies (Mysore and Guo, 2018). Dear Diary allows programmers to automatically obtain a structured and clear visualization of terminal commands and their outputs, as well as the dependencies related to the development and execution environment, at each stage of their development journey. They can annotate these with their insights and comments, enabling them to track not only the code but also the development and deployment setups throughout the implementation process.

Tracking and reviewing erroneous executions and the solutions adopted to overcome them: Given that Dear Diary is designed for programmers encountering a development framework for the first time, and given that this process is inherently non-linear with inevitable errors, our goal is to emphasize to non-experienced programmers that coding errors and faulty executions can be tracked and reviewed, as they are valuable learning resources. In practice, programmers usually search online forums, video tutorials, or official documentation (Ichinco and Kelleher, 2015; Head et al., 2018a), and they eventually solve the error after reading various sources and trying multiple code examples. While committing non-functioning versions of a software project is a bad practice, keeping track of the non-compliant code, erroneous executions, error messages, and especially how to overcome them is desirable in our non-experienced programmers' context. Thus, Dear Diary enables developers to capture non-functioning versions of the project, comment on non-compiling versions of the code files, and explain the erroneous command-line instruction outputs. Without documenting the erroneous executions, reporting the consulted sources to overcome the issues would not be possible. Furthermore, keeping track of the errors and adding self-explanations to their solutions can help novices remember their strategies and replicate them across diverse implementations (Vieira et al., 2017).

4.1. Implementation notes

4.1.1. Visual studio code integration

We implemented Dear Diary as a Visual Studio Code extension using TypeScript. We built its functionality upon the official Visual Studio Code API[3] and, as evidenced by the usage scenario, we carefully adhered

[3] "VS Code API | Visual Studio Code Extension API", https://code.visualstudio.com/api/references/vscode-api, last accessed November 11, 2025.

to the official Visual Studio Code user experience guidelines[4] to ensure the extension is as easy, intuitive, and seamless as possible for the average IDE user. Furthermore, we refined the graphical interface design through multiple rounds of discussions of hand-drawn sketches.

4.1.2. Snapshot management and Git integration

Dear Diary relies on Git to provide snapshot management features. Although the final user does not notice it, file management is achieved by integrating with Git. Therefore, in general terms, when Dear Diary is initialized, a Git repository is created in the workspace folder if one does not already exist, and a new commit is added to the repository each time a snapshot is taken. Then, to track the commits, associate them with snapshots, and store all user-entered data, Dear Diary manipulates a list of dictionaries in a hidden JSON file.

Each dictionary within that file corresponds to a snapshot and includes the following fields:

- **essential metadata** such as the snapshot's unique identifier and date of creation;
- the **user-entered data** (*e.g.*, title, description, tags, and the flag indicating whether the snapshot corresponds to a working version of the project);
- **commit-related details** stored in the `gitCommit` field, which contains a dictionary. The `hash` field within this dictionary is particularly important because, when a Dear Diary user navigates to a snapshot, the extension checks out the commit associated with it and renders the files, terminal commands, and dependencies accordingly in the graphical interface. In the same way, when the user switches from the Dear Diary interface to the Visual Studio Code Explorer in the activity bar, the extension automatically performs a checkout of the last commit, allowing the developer to continue working on the latest version of the project;
- and **files**, **terminal commands**, and **dependencies** fields, each of which is a list of dictionaries. Each dictionary, representing a file, terminal command, or dependency, contains essential metadata, user-entered data, and **component-specific fields**.

5. User study

We conducted a controlled user study to examine how programmers use Dear Diary while engaging with an unfamiliar development framework. Specifically, the study focused on the capture phase of the development journey: whether participants integrated the tool into an ongoing programming task, which Dear Diary features they used, what kinds of contextual knowledge they recorded, and how they perceived the tool's usability and relevance. This focus allows us to assess a necessary first step for supporting later reflection and reuse: the creation of rich, situated records of development activity.

To guide this evaluation, we addressed the following research questions:

RQ1. How do programmers use Dear Diary to capture their development activity while working with an unfamiliar framework?
RQ2. What kinds of contextual knowledge do programmers record through Dear Diary's snapshots, terminal commands, files, code snippets, and dependencies?
RQ3. How do programmers perceive the usability, relevance, and potential usefulness of Dear Diary for documenting development activity?

5.1. Participants

We recruited the participants from among former students of a master's degree course in web development who had recently passed their exams. The authors teach this course, which is mandatory for all Computer Engineering students and focuses on creating distributed modern single-page web applications, emphasizing front-end programming with the React framework (utilizing JavaScript).[5] In addition to technical skills, a primary goal of the course is for students to develop essential abilities in managing critical design decisions inherent in web application design and development.

Our decision to select participants from former students of this course achieved two main objectives. First, it ensured that they had a solid understanding of web application concepts and components, which was crucial for the programming task they were assigned—namely, working on implementing a web application, as detailed later in this section. Second, because Visual Studio Code was the primary development tool used throughout the course, and all the live programming sessions during the lessons were conducted using it and some extensions, we ensured to a reasonable extent that any potential issues regarding the interaction with Dear Diary were not related to any unfamiliarity with the IDE.

Concretely, the former students were contacted through an email invitation, which included a brief description of the study, the study's goals, and the expected time commitment. The email also contained a link to a Google Form where the students were asked to complete a pre-study questionnaire to determine their eligibility. In the questionnaire, they provided demographic data, their years of programming experience, and the programming languages they were familiar with. Additionally, they rated their familiarity on a Likert scale from 1 to 5 with React, JavaScript, Visual Studio Code, Angular, and TypeScript. Finally, they were asked to describe which Visual Studio Code extension they used and for what purpose. To reduce potential evaluation pressure, most participants (11 out of 18) were recruited with the help of the instructor of another section of the course, who did not take part in the study sessions, data collection, or analysis. The sessions themselves were conducted by a researcher who did not know and had not taught the vast majority of the participants (15 out of 18).

Two students were excluded: one rated his familiarity with Angular[6] as 5 (as will be extensively described in the next subsection, this was inconvenient for the study, which aimed to examine how participants used Dear Diary while working with an unfamiliar framework—in our case, Angular), and the other rated his familiarity with Visual Studio Code as 2. After screening, 18 students were selected to participate in the study (5 self-identified as women and 13 self-identified as men). Table 1 provides an overview of the participants' demographic data and programming experience. Their ages ranged from 22 to 25 years, with an average age of 23.11 years. All participants were enrolled in a master's program except for one pursuing a Ph.D. (**P18**). Their programming experience varied from 2 to 10 years, with an average of 4.82 years.

5.2. Method

The study, conducted in a controlled environment, comprised three phases: a **training phase**, a **programming phase**, and a final **assessment phase** that included a questionnaire and a semi-structured interview to evaluate participants' experience with Dear Diary both quantitatively and qualitatively. Each participant worked individually in a quiet room, using a laptop equipped with the Dear Diary extension, an external monitor, and a wireless keyboard and mouse to facilitate interaction and avoid the use of the laptop's keyboard and trackpad. Throughout all phases, the experimenter adhered to a pre-written script to ensure consistency, guaranteeing that all participants received the

[4] "UX Guidelines | Visual Studio Code Extension API", https://code.visualstudio.com/api/ux-guidelines/overview, last accessed November 11, 2025.

[5] "React", https://react.dev, last accessed November 11, 2025.

[6] "Home • Angular", https://angular.dev, last accessed November 11, 2025.

Table 1
Participants' demographics and programming experience.

ID	Age	Gender	Prog. Exp. (yrs.)	React (1–5)	JavaScript (1–5)	VSCode (1–5)
P1	24	Male	4	4	4	4
P2	23	Female	10	3	4	4
P3	23	Male	4	4	4	5
P4	25	Male	5	3	4	4
P5	23	Male	10	4	4	4
P6	22	Female	2	4	4	4
P7	22	Female	2	3	4	5
P8	23	Male	4	4	4	4
P9	25	Male	3	4	4	4
P10	22	Male	2	4	4	4
P11	24	Male	5	4	3	5
P12	22	Male	5	3	3	4
P13	24	Male	6	4	3	4
P14	23	Male	6	4	4	4
P15	22	Male	4	4	4	4
P16	23	Female	2	3	3	4
P17	24	Male	6	4	4	3
P18	22	Female	4	5	5	5

same information, that features were presented correctly during the training phase, that no instructions or explanations were omitted during the programming phase, and that all questions were asked during the final assessment phase. To avoid experimenter bias, all participants were guided by the same experimenter. The pre-written scripts used in the study are included in the Supplemental Materials.

5.2.1. Training phase

The training phase aimed to familiarize participants with Dear Diary and its features, ensuring they were well-prepared for the subsequent phases. During this phase, the experimenter and participant worked together through a pre-written script. The experimenter provided guidance and explanations as needed while the participant operated the computer and engaged directly with the tool. Participants were shown how to use the various features of Dear Diary while creating and executing a 'Hello World' web application from scratch using React, a framework they were familiar with. This allowed them to focus on understanding the tool's features rather than the implementation.

The task, although straightforward, was designed to exemplify how and when to use each Dear Diary feature. We demonstrated the application of these features at specific points during implementation and their intended purposes. An erroneous application execution was intentionally included to show how to explicitly track issues using Dear Diary. The training paid particular attention to presenting the functionalities in the typical sequence of development activities, illustrating how the tool can be seamlessly integrated into participants' regular development workflows.

At the end of the training, participants were asked eight questions orally to confirm their understanding of Dear Diary's features and functionalities. This assessment also checked their ability to use the tool effectively and allowed them to resolve any remaining questions or doubts. When participants indicated that they understood a particular functionality, they were asked to demonstrate it.

5.2.2. Programming phase

In this phase, we aimed to examine how participants used Dear Diary to capture their progress and contextual knowledge while working with an unfamiliar framework. Participants were tasked with creating a web application to check the weather forecast for a user-specified city using the Angular framework. The app had to include a text field for entering the city name, a button to submit the request to a weather API, and a display area for the forecast results.

Participants received a document divided into two parts. The first part guided them in setting up the development environment and introduced the fundamental concepts of Angular and Angular CLI. The second part outlined the required functionalities and provided general instructions on the components to use. This second part was intentionally not written as a step-by-step tutorial, allowing participants to explore and experiment with Angular, simulating a real-world learning scenario.

The programming task was designed to be of medium-high difficulty to challenge participants, encouraging them to consult external sources like documentation, tutorials, and forums as they encountered obstacles. Participants were encouraged to refer to any documentation or external sources and were free to generate various forms of documentation, such as code comments and handwritten notes. Most importantly, they were explicitly told that the primary goal was to examine their use of Dear Diary during the development process, not to complete the task within a specific time limit or ensure flawless functionality.

Participants were asked to approach the task naturally, focusing on their usual development practices. For example, if a participant typically searches for documentation online before coding, they were encouraged to do so. If they encountered an issue, they were encouraged to take their time to resolve it, even if it meant consulting external sources or experimenting with different solutions. AI coding assistants were not installed on the laptop provided for the study. Their use was neither explicitly permitted nor prohibited in the study protocol, and none of the participants used AI coding assistants or web-based generative AI tools during the programming task. They were given 40 min to work on the task, and their screens were video-recorded. The experimenter was present but did not interfere with or observe the participants to avoid adding pressure.

5.2.3. Assessment phase

Immediately after the programming phase, participants were asked to complete a **questionnaire** and participate in a **semi-structured interview**. The questionnaire aimed to collect quantitative data on participants' experiences with Dear Diary, focusing on satisfaction, ease of use, perceived usefulness, and the likelihood of recommending the tool to others. The interview aimed to gather qualitative data on participants' perceptions, opinions, and suggestions for improvement.

The questionnaire consisted of 8 questions (inspired by the ones formulated by Horvath et al. (Horvath et al., 2022a)) with responses rated on a Likert scale from 1 to 7, ranging from '*Completely Disagree*' to '*Completely Agree*.' To ensure honest and unbiased responses, participants completed the questionnaire alone while the experimenter remained out of sight, and the responses were collected anonymously. These measures were intended to reduce evaluation pressure, although, as discussed in Section 6.2, they cannot fully eliminate possible social desirability effects given the broader academic context of the study.

Half of the interview questions addressed the participants' current habits for tracking their development journey, such as note-taking, documenting important information and solutions, and strategies for recalling past solutions. This was important because the relevant baseline for Dear Diary is not a single tool or practice, but a heterogeneous set of strategies that programmers may combine differently, including code comments, external notes, terminal histories, Git-based histories, browser bookmarks, documentation pages, and personal recall. The interviews therefore allowed us to characterize participants' existing documentation practices and situate their perceptions of Dear Diary in relation to them, although the study did not include a controlled comparison condition.

The other half focused on participants' insights, thoughts, and experiences with Dear Diary. These questions explored how they used the tool during programming, scenarios where the tool could be more effective, features they found most and least helpful or confusing, and any additional functionalities they would like to see. There was no time limit for the interview, allowing participants to elaborate on their responses as much as they wished. At the end of this phase, all participants were rewarded with a university-branded mug.

5.3. Data analysis

As mentioned in the previous section, Dear Diary's persistence was implemented by saving a log of all the interactions with timestamps in a hidden JSON file. This implementation decision proved highly useful for accurately analyzing participants' tool usage and identifying usage patterns among the participants. Additionally, the laptop screen during the programming sessions was recorded in case additional insights into user interactions were needed. In practice, however, the recordings were only used to determine the exact moment when users completed the programming task to build the consolidated timeline of the participants' interactions with the tool (depicted in Section 5.4).

For the qualitative assessment, the audio recordings of the interviews were transcribed using Microsoft Word's transcription feature, and the transcripts were subjected to qualitative thematic analysis using an inductive and reflexive orientation (Braun and Clarke, 2006; Kiger and Varpio, 2020; Braun and Clarke, 2019). Consistent with this orientation, we treated coding as an interpretive activity; analytic rigour was supported through independent initial coding, collaborative discussion, and the reporting of illustrative extracts, rather than through inter-rater reliability metrics. Each author independently reviewed the transcripts and generated initial codes without sharing them, to avoid influencing each other's criteria. The authors then discussed these codes to compare and discuss interpretations, identify common patterns and the most revealing data extracts, and agree on three themes. These themes were used to structure the qualitative results (Section 5.5).

5.4. Quantitative results

To address **RQ1**, concerning how programmers use Dear Diary to capture their development activity, we analyzed participants' interaction logs, focusing on how often the tool's features were used and how these interactions unfolded throughout the programming task. To address **RQ2**, concerning the kinds of contextual knowledge recorded with Dear Diary, we examined the content participants created through titles, descriptions, notes, tags, and code-snippet annotations.

Table 2 outlines the number of times each feature was used by participants during the programming task, the participants who used each feature the least and the most, and the mean and standard deviation for each feature. This initial quantification indicates diverse participant behaviors, with substantial differences in how frequently and variably different operations are utilized.

Given the relatively short time allocated for the programming task (*i.e.*, 40 min), we were pleasantly surprised to see that all participants took at least one snapshot, with an average of three snapshots per participant. This behavior suggests that, as we intended when designing the tool and the snapshot feature, participants used snapshots not only to keep track of working versions of the project but also to monitor small changes, difficulties, and progress toward a working solution. Specifically, **P9** created the most snapshots with a total of 7, while **P4**, **P10**, and **P14** each created only one. Interestingly, one of the participants who made the fewest snapshots (**P4**) was also among the participants who created the most code snippets (7).

Table 2
Summary of Dear Diary feature usage during the programming task.

Feature	N	Min Count (by Participants)	Max Count (by Participants)	Mean	SD
Snapshots	55	1 (P4, P10, P14)	7 (P9)	3.06	1.66
Terminal Commands	95	0 (P3)	15 (P18)	5.28	3.72
Dependencies	2	0 (all except P1 and P16)	1 (P1, P16)	0.11	0.32
Files	33	0 (P5, P8, P15, P18)	12 (P3)	1.83	2.71
Snippets	38	0 (P5, P7, P10, P12, P16, P18)	7 (P4, P17)	2.11	2.47

Annotating terminal commands, for its part, was the most popular feature, with an average of 5.28 annotations per participant. Indeed, all of them, except **P3**, annotated at least one terminal command, with **P18** making the most annotations at 15. However, similar to what happened with the snapshots' feature, while **P3** did not annotate any terminal command, he made the most file annotations (12). Conversely, **P18**, who annotated the most terminal commands, made the fewest file annotations. This variation highlights the participants' diverse preferences for interacting with Dear Diary's features.

Regarding file annotations and code snippet creation, the higher standard deviation compared to the mean for both features indicates significant variability in usage. As shown by the maximum and minimum values in the table, some participants used these features much more frequently than average, while others used them minimally or not at all. Finally, the least used feature was dependency annotation, with an average of just 0.06 annotations per participant. Only two participants used this feature: **P3** and **P18**, each annotating one dependency. This low usage suggests that either participants found this feature less helpful than others or they did not have sufficient time to explore it during the programming task.

Additionally, to complement Table 2 and to provide a deeper look at the features used by participants, Table 3 presents statistics on the content created for each feature. It details how often titles, descriptions, or notes were written and includes descriptive statistics on word length.

As we intended during the design of Dear Diary, the various descriptions and notes comprised:

- **actions taken**, *e.g.*, **P1**: "Added a class to implement the weather fetching service," **P11**: "Application setup is done. Everything is working fine!" **P14**: "I insert all the variables that I can reference in HTML, similar to 'props,'" and **P9**: "Corrected the terminal commands that were previously wrong";
- **questions or exploratory details**, *e.g.*, **P2**: "How do I create a form and get the value from the form to change this variable?" **P17**: "Should it be a variable? Maybe something like useState?" and **P11**: "I had a problem while creating the app that I solved by executing 'sudo npm install.' A missing package probably caused the problem";
- **references to external resources**, *e.g.*, **P16**: "useful: https://openweathermap.org/api/geocoding-api (to get weather by city)," and **P2**: "How do I create a form and get the value from the form to change this variable? I was reading this article: https://v17.angular.io/guide/forms";
- **instructions or reminders**, *e.g.*, **P15**: "Remember to import the module to use it," **P6**: "Note that the name to enter here is the selector name specified in the component.ts," **P11**: "Remember

Table 3
Counts for notes, titles, and descriptions per feature, and word count statistics. The values in the **N** column indicate how many times titles, descriptions, or notes were present in the feature occurrences. The percentage reflects the proportion of all occurrences that included the specified content.

Feature		Word count			
Snapshots	N	Min.	Max.	Mean	SD
Title	55 (100.0%)	1	8	2.78	1.32
Description	38 (69.0%)	2	40	9.16	8.89
Terminal Commands	N	Min.	Max.	Mean	SD
Note	90 (94.7%)	2	196	9.39	21.20
Dependencies	N	Min.	Max.	Mean	SD
Note	1 (50.0%)	6	6	6.00	0.00
Files	N	Min.	Max.	Mean	SD
Note	23 (69.7%)	1	26	8.78	7.05
Snippets	N	Min.	Max.	Mean	SD
Note	38 (100.0%)	2	17	7.16	3.51

Table 4
Tagged features summary.

Feature	N	Min.	Max.	Mean	SD
Snapshots	43 (78.2%)	1	4	1.40	0.69
Terminal Commands	81 (85.3%)	1	3	1.22	0.47
Dependencies	2 (100.0%)	2	2	2.00	0.00
Files	20 (60.6%)	1	3	1.45	0.69

to run the command with privileges," and **P4**: "This is the app configuration file";
- and **issues or errors encountered**, *e.g.*, **P13**: "Non-working line: <app-hello-world> is not a known element," **P9**: "Suggested correction for the terminal issue," and **P17**: "Error importing the city input component into the app component".

Remarkably, one type of content we had not anticipated and that several participants created spontaneously using the tool was the TODOs. These were used to describe pending development **tasks to be completed**, *e.g.*, **P1**: "TODO: Implement the service and correctly set up the city input form in the template," **P12**: "Form added to the front end; onSubmit still needs to be handled" and **P18**: "Generated a new component. Still need to: (i) modify the generic code in <comp-name>.component.ts and <comp-name>.component.html; (ii) import the <comp-name> component into app.component.ts and use it".

On the other hand, concerning Dear Diary's support for tracking non-working application versions and unsuccessful terminal commands, the application status field ('*Is the application working?*') was used in 45 of the 55 snapshots (81.8%): 39 out of 45 snapshots indicated working versions, while six were marked as not working. In the case of terminal commands, the success field ('*Did the terminal command run successfully?*') was used in 87 out of 95 cases (91.6%): 58 were successful, and 29 were not. Unsuccessful terminal commands were commonly tagged with the label "error," and other tags included, for instance, "import errors" (**P1**), "typo" (**P2**), "wrong command" (**P5**), "fail" (**P6**), and "spelling mistake" (**P17**), among others.

Moreover, these commands were annotated with: ***explanations*** (*e.g.*, **P1**: "Missing import in app.component.ts", **P8**: "I didn't include the component in the import", **P18**: "Doesn't work since permissions are needed"); ***questions*** (*e.g.*, **P2**: "I don't know why it says it can't find it; did I import it wrong?", **P18**: "I got an error. Why? I forgot to run 'npm -v' first"); ***solutions*** (*e.g.*, **P5**: "The right command is ng generate component hello-world"); ***uncertainties*** (*e.g.*, **P6**: "It didn't generate the component, but I don't understand why", **P11**: "I didn't understand whether the installation worked because I got an error, but the app was created anyway. Check it later, maybe."); ***fragments from the terminal output, captured by Dear Diary*** (*e.g.*, **P9**: "Your cache folder contains root-owned files, due to a bug in previous versions of npm which has since been addressed (…)"); and ***TODOs or reminders*** (*e.g.*, **P16**: "Unknown Angular error; need for sudo (maybe?), investigate!").

Lastly, regarding the possibility of attaching tags to enable cross-cutting searches within each snapshot, Table 4 presents the number of snapshots, terminal commands, dependencies, and files with at least one tag (N), along with the minimum, maximum, and average number of tags per feature, as well as the standard deviation.

To accurately visualize and analyze each participant's use of Dear Diary during the programming task, we created a timeline of interactions, as shown in Fig. 3. This timeline helps us understand how participants engaged with different features and how their interactions evolved. Each feature is marked distinctly, and colors indicate the snapshot number, ranging from 1 to 7, which reflect the minimum and maximum number of snapshots. To standardize the figure, all interaction timestamps were normalized to a 40-minute duration, as shown on the x-axis. This adjustment accounts for minor variations in task completion times, such as differences in time spent on final annotations and saving. The timeline corroborates some of the observations previously described and reveals additional insights, namely:

- In all cases, participants' **interactions with the tool were spread throughout the programming task**. Regardless of how frequently they used Dear Diary, both frequent and infrequent users engaged with it at various points during the implementation. This observation is noteworthy, as a possible trend might have been to document everything at the end of the allotted time. Instead, participants used the tool continuously, indicating that it integrated smoothly into their workflow and allowed them to track their progress and thoughts seamlessly without being particularly disruptive or distracting. Indeed, one of our design motivations was to encourage this behavior by integrating the tool directly into the IDE.
- The **lack of a common pattern in participants' interactions** with Dear Diary—regarding timing and types of interactions—suggests that our programming task was sufficiently open and flexible, allowing each participant to approach it differently. In this regard, while some participants began interacting with the tool from the very beginning, others had few interactions during the first 10 min. This flexibility is a positive aspect of the task, as it encourages participants to engage with the tool in a way that best suits their needs.

  More importantly, **the differences in how often each user used different features suggest that participants appropriated the tool in diverse ways**, accommodating what each participant deemed most relevant to track. For instance, while the interactions of **P2** and **P18** suggest that the participants focused on tracking executed terminal commands, **P3** and **P13** preferred annotating files, and **P17** was more interested in creating and annotating code snippets. There was also the case of participant **P9**, whose interactions demonstrated the use of all the tool's features. These interactions were almost equally distributed between the first and second halves of the implementation, with terminal commands used predominantly in the first half and files and snippets used in the second half.
- Although Dear Diary allows users to navigate between snapshots that have been created and modify them and their resources at any time, **the prevalent trend was not to go back and modify resources belonging to past snapshots**. As can be discerned from the colors assigned to each snapshot in the timeline, participants' most commonly adopted behavior was to annotate files, create snippets, annotate terminal commands and dependencies, and then subsequently create the snapshot. This behavior reflects an incremental approach in the use of Dear Diary. A more complex or extended development task could likely encourage the navigation and modification of past snapshots.
- Another trend we identified in using Dear Diary was **the creation of snapshots that do not contain code snippets or annotations for files, terminal commands, or dependencies**. For instance, in the timeline of **P12**, the second, third, and fourth snapshots are not preceded by any other operations. A similar pattern is observed with the second and third snapshots of participant **P8**. In both cases, participants added very brief descriptions to the snapshots, serving as checkpoints to indicate what had been achieved at that point in the implementation (**P8**: "Created the project and launched the app"). Even in this mode of use, the extension allows participants to add more detailed descriptions and tags to describe progress in greater detail. Besides, participants can later return to a particular snapshot and perform operations on the resources of that specific version of the program.
- In the timeline graph, it can be observed that **some interactions occurred almost simultaneously**. Notice how the opacity of the color in some markers is more intense than in others. In these cases, the time elapsed between the two interactions was so short that,

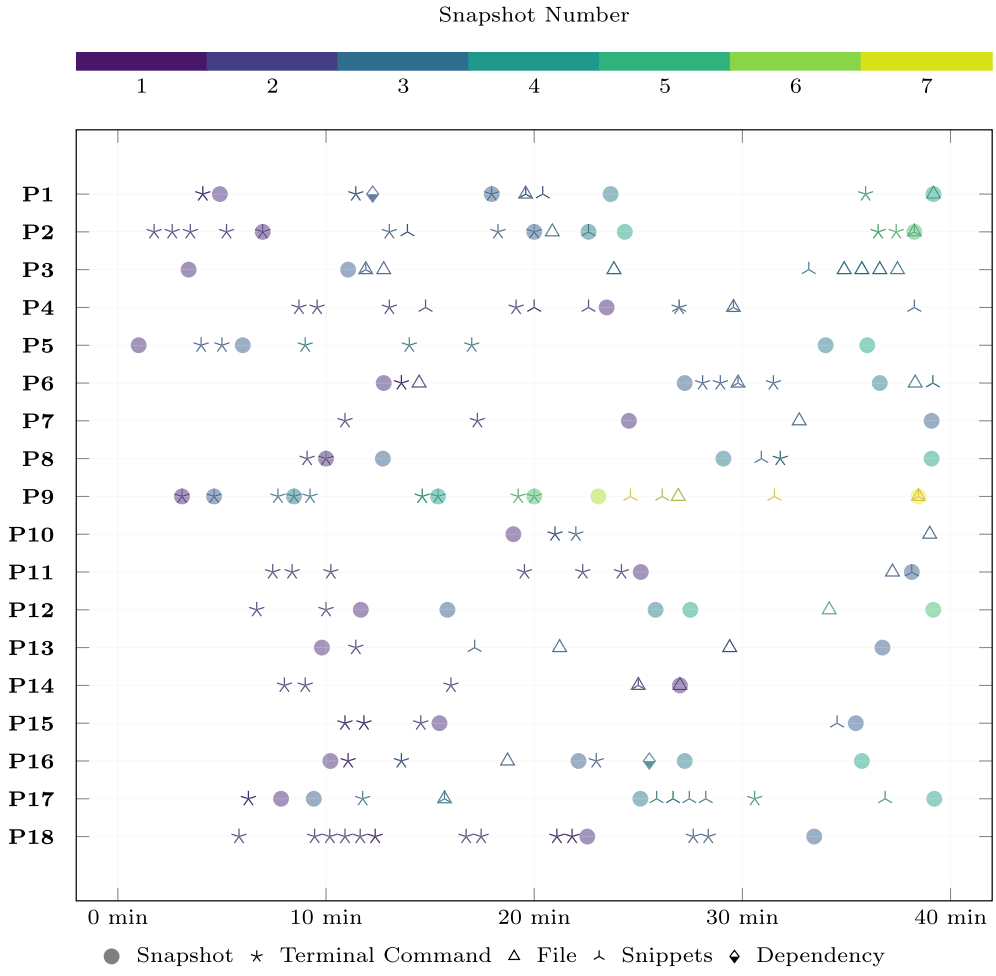

Fig. 3. Dear Diary usage timeline per participant.

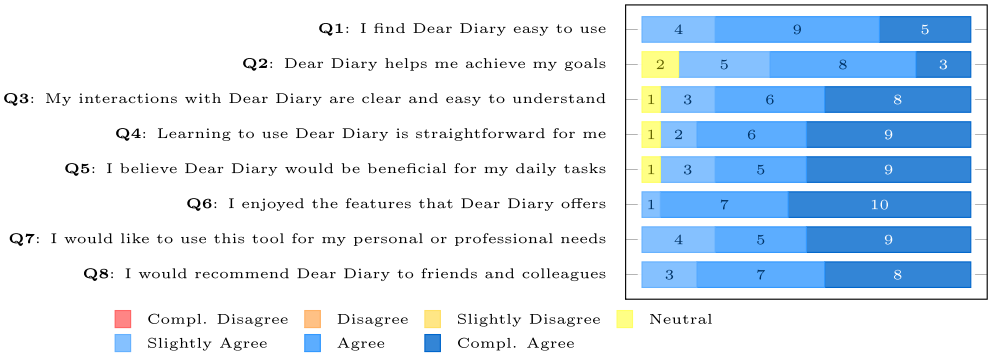

Fig. 4. Distribution of responses to the assessment phase questionnaire.

when plotted, the markers appear overlapping. This occurs both with interactions of the same type and between different types of operations, as seen, for example, in the timeline of participant **P1** when, around the 20-minute mark, they logged a terminal command and almost simultaneously created a snapshot. Shortly after, they commented on a file and created a code snippet. In our view, the fact that this occurs indicates the ease of use of the tool, which allows for smooth and quick use of the features required to track the various resources involved in the development process.

To address **RQ3**, which concerns participants' perceptions of Dear Diary's usability, relevance, and potential usefulness, we first analyzed their questionnaire responses and then complemented this analysis with the semi-structured interviews presented in the following section.

Fig. 4 shows the distribution of responses to the assessment phase questionnaire, while Table 5 complements this visualization by reporting item-level descriptive statistics. Overall, participants reported positive perceptions of Dear Diary's ease of use and clarity. They also reported positive perceptions of Dear Diary's relevance to their tasks

Table 5
Item-level descriptive statistics for the assessment phase questionnaire. Responses were collected on a 7-point Likert scale, from 1 ('Completely disagree') to 7 ('Completely agree').

Item	N	Min.	Max.	Median	Mean	SD
Q1: I find Dear Diary easy to use	18	5	7	6.00	6.06	0.73
Q2: Dear Diary helps me achieve my goals	18	4	7	6.00	5.67	0.91
Q3: My interactions with Dear Diary are clear and easy to understand	18	4	7	6.00	6.17	0.92
Q4: Learning to use Dear Diary is straightforward for me	18	4	7	6.50	6.28	0.89
Q5: I believe Dear Diary would be beneficial for my daily tasks	18	4	7	6.50	6.22	0.94
Q6: I enjoyed the features that Dear Diary offers	18	5	7	7.00	6.50	0.62
Q7: I would like to use this tool for my personal or professional needs	18	5	7	6.50	6.28	0.83
Q8: I would recommend Dear Diary to friends and colleagues	18	5	7	6.00	6.28	0.75

and willingness to continue using it for personal or professional needs and to recommend it to friends and colleagues. Across the questionnaire, all items had median values of 6.00 or higher, and mean values ranged from 5.67 to 6.50 on the 7-point scale.

Q2 ('*Dear Diary helps me achieve my goals*'), for its part, showed the lowest mean score among the questionnaire items, although it remained above the midpoint of the scale. We interpret this pattern in light of the study design. Unlike items focused on ease of use, clarity, or feature relevance, Q2 asks participants to judge Dear Diary's contribution to goal achievement. This is a broader judgment, especially in a short task where the programming goal was defined by the study rather than by participants' own ongoing projects. For this reason, we treat Q2 as a cautious indicator of perceived usefulness and as motivation for future longitudinal studies in which developers use Dear Diary toward self-defined development goals over time.

5.5. Qualitative results

Qualitative results stem from the analysis of the semi-structured interviews conducted with all participants. These interviews averaged 25 minutes in duration, with the shortest lasting 12 minutes and the longest lasting 1 h, and 8 min. The standard deviation of the interview lengths was 14 min. Below, we present the three themes identified as the most relevant from our thematic analysis.

The semi-structured interviews further informed **RQ3** by providing detailed accounts of how participants interpreted Dear Diary's features, possible use cases, and relationship to their existing documentation practices.

5.5.1. Comprehensive feature coverage and IDE integration as drivers of positive perception and willingness to use the tool

When asked about their favorite and most useful features, all features—except dependencies—were selected by several participants. Regarding dependencies specifically, several participants mentioned they had not used this functionality primarily because the time allocated for the programming task was limited and the required functionality was very specific. Consequently, they did not have the opportunity to compare and evaluate different library options during development. On the other hand, **P14** indicated that "once I find that library or dependency, I install it, and I know why I'm installing it theoretically (…) I wouldn't see myself using it (the dependency annotation functionality) anymore."

With the files and code snippet features, the most appreciated aspects were: firstly, the ability to avoid cluttering the code (*e.g.*, **P17**: "In the code comments, I might keep things more concise to avoid cluttering the code. However, since they (Dear Diary code snippets) are separated from the code, I feel freer to explain things in more detail"); secondly, the possibility to describe the functionality informally and, regardless of how large a code file is, quickly locate the parts of the code where comments were added (*e.g.*, **P3**: "in general, the ability to tag or comment on something and then search for that keyword to find where it is in the code is very useful. Being able to comment on a component, like saying 'this component does this,' and then searching for it by keyword is very helpful for me"); and thirdly, the accuracy and specificity of the code snippet annotations, which are precisely linked to specific code fragments (*e.g.*, **P3**: "The ability to take notes, especially those associated with a file or, even better, with a specific code snippet, is what I found most useful," and **P2**: "the fact that you can add both general comments on files and specific comments on code seems quite versatile and addresses most of what needs to be covered in the project").

Many participants highly appreciated the terminal commands feature. This appreciation stemmed from the fact that, despite most installation, configuration, and project execution tasks being done through terminal commands, programmers currently lack a way to track the commands they have executed or the results of each command (*e.g.*, **P9**: "at least in the code, you can add a comment, but in the terminal, you can't put anything," **P11**: "without Dear Diary, after you execute a terminal command, you close the session, and you lose everything," and **P6**: "(the terminal command feature) is very nice because I always forget which commands to use to install certain dependencies"). Furthermore, even if they wanted to and attempted to do so, without Dear Diary, participants found it challenging to keep track of their executed terminal commands and their outputs (*e.g.*, **P2**: "(…) it's impossible to do otherwise, and it's a bit tiring to remember to go and write them down in a separate file," and **P9**: "(…) for example, [with Dear Diary] when you execute a command, it shows the output directly below it. This is very convenient because saving things in your own file can become a 20-page document and get confusing. With Dear Diary, it's much clearer because you have the output for each instruction and your corresponding comment").

Similarly, other participants found the snapshot feature particularly useful because it enables them to easily track project versions over time and navigate to previous snapshots to review progress at specific moments, thanks to the annotations, tags, and indications of erroneous or successful project executions (*e.g.*, **P1**: "the part I personally find most useful is the snapshots because they allow the user to see the entire situation at a specific moment in time without needing to use a Git repository, perform revert operations, or navigate back and forth. Instead, we have a personal log where we already know what the snapshots are and what they do. We also have high-level, qualitative comments that make it easier to manage," and **P1**: "saving snapshots of important points in what you've done can be useful for later when you need to investigate and recall how you solved a particular problem (…) to recall how you solved it, you might have a more detailed outline compared to what you would have without it").

Because the interviews asked participants about their existing practices for tracking development knowledge, the qualitative results also help situate Dear Diary in relation to those practices. Participants described heterogeneous strategies that often combined multiple partial solutions, such as code comments, external files, terminal history, Git commits, and personal memory. These contrasts with Dear Diary should therefore be understood as situated qualitative comparisons with their own documentation practices, rather than as a controlled experimental baseline.

In this context, aside from participants' preferences for specific functionalities, we observed that a significant part of their positive perception and willingness to use the tool lies in its ability to track the

development journey directly from the IDE. This capability helps to firstly, decrease cognitive load (*e.g.*, **P5**: "I use external files to save various things, and it can be mentally taxing to switch windows. If I'm reflecting on a specific piece of code, switching to a completely different view can be complicated and disrupt my logical flow. With this tool, I can mark things with just a button and then return to coding, which helps me maintain my focus"); secondly, prevent forgetting important information (*e.g.*, **P3**: "maybe at that moment I write something down thinking I'll remember it, but later I don't know where it went. Instead, with Dear Diary, my comments are precisely linked to the code"); and thirdly, maintain the project's resources in their original format, particularly concerning the code. This is not possible when copying and pasting code into other programs, where indentation, text colors, and fonts generally become distorted (*e.g.*, **P6**: "I don't like having code in plain text format; it bothers me. I prefer having colors to distinguish different elements. So, it's actually convenient to see the code with all the context directly. It's useful to have everything clearly presented").

Taken together, these accounts suggest that participants valued IDE integration not only as a matter of convenience, but because it reduced the translation work normally required to move situated development knowledge into external documentation. When notes are kept in separate files, programmers must decide what to copy, how to preserve formatting, how to describe the relevant context, and later reconnect the note to the corresponding code, command, or project state. Dear Diary's design reduces this gap by keeping explanations close to the artifacts that produced them. This helps explain why participants valued features differently: code snippets, files, terminal commands, and snapshots each supported a different kind of contextual anchoring. The broader insight is therefore not simply that participants liked several features, but that they perceived value in having multiple artifact-specific ways to capture knowledge without leaving the development environment.

5.5.2. Participants' perspectives on the potential applications of Dear Diary: collaborative and academic use cases

During the interviews, participants were asked about scenarios where the tool could be more effective. They emphasized that in collaborative settings, tracking the development process serves as valuable guidance for other programmers. In words of **P17**: "(Dear Diary) can be useful in a team to keep track of things that might be missed. For instance, it helps someone who doesn't understand a decision made by another person to grasp the reasoning behind it, allowing them to see not just what a particular version does, but also how the person arrived at that decision."

In line with this collaborative potential, other participants highlighted the relevance that Dear Diary could have in other courses they were taking. Specifically, **P5** envisioned the tool relevance in a Software Engineering course, where work is done in groups, and particular attention is given to the design phase of the application: "(…) maybe not only during the programming phase but also in the design phase. In my opinion, this could be useful. For example, we have a design phase in the course where we document our design. Having a tool that tracks all the research we've conducted and the information we've gathered from the internet could be very beneficial." **P2**, for their part, expressed their willingness to use the tool in a Machine Learning course group project where error tracking was crucial: "In the course project, which was quite complicated and done in pairs, I tracked the errors I encountered and how I fixed them by linking to the relevant pages, knowing I might have to explain it to someone else trying to do the same thing. However, I always used text files for this."

Interestingly, **P16**, who also works as a data scientist at a development company, highlighted Dear Diary's collaborative potential in a professional setting, noting that it would have significantly improved their experience when getting involved in legacy projects: "I would gladly use it at work with others. I mean, I would have liked it if, when I started at work, all the projects had this kind of tracking. Instead, I had to check everything on my own, and often it wasn't clear what had been done. Having this type of tracking would have been a game-changer for me."

Aside from its collaborative potential, participants also suggested that Dear Diary could be used in academic settings, particularly during live programming sessions. In the case of the course attended by the study participants, this corresponded to the second half of the lectures, where the professor illustrated how to apply the theoretical concepts by working on an exercise that was implemented incrementally throughout the semester (*e.g.*, **P2**: "I think it might be more useful during the semester when you're attending classes (…) even the professor encountered errors during live programming, and it can be useful to note these down. Indeed, I tried adding comments like '*if you do this, this happens*,' to document common errors that the professor encountered and later happened to me while working on the project," and **P12**: "Generally, during exercises where the professor writes the code and you just listen, you learn to understand what potential errors might be. However, this could be a way to integrate that knowledge").

These examples extend the value of Dear Diary beyond individual note-taking. Participants' suggested collaborative and academic scenarios point to a transfer problem: development knowledge is often created in one person's activity, but later needed by someone else, such as a teammate, a student, or a developer joining an existing project. In these situations, the relevant information is not only the final code or the final working version, but the path through which the result was reached, including decisions, errors, consulted sources, and explanations that may otherwise remain implicit or disappear after the task. Dear Diary's snapshot-based and artifact-linked structure was therefore perceived as a way to make this situated knowledge more shareable and teachable. At the same time, these scenarios also reveal that the tool's usefulness may depend on social and organizational practices, such as whether teams or instructors consistently create and maintain meaningful records.

5.5.3. 'I don't see it as a tracking of changes, i see it more as a storytelling of the project'

As the title of this subsection indicates (quoted from **P16**), participants highlighted how Dear Diary differs from VCSs as well as its benefits. Expanding on this, **P16** provided a key insight into these differences that aligns well with our design goals and the distinguishing factors of our tool compared to VCSs:

> If, for example, I come across a project that I really like, I can't simply arrive and see how the new project is set up. I can't just install and run it and then understand how it was done. I need to figure out what the developer did—what are the main components, what errors occurred, and how they were resolved. For instance, if an error was resolved by re-running a command, understanding this process is crucial. Essentially, **it's about understanding the developer's reasoning and approach rather than just knowing what a specific version does**.

Along the same lines, **P2** pointed out that while commit messages have a maximum length that limits the ability to fully explain the reasoning behind implementations, Dear Diary allows for more extensive explanations and detailed commentary on the development process: "Instead of asking other programmers multiple questions or trying to understand what they wrote in the commit message, which might still need explanation, it's much more convenient this way. You can provide the entire block of information, and they can see, 'okay, they did this for this reason and in this manner.' "

Furthermore, **P8** also emphasized the advantages of Dear Diary, noting that it offers more specificity than VCSs. With features like snapshots, detailed code comments, and tags, Dear Diary provides a richer source of information: "Dear Diary is much more specific; you can use snapshots, comment on the code in more detail, and use tags. So, there is more information available. Overall, it is much more useful from that perspective." Additionally, **P7** highlighted that Dear Diary helps maintain

a coherent trace of the development journey and the reasoning behind implementation choices, which is especially important in group work: "Through commits, I can see the history and retrace what I've done. However, when working in a group, it's crucial to update everyone step by step to maintain a coherent workflow and help others understand, because while you might grasp what you've done, others may not."

This theme clarifies that participants did not interpret Dear Diary merely as a more detailed version-control interface. Rather, they framed it as a different kind of record: one concerned with the meaning of development actions, not only their occurrence. Git-based histories can show that a change happened and, through commit messages, provide a short explanation; participants' accounts suggest that Dear Diary can complement this by capturing why a path was taken, which alternatives or errors shaped it, and how heterogeneous artifacts such as commands, outputs, files, and snippets relate to one another. The phrase "storytelling of the project" captures this distinction: the value lies in preserving a narrative of reasoning and recovery that can make a development journey intelligible to the original programmer or to others later.

6. Discussion and future work

6.1. Discussion

The study results provide an initial account of how Dear Diary supported participants in capturing their development activity during a controlled framework-learning task. The combination of interaction logs, questionnaire responses, and semi-structured interviews offers a nuanced perspective on how participants integrated the tool into their workflow, which features they used, what kinds of contextual knowledge they recorded, and how they perceived its usability and relevance. These findings speak primarily to the capture phase of the development journey: the creation of rich, artifact-linked records that can later support reflection, retrieval, and reuse.

- One of the most relevant findings was that the timeline visualization allowed us to observe how participants used Dear Diary throughout the programming task, rather than saving all their notes for the end. This continuous engagement suggests that participants were able to incorporate the tool into the ongoing programming activity, at least within the controlled setting of the study. The pattern of annotating files, creating snippets, and then generating snapshots points to an incremental capture practice, where participants recorded progress, decisions, and issues step by step. This behavior aligns with Dear Diary's goal of supporting documentation "on the go" directly within the IDE.
- The qualitative and quantitative outcomes suggest that participants particularly valued Dear Diary's integration with the IDE, especially for managing code comments, snippets, and terminal commands. Participants' comments indicate that handling these elements directly within the IDE may reduce context switching, help prevent information loss, and preserve the original formatting of code. These findings support the design rationale for IDE integration, while longitudinal and comparative studies are needed to examine how these perceived advantages relate to programmers' existing documentation workflows.
- From the participants critical insights during the semi-structured interviews, emerged the fact that in academic and collaborative contexts, Dear Diary was seen as a valuable tool for enhancing both individual and group projects. Participants highlighted its potential for detailed tracking of design decisions and error resolutions, which could facilitate better learning and understanding. The tool's applicability in live programming sessions was also noted, as it could aid in documenting errors and solutions during instructional exercises, enriching the learning experience and providing a detailed record of common issues and their resolutions (Fast et al., 2014).
- In contrast, the dependency annotation feature saw minimal use, with only two participants engaging with it. This limited engagement suggests that the feature may have been overlooked or deemed less relevant compared to others. This finding points to a potential area for improvement, indicating a need to explore why this feature did not gain traction and how it could be made more appealing or useful for users.
- An unexpected yet notable finding was the spontaneous creation of TODOs by several participants. This feature, which was not initially anticipated, became a significant part of the note-taking process. This behavior reflects Dear Diary's adaptability and its ability to evolve with user needs, supporting task management and planning within the development process. The integration of TODOs illustrates the tool's flexibility in accommodating various documentation practices. Integrating this feature could be particularly relevant, as literature suggests that a significant proportion of TODO comments in open-source projects remain unresolved or take a long time to be addressed (Wang et al., 2024b).

Finally, the increasing use of AI-assisted development tools raises important questions about how Dear Diary could fit into future programming workflows. AI coding assistants can help developers generate code, explain errors, suggest terminal commands, or summarize implementation alternatives. However, these tools do not automatically preserve a situated record of the developer's journey across project states, terminal outputs, dependencies, unsuccessful attempts, and self-explanations. In this sense, Dear Diary addresses a complementary problem: helping programmers externalize and organize the knowledge that emerges while developing, including what they tried, what failed, what they accepted, what they rejected, and what they want to remember.

At the same time, AI could become a valuable means for facilitating this tracking process. For example, future versions of Dear Diary could use AI to suggest tags, summarize terminal outputs, identify potentially relevant errors, or draft initial descriptions for snapshots, files, commands, and dependencies. Nevertheless, we see such support as a human-in-the-loop aid rather than as a replacement for the programmer's agency. A central design principle of Dear Diary is that programmers should be able to describe their development journey in their own terms, selecting what is meaningful and articulating explanations that will remain understandable and useful to themselves or others later. Fully automating this process could undermine the reflective value of the diary metaphor; AI assistance should therefore reduce friction while preserving users' control over what is captured and how it is explained.

6.2. Limitations and scope of claims

Our study has limitations related to participant sampling and the interpretation of self-reported data. Participants were recruited among former students of a web development course taught at the authors' institution, which may have introduced sampling bias and possible social desirability effects. However, most participants (11 out of 18) were recruited through the instructor of another section of the course, who did not take part in the study sessions, data collection, or analysis. In addition, the sessions were conducted by a researcher who did not know and had not taught the vast majority of the participants (15 out of 18), and questionnaire responses were collected anonymously. These measures mitigate the risk of direct instructor–student pressure, but they do not fully eliminate the possibility that participants' questionnaire ratings and interview responses were influenced by the broader academic context. Therefore, we interpret these self-reported data as indicators of perceived usability, relevance, and potential usefulness, rather than as standalone evidence of objective effectiveness.

Our participant sample should also be considered in relation to the goals of the controlled study. Recruiting graduate Computer Engineering students who had recently completed a web development course allowed us to involve participants who were familiar with web application

concepts and Visual Studio Code, while still being unfamiliar with the framework used in the task. This made the sample suitable for studying the initial use of Dear Diary during a framework-learning activity. At the same time, the sample was relatively homogeneous in terms of age, educational background, and institutional context. Therefore, our empirical findings should be interpreted as initial evidence from this specific population and setting, rather than as representative of all programmers learning new frameworks. Professional developers may face different workflows, collaboration practices, time pressures, and documentation constraints, which should be examined in future studies.

The study did not include a controlled comparison condition or experimental baseline. A direct comparison is not straightforward because the relevant baseline for Dear Diary is not a single tool, but a heterogeneous set of documentation practices that programmers combine differently, such as plain text notes, Notion pages, wikis, code comments, Git commit messages, terminal history, browser bookmarks, and personal recall. For this reason, the semi-structured interviews explicitly examined participants' current habits for tracking their development journey, including note-taking, documenting important information and solutions, and strategies for recalling past solutions. These data allowed us to interpret participants' perceptions of Dear Diary in relation to their existing practices, and several participants contrasted Dear Diary with external files, code comments, terminal history, and Git-based workflows. Thus, the present study provides a situated qualitative comparison with existing practices, but it does not determine whether Dear Diary provides measurable benefits over those practices in terms of effectiveness, efficiency, or long-term reuse. Controlled comparative studies are needed to quantify relative benefits, costs, and trade-offs.

6.3. Future work

Future work will build on the quantitative and qualitative findings from the lab study, emphasizing participants' insights and suggestions. We organize these directions around three areas: evaluating Dear Diary beyond the initial capture phase, exploring new use contexts, and extending the tool's functionality.

First, future work should evaluate Dear Diary longitudinally and comparatively, beyond the initial capture phase studied here. A longer deployment would make it possible to examine whether programmers return to previous snapshots, retrieve and reuse captured explanations, and benefit from the accumulated record of errors, decisions, terminal commands, dependencies, and implementation alternatives over time. Such studies should also examine Dear Diary in relation to the heterogeneous documentation practices that programmers already combine, such as plain text notes, wikis, Notion pages, code comments, Git commit messages, terminal history, browser bookmarks, and personal recall. This would make it possible to assess how Dear Diary complements, reorganizes, or replaces parts of existing documentation workflows across different settings. Future studies should also examine Dear Diary with more diverse populations, including professional developers learning unfamiliar frameworks in industry settings, to understand how the tool fits different workflows, collaboration practices, time pressures, and documentation constraints.

Second, future work should explore new use contexts for Dear Diary. One promising direction, suggested by participants, is the use of Dear Diary during live programming sessions. In this scenario, project elements could be annotated with the professor's explanations, advice, considerations, and reminders captured during the lesson. Participants observed that valuable insights, which could help students better understand the code and underlying concepts, often get lost in the flow of a live session. They also mentioned struggling to balance attention between the professor's explanations and copying the code onto their laptops. As a result, they often wrote quickly without fully grasping the steps or reasoning behind each development task.

In light of this situation, and considering that in the course followed by the participants in our study, both the professor and students use the same IDE (Visual Studio Code), we could develop a mechanism to periodically synchronize the professor's project to each student's computer at the professor's discretion (in the form of Dear Diary snapshots) during live programming sessions. By doing so, students could focus more on the professor's explanations rather than on copying the code, and most importantly, they could rely on Dear Diary's features to associate relevant information with the project's elements and the development activities. Naturally, in the design of this approach, we will need to balance the learning advantages of manually copying the professor's code and performing development activities on students' laptops with the alternative of sharing resources in real-time with the students. The assessment of this approach will be conducted through a large-scale field study involving students and professors during an introductory web development course (different from that followed by the lab study participants).

Finally, improvements to Dear Diary should focus on extending its functionality, both through participant-suggested features and through AI-assisted support. Future work should explore human-in-the-loop AI support for Dear Diary, for example by suggesting tags, summarizing terminal outputs, identifying relevant errors, or drafting initial descriptions, while preserving programmers' agency to decide what is meaningful to capture and to describe their development journey in their own terms.

The new features envisioned by participants that we plan to implement include the ability to **capture and track tasks (TODOs)**, which participants spontaneously used Dear Diary for. Each task can be associated with various elements, allowing programmers to track their progress and, once completed, indicate how the task relates to code snippets, files, terminal commands, and dependencies. In **P3** words, such functionality would be "(…) like a tracker of what I've done and what I need to do; exactly like a bookmark." Another feature participants considered necessary to add relates to the ability to **perform global searches**: cross-sectional searches allowing resources within all snapshots to be located from a unified point. This way, notes, descriptions, and tags would not be scattered within each snapshot but shared across the entire project. As stated by **P18**: "I think it would be useful if it could also search across other snapshots because I might not remember in which snapshot I did something or left a comment."

Improvements to existing features include the ability to **reuse tags**, allowing the system to suggest autocompletion based on tags used in the past as the programmer types. This would enable users to quickly select previously used tags, ensuring more consistent categorization and enhancing the effectiveness of the search feature. Additionally, we plan to incorporate the ability to **format descriptions and notes with basic options** such as bullet points, bold text, and special formatting for links. This enhancement addresses the issue that extensive annotations made by participants can be difficult to interpret when presented only in plain text. Finally, participants suggested two usability enhancements: **adding colors to tags** to provide visual cues for easier identification and use, and **introducing shortcuts** to enable quicker access to Dear Diary features.

7. Conclusion

This paper introduces Dear Diary, a Visual Studio Code extension designed to support programmers in tracking their development journey by providing self-explanatory insights related to code snippets, files, executed terminal commands, and dependencies. These insights account for their decision-making process, encountered errors, consulted sources, exploratory details, and noteworthy advice. After conducting a controlled user study with 18 participants, in which they used Dear Diary during a realistic framework-learning task, quantitative and qualitative results provide initial evidence that participants were able to use the tool to capture diverse aspects of their development activity, including notes related to code snippets, files, terminal commands, dependencies, errors, consulted sources, exploratory questions, reminders, and TODOs.

Participants repeatedly used the features offered by the tool throughout the assigned programming task, suggesting that Dear Diary can be incorporated into ongoing development activity without necessarily postponing documentation until the end of the task. They also reported positive perceptions regarding the tool's usability, relevance, and potential usefulness, and identified additional scenarios in which Dear Diary could be valuable. These findings support the value of Dear Diary's capture-oriented design, while future longitudinal work is needed to evaluate how developers revisit and reuse these records over extended development processes. Future work will also focus on leveraging the collaborative potential identified by participants and assessing the tool's applicability in academic settings, specifically in programming-related courses during live programming sessions conducted by the professor.

CRediT authorship contribution statement

Juan Pablo Sáenz: Writing – original draft, Validation, Supervision, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Luigi De Russis:** Writing – review & editing, Validation, Supervision, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary data to this article can be found online at doi: 10.1016/j.ijhcs.2026.103943.

Data availability

The data that have been used are confidential.

References

Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., 2019. Software documentation issues unveiled. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), pp. 1199–1210, https://doi.org/10.1109/ICSE.2019.00122

Aghajani, E., Bavota, G., Linares-Vásquez, M., Lanza, M., 2021. Automated documentation of android apps. IEEE Trans. Softw. Eng. 47 (1), 204–220. https://doi.org/10.1109/TSE.2018.2890652

Al Safwan, K., Elarnaoty, M., Servant, F., 2022. Developers' need for the rationale of code commits: an in-breadth and in-depth study. J. Syst. Softw. 189, 111320. https://doi.org/10.1016/j.jss.2022.111320

Alexeeva, Z., Perez-Palacin, D., Mirandola, R., 2016. Design decision documentation: a literature overview. In: Tekinerdogan, B., Zdun, U., Babar, A. (Eds.), Software Architecture. Springer International Publishing, Cham, pp. 84–101.

Alkadhi, R., Johanssen, J.O., Guzman, E., Bruegge, B., 2017. REACT: an approach for capturing rationale in chat messages. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 175–180, https://doi.org/10.1109/ESEM.2017.26

Allen, J., Kelleher, C., 2024. Exploring the impacts of semi-automated storytelling on programmers' comprehension of software histories. In: 2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 148–162, https://doi.org/10.1109/VL/HCC60511.2024.00025

Braun, V., Clarke, V., 2006. Using thematic analysis in psychology. Qual. Res. Psychol. 3 (2), 77–101. https://doi.org/10.1191/1478088706qp063oa

Braun, V., Clarke, V., 2019. Reflecting on reflexive thematic analysis. Qual. Res. Sport Exerc. Health 11 (4), 589–597. https://doi.org/10.1080/2159676X.2019.1628806

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A., 2016. 10 years of software architecture knowledge management. J. Syst. Softw. 116 (C), 191–205. https://doi.org/10.1016/j.jss.2015.08.054

Ciancarini, P., Farina, M., Okonicha, O., Smirnova, M., Succi, G., 2023. Software as storytelling: a systematic literature review. Comput. Sci. Rev. 47, 100517. https://doi.org/10.1016/j.cosrev.2022.100517

Clements, P., Garlan, D., Little, R., Nord, R., Stafford, J., 2003. Documenting software architectures: views and beyond. In: 25th International Conference on Software Engineering, 2003. Proceedings, pp. 740–741, https://doi.org/10.1109/ICSE.2003.1201264

Dagenais, B., Robillard, M.P., 2010. Creating and evolving developer documentation: understanding the decisions of open source contributors. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE '10. Association for Computing Machinery, New York, NY, USA, pp. 127–136, https://doi.org/10.1145/1882291.1882312.

Fast, E., Steffee, D., Wang, L., Brandt, J.R., Bernstein, M.S., 2014. Emergent, crowd-scale programming practice in the IDE. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '14. Association for Computing Machinery, New York, NY, USA, pp. 2491–2500, https://doi.org/10.1145/2556288.2556998

Guo, P., 2017. Building tools to help students learn to program. Commun. ACM 60 (12), 8–9. https://doi.org/10.1145/3148245

Happel, H.-J., Maalej, W., 2009. From work to word: how do software developers describe their work? In: 2009 6th IEEE International Working Conference on Mining Software Repositories. MSR 2009. IEEE Computer Society, Los Alamitos, CA, USA, pp. 121–130, https://doi.org/10.1109/MSR.2009.5069490

Hartmann, B., MacDougall, D., Brandt, J., Klemmer, S.R., 2010. What would other programmers do: suggesting solutions to error messages. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10. Association for Computing Machinery, New York, NY, USA, pp. 1019–1028, https://doi.org/10.1145/1753326.1753478.

Head, A., Appachu, C., Hearst, M.A., Hartmann, B., 2015. Tutorons: generating context-relevant, on-demand explanations and demonstrations of online code. In: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 3–12, https://doi.org/10.1109/VLHCC.2015.7356972

Head, A., Glassman, E.L., Hartmann, B., Hearst, M.A., 2018a. Interactive extraction of examples from existing code. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18. Association for Computing Machinery, New York, NY, USA, pp. 1–12, https://doi.org/10.1145/3173574.3173659.

Head, A., Sadowski, C., Murphy-Hill, E., Knight, A., 2018b. When not to comment: questions and tradeoffs with API documentation for C++ projects. In: Proceedings of the 40th International Conference on Software Engineering, ICSE '18. Association for Computing Machinery, New York, NY, USA, pp. 643–653, https://doi.org/10.1145/3180155.3180176.

Head, A., Hohman, F., Barik, T., Drucker, S.M., DeLine, R., 2019. Managing messes in computational notebooks. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19. Association for Computing Machinery, New York, NY, USA, pp. 1–12, https://doi.org/10.1145/3290605.3300500.

Hesse, T.-M., Kuehlwein, A., Roehm, T., 2016. DecDoc: a tool for documenting design decisions collaboratively and incrementally. In: 2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH), pp. 30–37, https://doi.org/10.1109/MARCH.2016.9

Horvath, A., Myers, B., Macvean, A., Rahman, I., 2022a. Using annotations for sensemaking about code. In: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology, UIST '22. Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3526113.3545667.

Horvath, A., Liu, M.X., Hendriksen, R., Shannon, C., Paterson, E., Jawad, K., Macvean, A., Myers, B.A., 2022b. Understanding how programmers can use annotations on documentation. In: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, CHI '22. Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3491102.3502095.

Horvath, A., Macvean, A., Myers, B.A., 2023. Support for long-form documentation authoring and maintenance. In: 2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE Computer Society, Los Alamitos, CA, USA, pp. 109–114, https://doi.org/10.1109/VL-HCC57772.2023.00020

Ichinco, M., Kelleher, C., 2015. Exploring novice programmer example use. In: 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 63–71, https://doi.org/10.1109/VLHCC.2015.7357199

Jansen, A., Bosch, J., Avgeriou, P., 2008. Documenting after the fact: recovering architectural design decisions. J. Syst. Softw. 81 (4), 536–557. https://doi.org/10.1016/j.jss.2007.08.025

Kery, M.B., Radensky, M., Arya, M., John, B.E., Myers, B.A., 2018. The story in the notebook: exploratory data science using a literate programming tool. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18. Association for Computing Machinery, New York, NY, USA, pp. 1–11, https://doi.org/10.1145/3173574.3173748.

Kiger, M.E., Varpio, L., 2020. Thematic analysis of qualitative data: AMEE guide No. 131. Med. Teach. 42 (8), 846–854. https://doi.org/10.1080/0142159X.2020.1755030

Ko, A.J., Myers, B.A., Coblenz, M.J., Aung, H.H., 2006. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Trans. Softw. Eng. 32 (12), 971–987. https://doi.org/10.1109/TSE.2006.116

LaToza, T.D., 2020. Information needs: lessons for programming tools. IEEE Softw. 37 (6), 52–57. https://doi.org/10.1109/MS.2020.3014343

Le, T.-D.B., Linares-Vasquez, M., Lo, D., Poshyvanyk, D., 2015. RCLinker: automated linking of issue reports and commits leveraging rich contextual information. In: 2015 IEEE 23rd International Conference on Program Comprehension, pp. 36–47, https://doi.org/10.1109/ICPC.2015.13

LeClair, A., Jiang, S., McMillan, C., 2019. A neural model for generating natural language summaries of program subroutines. In: Proceedings of the 41st International Conference on Software Engineering, ICSE '19. IEEE Press, pp. 795–806, https://doi.org/10.1109/ICSE.2019.00087.

Lethbridge, T., Singer, J., Forward, A., 2003. How software engineers use documentation: the state of the practice. IEEE Softw. 20 (6), 35–39. https://doi.org/10.1109/MS.2003.1241364

Li, J., Ahmed, I., 2023. Commit message matters: investigating impact and evolution of commit message quality. In: Proceedings of the 45th International Conference on Software Engineering, ICSE '23. IEEE Press, pp. 806–817, https://doi.org/10.1109/ICSE48619.2023.00076.

Liang, J.T., Arab, M., Ko, M., Ko, A.J., LaToza, T.D., 2023. A qualitative study on the implementation design decisions of developers. In: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 435–447, https://doi.org/10.1109/ICSE48619.2023.00047

Liu, M.X., Hsieh, J., Hahn, N., Zhou, A., Deng, E., Burley, S., Taylor, C., Kittur, A., Myers, B.A., 2019. Unakite: scaffolding developers' decision-making using the web. In: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST '19. Association for Computing Machinery, New York, NY, USA, pp. 67–80, https://doi.org/10.1145/3332165.3347908.

Liu, M.X., Kittur, A., Myers, B.A., Apr 2021. To reuse or not to reuse? A framework and system for evaluating summarized knowledge. Proc. ACM Hum.-Comput. Interact. 5 (CSCW1), https://doi.org/10.1145/3449240

Maalej, W., Tiarks, R., Roehm, T., Koschke, R., Sep 2014. On the comprehension of program comprehension. ACM Trans. Softw. Eng. Methodol. 23 (4), https://doi.org/10.1145/2622669

Mahoney, M., 2023. Storyteller: guiding students through code examples. In: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023. Association for Computing Machinery, New York, NY, USA, pp. 1131–1135, https://doi.org/10.1145/3545945.3569843.

Margulieux, L., Denny, P., Cunningham, K., Deutsch, M., Shapiro, B.R., 2021. When wrong is right: the instructional power of multiple conceptions. In: Proceedings of the 17th ACM Conference on International Computing Education Research, ICER 2021. Association for Computing Machinery, New York, NY, USA, pp. 184–197, https://doi.org/10.1145/3446871.3469750.

McBurney, P.W., McMillan, C., 2016. Automatic source code summarization of context for Java methods. IEEE Trans. Softw. Eng. 42 (2), 103–119. https://doi.org/10.1109/TSE.2015.2465386

Mehrpour, S., Latoza, T.D., Sep 2023. A survey of tool support for working with design decisions in code. ACM Comput. Surv. 56 (2), https://doi.org/10.1145/3607868

Mehrpour, S., LaToza, T.D., Kindi, R.K., 2019. Active documentation: helping developers follow design decisions. In: 2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 87–96, https://doi.org/10.1109/VLHCC.2019.8818816

Meyer, A.N., Satterfield, C., Züger, M., Kevic, K., Murphy, G.C., Zimmermann, T., Fritz, T., 2022. Detecting developers' task switches and types. IEEE Trans. Softw. Eng. 48 (1), 225–240. https://doi.org/10.1109/TSE.2020.2984086

Mysore, A., Guo, P.J., 2018. Porta: profiling software tutorials using operating-system-wide activity tracing. In: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, UIST '18. Association for Computing Machinery, New York, NY, USA, pp. 201–212, https://doi.org/10.1145/3242587.3242633.

Nasehi, S.M., Sillito, J., Maurer, F., Burns, C., 2012. What makes a good code example?: A study of programming Q amp;a in StackOverflow. In: 2012 28th IEEE International Conference on Software Maintenance (ICSM), pp. 25–34, https://doi.org/10.1109/ICSM.2012.6405249

Nassif, M., Robillard, M.P., 2017. Revisiting turnover-induced knowledge loss in software projects. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE Computer Society, Los Alamitos, CA, USA, pp. 261–272, https://doi.org/10.1109/ICSME.2017.64

Oney, S., Brandt, J., 2012. Codelets: linking interactive documentation and example code in the editor. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '12. Association for Computing Machinery, New York, NY, USA, pp. 2697–2706, https://doi.org/10.1145/2207676.2208664.

Parnin, C., DeLine, R., 2010. Evaluating cues for resuming interrupted programming tasks. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10. Association for Computing Machinery, New York, NY, USA, pp. 93–102, https://doi.org/10.1145/1753326.1753342.

Parnin, C., Rugaber, S., 2011. Resumption strategies for interrupted programming tasks. Softw. Qual. J. 19 (1), 5–34. https://doi.org/10.1007/s11219-010-9104-9

Pimentel, J.A.F., Murta, L., Braganholo, V., Freire, J., 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In: Proceedings of the 16th International Conference on Mining Software Repositories, MSR '19. IEEE Press, pp. 507–517, https://doi.org/10.1109/MSR.2019.00077.

Raglianti, M., 2022. Topology of the documentation landscape. In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 297–299, https://doi.org/10.1145/3510454.3517068

Rai, S., Belwal, R.C., Gupta, A., Feb 2022. A review on source code documentation. ACM Trans. Intell. Syst. Technol.Just Accepted (https://doi.org/10.1145/3519312

Robillard, M.P., Marcus, A., Treude, C., Bavota, G., Chaparro, O., Ernst, N., Gerosa, M.A., Godfrey, M., Lanza, M., Linares-Vásquez, M., Murphy, G.C., Moreno, L., Shepherd, D., Wong, E., 2017. On-demand developer documentation. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 479–483, https://doi.org/10.1109/ICSME.2017.17

Rule, A., Tabard, A., Hollan, J.D., 2018. Exploration and explanation in computational notebooks. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18. Association for Computing Machinery, New York, NY, USA, pp. 1–12, https://doi.org/10.1145/3173574.3173606.

Storey, M.-A., Ryall, J., Bull, R.I., Myers, D., Singer, J., 2008. TODO or to bug: exploring how task annotations play a role in the work practices of software developers. In: Proceedings of the 30th International Conference on Software Engineering, ICSE '08. Association for Computing Machinery, New York, NY, USA, pp. 251–260, https://doi.org/10.1145/1368088.1368123.

Storey, M.-A., Ryall, J., Singer, J., Myers, D., Cheng, L.-T., Muller, M., 2009. How software developers use tagging to support reminding and refinding. IEEE Trans. Softw. Eng. 35 (4), 470–483. https://doi.org/10.1109/TSE.2009.15

Storey, M.-A., Zagalsky, A., Filho, F.F., Singer, L., German, D.M., 2017. How social and communication channels shape and challenge a participatory culture in software development. IEEE Trans. Softw. Eng. 43 (2), 185–204. https://doi.org/10.1109/TSE.2016.2584053

Sulír, M., Nosál, M., Porubän, J., 2016. Recording concerns in source code using annotations. Comput. Lang. Syst. Struct. 46, 44–65. https://doi.org/10.1016/j.cl.2016.07.003

Tantisuwankul, J., Nugroho, Y.S., Kula, R.G., Hata, H., Rungsawang, A., Leelaprute, P., Matsumoto, K., 2019. A topological analysis of communication channels for knowledge sharing in contemporary GitHub projects. J. Syst. Softw. 158, 110416. https://doi.org/10.1016/j.jss.2019.110416

Tian, Y., Zhang, Y., Stol, K.-J., Jiang, L., Liu, H., 2022. What makes a good commit message? In: Proceedings of the 44th International Conference on Software Engineering, ICSE '22. Association for Computing Machinery, New York, NY, USA, pp. 2389–2401, https://doi.org/10.1145/3510003.3510205.

van der Ven, J.S., Jansen, A.G.J., Nijhuis, J.A.G., Bosch, J., 2006. Design Decisions: The Bridge Between Rationale and Architecture. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 329–348. https://doi.org/10.1007/978-3-540-30998-7_16

Vassallo, C., Panichella, S., Di Penta, M., Canfora, G., 2014. CODES: mining source code descriptions from developers discussions. In: Proceedings of the 22nd International Conference on Program Comprehension, ICPC 2014. Association for Computing Machinery, New York, NY, USA, pp. 106–109, https://doi.org/10.1145/2597008.2597799.

Vieira, C., Magana, A.J., Falk, M.L., Garcia, R.E., Aug 2017. Writing in-code comments to self-explain in computational science and engineering education. ACM Trans. Comput. Educ. 17 (4), https://doi.org/10.1145/3058751

Wang, A.Y., Head, A., Zhang, A.G., Oney, S., Brooks, C., 2023. Colaroid: a literate programming approach for authoring explorable multi-stage tutorials. In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI '23. Association for Computing Machinery, New York, NY, USA, https://doi.org/10.1145/3544548.3581525.

Wang, G., Sun, Z., Dong, J., Zhang, Y., Zhu, M., Liang, Q., Hao, D., Sep 2024a. Is it hard to generate holistic commit message? ACM Trans. Softw. Eng. Methodol.Just Accepted (https://doi.org/10.1145/3695996

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

Wu, S., Zhou, Y., Wang, X., 2021. Exploring User Experience of Automatic Documentation Tools. Association for Computing Machinery, New York, NY, USA.

Wuilmart, P., Söderberg, E., Höst, M., 2023. Programmer stories, stories for programmers: exploring storytelling in software development. In: Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming, Programming '23. Association for Computing Machinery, New York, NY, USA, pp. 68–75, https://doi.org/10.1145/3594671.3594677

Zaychik, V., Regli, W.C., 2003. Capturing communication and context in the software project lifecycle. Res. Eng. Des. 14 (2), 75–88. https://doi.org/10.1007/s00163-002-0027-8

Zhang, J., Wang, X., Zhang, H., Sun, H., Liu, X., 2020. Retrieval-based neural source code summarization. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20. Association for Computing Machinery, New York, NY, USA, pp. 1385–1397, https://doi.org/10.1145/3377811.3380383.

Zhi, J., Garousi-Yusifoğlu, V., Sun, B., Garousi, G., Shahnewaz, S., Ruhe, G., 2015. Cost, benefits and quality of software development documentation: a systematic mapping. J. Syst. Softw. 99, 175–198. https://doi.org/10.1016/j.jss.2014.09.042