HowToo: A Platform for Sharing, Finding, and Using Programming Strategies

Maryam Arab[1], Jenny Liang[2*], Yang Yoo[1*], Amy J. Ko[2], Thomas D LaToza[1]
[1] *Department of Computer Science, George Mason University, Fairfax, VA, USA, {marab, yyoo4, tlatoza}@gmu.edu*
[2] *The Information School, University of Washington, Seattle, WA, USA, {jliang9, ajko}@uw.edu*
[*]Note: These authors contributed equally.

***Abstract*—Developers rely heavily on resources to find technical insights on how to use languages, APIs, and platforms, seeking help from Stack Overflow, GitHub, meetups, blogs, live streams, forums, documentation, and more. However, there is one kind of knowledge for which resources are hard to find: strategic knowledge. In contrast to technical knowledge, strategic knowledge provides insight into how to approach problem-solving. Prior work has demonstrated that developers can make use of written strategies to improve their problem-solving. However, there is currently no way for developers to share, curate, and search for this knowledge at scale. To address this gap, we contribute *HowToo*, a platform for sharing, finding, and using programming strategies. Its key insight is that there are many different approaches to the same problem, and developers may need different strategies depending on their situation. In a longitudinal evaluation with more than 30 students in a project-based software engineering course, we found that: 1) students viewed *HowToo* as complementary to technical resources; 2) students viewed strategies as helping them be more systematic and complete in their work; 3) *HowToo* helped students be more confident in their problem solving; 4) when students were under time pressure, they were less inclined to use *HowToo* to structure their work, as being mindful required them to slow down.**

***Index Terms*—Q&A, knowledge sharing, programming strategies**

I. Introduction

Software engineering is a highly distributed and social activity, as developers share knowledge on online communication channels such as social media and online chats to solve their programming problems [1]–[6]. They ask and answer questions on Stack Overflow [7], [8], meet through meetups, learn from tech talks, and participate in workshops [4], [5] to gain expertise. Stack Overflow, in particular, has become a primary resource for accessing technical knowledge about programming languages, APIs, and platforms [7]. Many of these shifts to social stem from an underlying need to seek expert knowledge [7], [8].

While these sources excel at answering questions about technical artifacts—explaining the semantics of a programming language construct, sharing design patterns for a library, revealing hidden side effects from a configuration setting in a web service —they do not answer every question. In particular, developers often need answers to "how to" questions: how to debug, how to test, how to design a data structure, and how to refactor code without introducing defects. However, these kinds of *problem-solving* knowledge are rarely shared online or taught in formal CS education contexts. Instead, developers are left to develop their approaches independently, often resulting in ineffective practices [9].

One way of capturing and sharing problem-solving knowledge is through *programming strategies*, higher-level software engineering techniques about how to approach solving a problem, described as a sequence of steps to reach a goal. Steps may encompass acting, checking a condition, or collecting requirements and information [10]–[12]. Figure 1(I) lists a strategy for debugging a defect in an HTML page. The strategy guides the developer to use the Chrome inspector to understand how the browser interpreted the HTML to infer the problem. Strategic knowledge is a central part of expertise [13]–[15]. Prior work has demonstrated that developers can make use of programming strategies to improve their problem-solving [10].

While prior work suggests the potential of explicit programming strategies to improve how developers solve problems, it is unclear how to share and use them at scale. If developers share strategies online, how should they be represented so that other developers can search, browse for, and recognize them as relevant? When developers use other developers' strategies, how should they clarify ambiguities and provide feedback? If there were a way to find and use strategies, would developers leverage them to structure their work or follow their existing habits?

We designed HowToo, a platform for sharing, finding, and using strategic knowledge from experienced developers to answer these questions. HowToo contributes a novel combination of features for a poorly supported aspect of programming. This includes a novel representation of programming strategy metadata for searching and browsing, an interface for using and executing strategies step by step, and interactions that facilitate strategy refinement based on developer feedback (Figure 1). Together, these offer a comprehensive solution to sharing hard-won software engineering expertise with the world.

To evaluate HowToo, we conducted a formative evaluation in an undergraduate software engineering course. This study is the first to test programming strategies in a real team-based software engineering context rather than a lab. The study contained two phases: a training phase that required using strategies in HowToo to familiarize participants with them, and a testing phase where participants chose whether to

978-1-6654-4592-4/21/$31.00 ©2021 IEEE

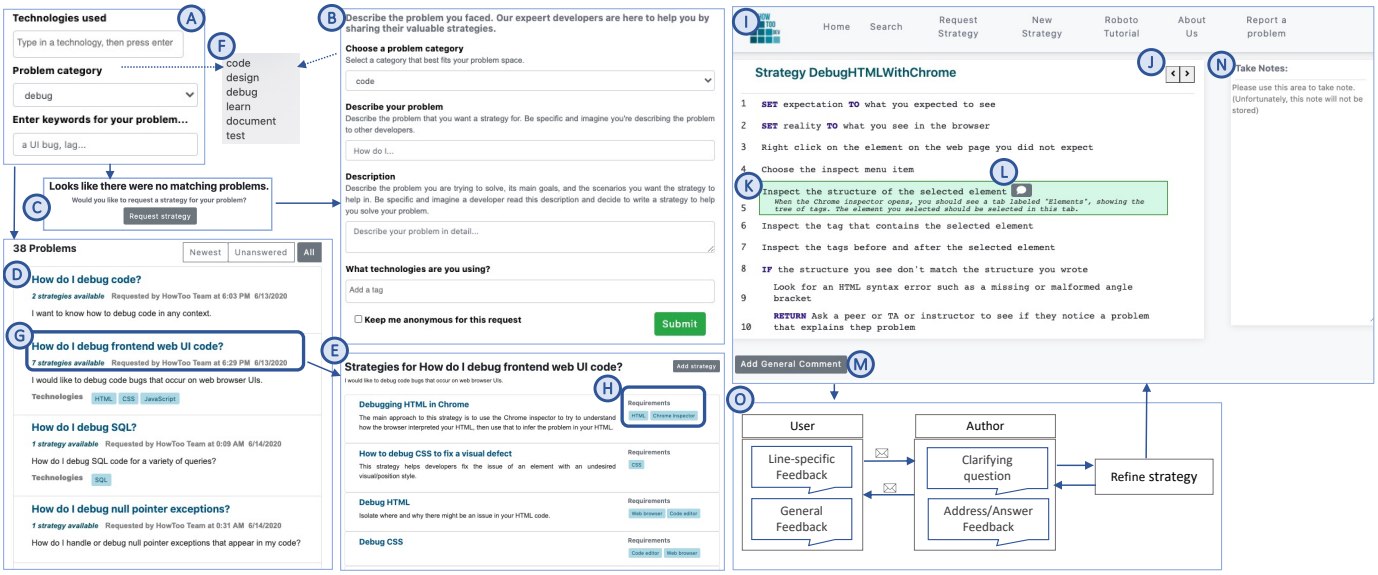

Fig. 1: HowToo enables (A, C, D, G, E) searching for, (B) requesting, authoring, (I, J, K, N) using, commenting on (L, M) and (O) refining strategies. Arrows indicate transitions between system states.

use strategies in HowToo. We collected data through student interviews, comments about using strategies during the testing phase, logs of HowToo interactions, and a survey conducted at the end of the semester.

The results revealed that 60% of participants continued using the platform when it was not mandatory. Students found HowToo to be complementary to other technical resources such as Stack Overflow. They viewed strategies as offering a checklist of tasks to complete, which helped them be more confident about their internal knowledge while filling the gaps in their current practices. Some students found that strategies needed more details, descriptions, and clarity to be helpful. Those who requested strategies from their peers on HowToo believed that HowToo was the best place to find people with similar problems. Finally, some students reported that more collaboration and communication would make the platform a better place to learn.

II. Background

A. Explicit Programming Strategies

Many industries share strategic knowledge in the form of standard operating procedures (SOPs) [16], and checklists [17]. In safety-critical domains, such as operating a nuclear power plant, SOPs provide step-by-step instructions which promote efficiency, error prevention, and regulation compliance. Checklists are commonly used in healthcare settings to reduce errors and standardize medical procedures [17].

Similarly, software engineering is not just about solving a specific problem—it is also about learning when and how to apply strategies to solve a problem [18]. Prior work shows that metacognition and problem solving play a key role in improving the productivity, independence, and self-efficacy of developers [10], [19]–[21]. Engaging with metacognitive behaviors, such as reflection and comprehension monitoring, is correlated with success in programming courses [21]. Programming strategies constitute a central component of programming expertise [13]–[15].

Explicit programming strategies are human-executable procedures for accomplishing a task [10]. Strategies enumerate a series of steps to accomplish a goal and identify actions to overcome challenges. Following an expert developer's explicit programming strategy can increase task completion rates while also enabling developers to work more systematically [9], [10]. Prior work has shown that explicit programming strategies reduce comprehension errors [12], improve debugging success [22], and reduce debugging time [11], [23].

B. Knowledge Sharing Platforms in Software Engineering

Question and answer (Q&A) platforms such as Stack Overflow can offer many benefits to software engineering problem solving by crowdsourcing answers to questions about how to use programming languages, APIs, and platforms [7], [8], [18]. Using Q&A platforms is positively associated with developer productivity [24], [25]. However, prior work has shown that Stack Overflow questions occasionally go beyond strictly technical knowledge, offering opinions, reviews, and other content [26], [27]. Similar Q&A platforms exist for student developers, such as Piazza [28], where students are often encouraged to ask questions that reflect their reasoning and problem-solving processes [29].

One benefit of Q&A platforms is the code examples they contain [30], which enable developers to copy and adapt solutions opportunistically [31]. While this can reduce development time, it introduces other issues, such as propagating vulnerable [32] and defective code [33], [34], as well as not correctly attributing source code to authors [35]. Code examples often lack essential contexts, such as background

knowledge, the rationale for the solution, or step-by-step instructions for arriving at similar or related solutions [18].

Thus, while Q&A platforms can help answer many technical questions about specific languages, APIs, and platforms, they rarely offer strategies for solving software engineering problems with unknown answers.

III. Motivating Example

To illustrate how HowToo facilitates developers in sharing, finding, and using strategies, consider a fictional scenario. Alice is implementing a web page, and the page does not render as expected. She visits HowToo to search for a strategy to help (Figure 1), inputting the technologies she is using alongside a few keywords describing her problem (A). She then selects "debug" for the problem category (F). HowToo finds no problems matching her query (C). Before requesting a strategy for a new problem (B), Alice decides to edit her search criteria. She edits the keyword and technology specifiers and reruns her query. HowToo lists all problems in the debugging category defined or requested by other developers (D). She finds the second problem in the results relevant to her problem, with seven strategies defined (G). Checking the list of existing strategies (E), Alice notices that each lists its usage requirements (H). Checking these requirements and the description of each, she decides to try "Debugging HTML in Chrome". Alice starts using the strategy (I). HowToo supports her in systematically following the strategy, helping her keep track of her place by highlighting her current step in the strategy (K), switch steps using arrow keys or buttons (J), and record information she discovers and wants to remember for later using a side-panel (N). Alice finds the usage of the word "Inspect" on line 5 in the strategy unclear. As her confusion is related to a specific line rather than the strategy as a whole, she clicks the icon at the end of line 5 (L) to create a comment, asking: "What does 'inspect' mean? How should I inspect the HTML element?" HowToo publishes her comment and notifies the strategy author via email. The author then decides to address the comment, adding additional detail to the line. They then respond to the comment, describing the changes made. HowToo notifies Alice that she received a response to her comment.

Facing a second problem, Alice does not find the existing strategies helpful. She navigates to the "Request Strategy" tab (I), describing her problem and requesting a new strategy (B). Her request is added to the list of new requests shown publicly under "Unanswered" problems (D). Developers with strategies to address this problem may respond to the open request and create a new strategy (Figure 2).

IV. Sharing Strategies with HowToo

The promise of explicit programming strategies [10], combined with the lack of support for sharing strategies in modern Q&A sites, creates a critical gap in software engineering resources: how can developers share and reuse strategic knowledge at scale? HowToo answers this question. HowToo supports the accumulation and refinement of strategic software engineering expertise in a central platform to help improve and add to the collective knowledge of a developer community over time. Developers first request a strategy by describing their problem (Section IV-A). Strategy authors may then respond to an open request by creating a strategy in an editor using a specialized strategy description language (Section IV-B). Developers search for a strategy to address a problem they face, using their understanding of their problem to identify potentially relevant strategies (Section IV-C). Developers then follow the strategy, using the environment to keep track of their place and the information they collect. When the developer finds the strategy confusing, incomplete, or unhelpful, they may offer feedback or ask clarifying questions, which prompts the strategy author to improve and refine their strategy (Section IV-E).

A. Requesting strategies

When a developer cannot find an existing strategy relevant to their problem, they may request a new one by describing the problem they face. Developers describe their problem by recording metadata, including a title, description of the problem, goals, technologies being used, and a problem category. The optional technology metadata is helpful when the problem is tied closely to a specific technology. The problem categories are derived from a taxonomy of information needs [36], refined through several rounds of piloting and iteration. They include *testing*, *debugging*, *learning*, *documenting*, *designing*, and *coding* (Figure 1-F). New strategy requests are posted to a list of unanswered requests shown to all HowToo users, prompting developers with the relevant expertise to write a strategy. Strategy requests may generate multiple strategies for the same problem, offering different approaches or supporting different levels of developer prior knowledge.

B. Authoring strategies

There are many possible ways to encode strategic programming knowledge, such as unstructured natural language or natural language with hierarchical bulleted lists. HowToo uses a more structured representation, Roboto [10], which is primarily natural language but includes simple control flow constructs, such as conditionals and loops, to enable strategy users to be more systematic and comprehensive. Figure 2 shows the HowToo strategy authoring interface. HowToo provides guidelines on how to use Roboto to define strategies (C) and supports mimicking Roboto syntax through a sample Roboto strategy (D). An editor for writing strategies (B) helps developers learn Roboto syntax, offering meaningful syntax errors.

One challenge in authoring an effective strategy is to generalize it to support users with diverse expertise. To address this challenge, authors are provided guidelines for writing effective strategies (Figure 2-C). For example, HowToo suggests that strategy authors separate the details from the main strategy steps, moving them to descriptive text for only those who might need them. HowToo supports *hyperlinks* for including links to tutorials and supporting references in the strategy.

Strategy authors are encouraged to provide strategy metadata (Figure 2-A), helping strategy users more effectively assess relevance in browsing and searching.

C. Searching for strategies

Unlike searching for answers to questions about using a programming language, API, or platform, which often benefit from unique, pre-defined identifiers, there is no standardized terminology for referring to software engineering problems. Strategy seekers must be aware of what their problem is and translate this into keywords to search. For example, imagine someone searching for how to debug a defect involving multiple threads: a strategy seeker using a Q&A system might search for "debug threads", but a strategy author might have labeled a strategy "diagnose concurrency defects".

To address this problem, HowToo does not rely exclusively on search. Instead, it uses a standardized vocabulary of software engineering activities and a combination of search, browsing, and filtering. This first helps developers find the *problem* for which they need help, and then have them select from one or more strategies that may help solve this problem. Figure 1 illustrates this process.

While searching for a programming strategy, there may be multiple similar yet distinct candidate problems that match a developer's query. For example, there are many strategies to debug a defect [10]. Further, any given problem may potentially have many viable strategies. This requires developers to decide which problems, as well as strategies, are relevant to their situation. Previous work shows that users struggle with recalling information over recognizing information [37]. To support a developer's recognition for potential problems and strategies, we designed a rich metadata structure for strategies and problems to display to the developer.

To implement the search functionality, we use a natural language query to retrieve relevant problems for developers. We expect that the problems expressed in the search query will be similar to the titles of the problems on our platform (e.g., the query "debugging front-end" will be similar to the problem of "How do I debug front-end web UI code?"). Our natural language search relies on fuzzy string searching for matching queries with problem metadata [38].

To reduce the number of results displayed, developers can refine their query by filtering problems by problem category or technologies. For instance, in the list of strategies in Figure 1-E, each strategy is tagged with a set of technologies required for the strategy to be effective (H). The developer may define a problem category and technologies with which they are familiar (Fig. 1-A) to find both related problems and strategies (Figs. 1-D, E).

D. Using strategies

Strategies teach software engineering techniques with specific, prescriptive advice that describe concrete actions a developer should take in a specific order [10]–[12]. As humans have limited working memory that might lead to human error when there are too many items to keep track of [39], developers may forget the order of performing actions, lose track of the steps they are performing, or forget variable values and information in the previous step [10].

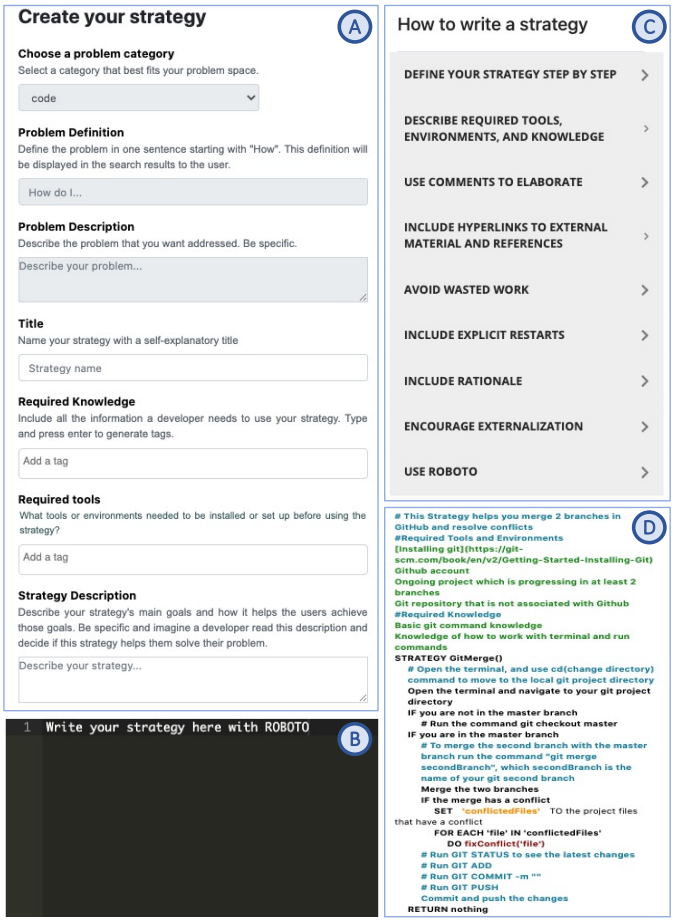

Fig. 2: Developers author strategies in HowToo by (A) writing metadata and (B) using an editor to write a strategy in a strategy description language. To support this process, developers may consult (C) guidelines for authoring effective strategies and (D) a sample strategy illustrating the language syntax.

To help developers avoid these working memory mistakes, HowToo offers two ways of externalizing the problem-solving state. Users can control a program counter with keyboard arrow keys and the "Next" and "Previous" buttons (Fig. 1-J). A strategy statement's description and line-specific comment button are only shown when it is highlighted (Fig. 1-K, L). To offload the burden of remembering information, strategy users may take notes on the information they may need for later, such as variable values, in a dedicated panel (Figs. 1-N). Notes are not line-specific, allowing developers to access the recorded information at any step.

E. Refining strategies

Authoring a strategy poses many challenges, such as choosing the target level of knowledge for strategy users and the level of detail. Understanding potential strategy users' expectations and needs could help strategy authors articulate

a better strategy that is more usable for varying strategy users. There are many ways to address this problem, such as encouraging strategy authors to write additional details and providing strategy writing guidelines. However, this does not guarantee that enough detail is provided for a broad set of strategy users. One approach is to treat strategies as a shared document on which users can give feedback and suggestions. This allows strategy authors to respond to feedback and improve the strategy over time.

To support feedback loops, HowToo allows strategy users to post threaded comments on both individual lines of a strategy or across the entire strategy (Figure 1-L,M). Figure 1-O depicts the HowToo commenting workflow. When a user posts a comment, HowToo sends an email notification to the strategy author. Based on the comment, the author may respond to the question, ask for clarification, or directly refine the strategy. HowToo then sends a notification email to the user regarding their comment. The user then may address the author's response. This communication cycle between the strategy author and user may take several iterations while both parties clarify, refine, and improve the strategy.

V. User Study

In designing HowToo, we had two formative questions about the platform's ability to support sharing strategies:

- RQ1: How do students experience HowToo when using it to support their programming?
- RQ2: How does HowToo benefit students?

To investigate these questions, we deployed HowToo in a senior elective project-based software engineering course. The course taught software engineering foundations while applying this knowledge in team projects. The class consisted of 35 students that were divided into nine teams (T1-T9), each with a project manager, a designer, and one or two developers. The teams submitted weekly assignments to scaffold their progress. The instructor authored seven strategies in HowToo related to the assignments: setting up a GitHub repository, clarifying design specifications, extracting requirements, designing an architecture, translating requirements to testing plans, performing black-box testing, and triaging issues. The fourth author was the instructor. In total, the platform had 43 strategies at launch. During the second phase of the study, three strategies were authored by students in response to peer requests. An Institutional Review Board approved the study design.

The instructor encouraged students to contribute to HowToo for extra credit by answering peer questions in the comments, responding to strategy requests by authoring strategies, or sharing their own internal strategies. To understand how students interacted with HowToo, we logged interactions on the platform. Throughout the course, the instructor described HowToo as an independent service adopted to support the class to avoid participant response bias [40], and encouraged students to give critical feedback about the platform to help the instructor decide whether to continue using it.

All students were 18 years or older and had completed at least three programming courses. Students who were interviewed received $15 Amazon gift card(s) for each interview session in compensation for their time.

A. Method

The study consisted of a training phase and a testing phase. During the training phase in the first five weeks of the course, student teams were required to use a related strategy for each weekly assignment. The instructor sent a link to a HowToo strategy. Teams were required to use it and reflect on challenges they faced and how they solved them by writing a comment on a line or on the whole strategy.

In the testing phase during the last four weeks of class, there was no obligation to use HowToo strategies. The purpose of this phase was to investigate if students would use strategies without significant external motivators or prefer to use other resources for help. The only incentive provided was a single point of extra credit for posting a strategy that a classmate voluntarily used, which mirrored the kinds of reputational incentives given on similar Q&A platforms. Students were encouraged to use the platform to search for a desired strategy or request one from the instructor, TA, or their peers. The first author invited students to participate in an interview in three cases: 1) to clarify HowToo comments, 2) to study a chain of communication on the platform, and 3) to gather additional context to the logs of certain interactions on the platform, such as authoring a strategy, requesting a new strategy, or searching for and using other strategies. At the end of each phase, the first author invited all students to the interview. All the interviews were conducted via Zoom, while the interviewer video recorded the session. The videos were deleted after the interview transcription.

At the end of the course, the instructor asked students to complete a survey about their experience using HowToo, the approaches they used to solve their problems during the testing phase, what prevented or motivated them to continue using the platform, and alternative approaches they leveraged.

The first author prompted the instructor to write weekly structured diary entries on her experience for each strategy she wrote during the training phase and her observations on how students used each strategy. The diary prompted the instructor to reflect on challenges the strategies helped with, reactions to feedback that students provided, and overall impressions of the strategies' impact on students' problem-solving.

B. Data collection

To answer our research questions, we collected data from five sources: 1) semi-structured interviews; 2) comments/feedback students made about the strategies; 3) the final survey data collected from all 33 students about the strategies they used, the platform itself, and the alternative approaches they took; 4) the logs of user interactions including searching, using, requesting, and authoring on HowToo from 20 students; and 5) the instructor's diary.

We conducted semi-structured interviews using specified questions tailored the interviewee's logs and open-ended questions to expand upon unexpected responses. The interviewers

were encouraged to ask relevant follow-up questions to explore exciting topics. Interviewers gathered the user's comments and interaction logs before the interview.

In total, we conducted five interviews with four of the nine teams. Four interviews focused on the students' experience using strategies in the training phase, while one interview was with a student who requested a strategy and received a response from a peer in the testing phase. For the first four interviews and for each strategy, we asked interviewees to 1) walk us through the steps they took to use each strategy; 2) explain how they solved a challenge they reported while using the strategy, if applicable; 3) describe the benefits and drawbacks of each strategy; and 4) report feedback about the strategies and platform. For the strategy requester interview, the interviewer asked the participant to describe 1) the steps of requesting for and using a strategy, 2) other approaches they tried before requesting a strategy, 3) the motivation to use to HowToo, and 4) what motivated her to request a strategy.

We collected the interview transcriptions, the comments from the training phases, and the final survey responses listed in documents for data analysis and coding. We extracted the number of students who used HowToo for the whole semester and how they used the platform from the interaction logs.

C. Data Analysis

To analyze the data, we followed current best practices in qualitative coding [41]. We did not analyze the participants' ideas as quantified data; instead, we treated the analysis results as claims for future investigation and checked the reliability of our coding by sharing disagreements and resolving them to arrive at an agreement.

Following these guidelines, we created separate documents of the feedback from the training phase, interview transcriptions, and survey responses for qualitative analysis [42]. Three paper authors separately read each document and inductively generated codes in the first round of qualitative coding. The three paper authors separately identified topics related to the research questions by creating codes with a brief description and individually labeled each data record with zero or more codes. To aggregate these codes, the paper authors first compared the separately generated codes to identify those with similar definitions and added them to the codebook under a uniquely labeled code. The three authors compared the codes, discussed instances of disagreement, and reached an agreement by either adding or removing the code from the codebook. Disagreements mainly stemmed from variations in scoping codes rather than the meaning of the statements. The authors agreed on scoping and resolved disagreements in the first round of discussion. During this process, the authors found that all remaining codes conveyed unique ideas participants reported and added them to the codebook. The authors then coded the responses in a second round using the final codebook. They then applied pattern coding to the final codes [43], which groups codes into several broader categories. The result of this process are shown in Table I.

We also collected the instructor's diary and the student's survey responses on their experiences authoring a strategy. Because the instructor was also an author of this paper, our analysis approach was to have the instructor read her reflections for each of the five strategies she authored and summarize the reflections holistically.

D. Results

1) RQ1: How do students experience HowToo when using it to support their programming?

Students found strategies helpful in guiding them to structure their work, serving as a checklist of the task's essential steps. Table I summarizes students' experiences with HowToo, which we discuss below.

Elaboration & clarification. Students reported challenges with strategies' clarity. Some addressed this by requesting the strategy author add additional information to the strategy or at a particular step. Others suggested including an example of a scenario or concept or adding "additional steps" to the strategy. However, students were concerned about how adding detail could sacrifice the strategy's simplicity. One team reported:

"I think the fact some strategies needed to be improved a lot also hindered how well they could help us, as some strategies were more confusing than others to implement."(T3)

In other cases, students believed that they needed additional resources or tutorials to understand how to perform specific steps in the strategy. Some requested the strategy author to provide background or tutorial resources with which they could learn unfamiliar concepts. Others needed "guidelines" for decision-making. For example, in a strategy which structured requirements extraction from a design prototype, students wanted additional guidance on whether all of the requirements were met and all the scenarios were covered.

Incompleteness, wording, & syntax. Students reported difficulties with confusing wording and strategies not being comprehensible, not covering all possible scenarios, not including descriptions on how to detect edge cases and separate overlapping concepts, and not understanding when a step is completed.

The Roboto syntax was new to students, and some reported that some Roboto syntax was complex. For instance, students found the "foreach" loop less confusing to use than the "until" loop. Less-experienced students had more difficulty understanding the Roboto syntax. One participant reported that Roboto syntax's similarity to traditional programming languages made it easy to understand and follow:

"I think the "until" and "if" steps are useful in terms of guiding us through the thought process of revising our requirements."(T8)

Platform engagement. The students interacted with each other on the platform and found reading their peers' comments helpful. One student reported that communication about the strategy helped to understand the strategy better and improve its content over time:

"I think it (commenting) was more like having people share their thoughts on it (strategy). So, like, Oh, do I agree with

TABLE I: A summary of the experiences of strategy users and authors

Theme	Description
RQ1: How do students experience HowToo when using it to support their programming?	
Elaboration & clarification	Students needed more clarity on specific steps or the strategy as a whole. Additional resources were needed to learn unfamiliar concepts.
Incompleteness, wording, & syntax	Students experienced difficulties with confusing phrasing and strategy incompleteness and not covering scenarios or edge cases. Roboto syntax made strategies easy to follow for some and hard for others (e.g. confusion with the "until" loop).
Platform engagement	Students found interacting with peers helpful and desired additional engagement opportunities.
Systematic approach	Students found strategies helpful in systematic components creation and work construction, overlaps and redundancies reduction, and covering gaps. Some used strategies as a list of hints on how to get unstuck.
Following strategies	Some felt that strategies were helpful for detailed programming problems, while others felt they would be more helpful as high-level checklists for tasks.
RQ2: How does HowToo benefit students?	
High-level guidance	HowToo provides high-level guidance for completing tasks rather than a direct solution to problems.
Complementary resource	Students found HowToo strategies to be complementary to other resources they reported using: Googling, TA and Instructor, YouTube, StackOverflow, classmates, online tutorials, documentation, lab sessions, previous course materials, and textbooks
Find others with the same problem	Students reported that HowToo helped them find others with a similar problem context to receive solutions faster.

that? Do I not? I think it's more of like, at least so that user who posted a strategy understands. It's like a work in progress over time. I feel like when you're doing commenting and people get feedback and improve on it." (T6)

Others showed interest in being more engaged in the platform if there was no rush to complete the assignment. A few students said that when they returned to the training phase's strategies during the testing phase, they found those strategies very helpful if they followed them precisely and carefully. Others suggested additional features for HowToo, including visual aids, screenshots, videos, attachments, and direct messaging.

Systematic approach. Strategies supported students in problem solving. Students reported that having a strategy helped overcome problem-solving difficulties:

"The strategy was very helpful for our group to systematically create the necessary components. There was some initial difficulty to make sure that the components covered all requirements, and to make sure that there wasn't redundant overlap between components." (T3)

Some believed that strategies helped them to be more systematic, especially in structuring their work. Others reported that strategies provided them with a starting point when they were confused and hints to get unstuck at any problem solving step. One team reported:

"[The] strategy helped us narrow down and make our requirements to be concise and clear- I think it was valuable for me and my team when we were struggling to figure out where to start." (T7)

Following strategies. Some students felt that strategies were not helpful in programming-level tasks and were instead more beneficial as a high-level checklists. They believed it would be beneficial for managers to divide a task into procedural task steps. However, others found the strategies helpful for programming and learning. One team reported:

"The platform felt very developer-friendly but not to other roles of a software engineering team." (T2)

Some felt that HowToo was not helpful for developers with urgent deadlines because systematically following strategies was often time-consuming.

2) RQ2: How does HowToo benefit developers?

To answer this research question, we focused on understanding who would continue to use HowToo when it was no longer required during the testing phase. Would HowToo be considered a valuable source of information? If not, what alternative approaches might students select, and why? By evaluating the logs of interactions, we found that out of 33 students, 20 students continued using HowToo after the requirement to use it in the training phase ended.

Three out of twenty students requested a strategy, and three other students authored a strategy for their peers. The first request was *"How do I deliver and present an effective software demo?"*, and seven students, including the requester, used it. The second request was *"How do I complete an effective user testing session?"*, which eight students used. The third request was *"How do I set up a Firebase React database?"*, which six students used. Our analysis revealed that students used their classmates' strategies as a source of help for their assignments and as a reference to author a new strategy. After getting stuck, one requester reported referring to HowToo, as she believed that the course instructor and peers who might face the same problem were potential resources. She reported:

"I think that is just like feedback and knowledge from other peers or instructors on a problem like this, and this is what

you should expect." (T6)

The survey results revealed that students found HowToo to be a *"complementary"* resource, providing guidance for problem-solving. Students found HowToo helpful in becoming unstuck. Students who usually sought other resources found HowToo to be a new type of resource that helped them guide what they did. Others believed that HowToo could help them find others with the same problem more quickly.

VI. Limitations

As a preliminary formative study of a new platform, our study was necessarily limited. HowToo does not yet have the scale of the content found on most mature Q&A platforms, and so experiences with it might differ at a larger scale. Our study was with undergraduate students enrolled in a course, which differed from a professional work context in several ways. Strategy use and understanding require mindful users. Time-constrained projects may not be an ideal opportunity in which to use and learn a new strategy. Prior work found that strategy users need to repeat using a strategy to understand it and fully benefit from it [10]. In this way, developer experiences in a less time-constrained setting might differ.

To introduce HowToo and the concept of a strategy, we required the use of the platform for five weeks during a training phase, with a short phase of optional use following this. In other contexts, users might never be required to use the platform. Students were also given nominal incentives to write a strategy; such incentives might not exist. Conversely, the platform contained only 46 strategies, including the three strategies authored by students. A widely used platform might have many, many more strategies and might inspire more confidence that new high-quality strategies could be rapidly created (as is the case for content in existing Q&A platforms [7]). As there were few strategies on the platform, students generally browsed for strategies, and our study revealed little about the challenges users might face in a context with more strategies. Our results might also have been biased by the students who chose to participate in our interviews. Finally, the pandemic, online courses, and ongoing national and global crises posed significant student participation barriers, limiting our data collection.

VII. Discussion

This paper introduced HowToo, the first platform for collecting and sharing programming strategies. In a formative case study, we found that HowToo could be complementary to other resources and provide high-level guidance about how to complete a task. Consistent with prior work [7], we found that certain features of the HowToo design were critical for users, such as fast response times, clear answers, guidelines, and communication. The results also showed that, while students found HowToo beneficial, it was not helpful under time pressure, as it required students to problem solve more slowly and mindfully. Some of these findings might, or might not, vary in a professional work context, where the quality of work might sometimes outweigh schedule pressure.

The instructor (the 4th author) also interpreted the students' experiences. When adopting HowToo, she leveraged a multi-year history of observing students struggle with design specifications, requirements engineering, architecture, and test planning while authoring strategies. The strategies she wrote targeted particular skill misconceptions by students, and she hoped that focusing the strategies on common difficulties would improve the quality of student work. Across the set of strategies she wrote, she targeted students' insufficient attention to detail by scaffolding a specific repetitive task (e.g., checking every requirement for certain properties, writing a comprehensive acceptance test plan based on requirements). Therefore, she intended to help students complete those repetitive tasks more thoroughly than in previous years. The teaching diary revealed that students appreciated the guidance on skills they lacked and reported feeling like their work was more thorough. However, they desired more guidance than the strategies provided. In assessing students' work, she perceived their submissions to be even more complete and with greater attention to detail than in previous years. However, the strategies did not necessarily lead to improvements in other dimensions that were not targeted.

Based on the students' feedback, the log data, and the instructor's reflections, one clear implication is that community engagement is critical to the success of strategy sharing. We found that strategy users mainly used communication within strategies to clarify confusion points and help others when they got stuck. Some of the comments suggested alternative approaches to improve the strategy. More communication and collaboration on the platform could make it a valuable resource for lurking, offering additional insight into strategies. Students referred back to HowToo during testing when they found others with relevant knowledge (peers, TA, and instructor) more easily, which assured faster responses. Consistent with prior work [7], this demonstrates the importance of fast responses to motivating participation in the platform.

Acknowledgements

We thank Hassan Assif and Stephen Hall for their contributions to the development and testing of HowToo and our study participants for their time. This work was supported in part by the National Science Foundation under grants CCF-1703734 and CCF-1703304.

References

[1] N. Bakhuisen, "Knowledge sharing using social media in the workplace," *University Amsterdam*, 2012.

[2] S. Panahi, J. Watson, and H. L. Partridge, "Social media and tacit knowledge sharing: Developing a conceptual model," *World Academy of Science, Engineering and Technology*, pp. 1095–1102, 2012.

[3] M.-A. D. Storey, L. Singer, B. Cleary, F. M. F. Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," *IEEE/ACM International Conference on Software Engineering*, pp. 100–116, 2014.

[4] T. Chau and F. Maurer, "Knowledge sharing in agile software teams," *Logic versus approximation*, pp. 173–183, 2004.

[5] L. Singer, F. M. F. Filho, and M.-A. D. Storey, "Software engineering at the speed of light: How developers stay current using Twitter," *IEEE/ACM International Conference on Software Engineering*, pp. 211–221, 2014.

[6] M. Storey, L. Singer, B. Cleary, F. M. F. Filho, and A. Zagalsky, "The (r) evolution of social media in software engineering," *Future of Software Engineering Proceedings*, pp. 100–116, 2014.

[7] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest Q&A site in the west," *ACM Conference on Human Factors in Computing Systems*, p. 2857–2866, 2011.

[8] J. Herbsleb and D. Moitra, "Global software development," *IEEE Software*, pp. 16–20, 2001.

[9] M. Raadt, R. Watson, and M. Toleman, "Chick sexing and novice programmers: Explicit instruction of problem solving strategies," *Australasian Conference on Computing Education*, pp. 55–62, 2006.

[10] T. D. LaToza, M. Arab, D. Loksa, and A. J. Ko, "Explicit programming strategies," *Empirical Software Engineering*, pp. 2416–2449, 2020.

[11] M. A. Francel and S. Rugaber, "The value of slicing while debugging," *Science of Computer Programming*, pp. 151–169, 2001.

[12] B. Xie, G. L. Nelson, and A. J. Ko, "An explicit strategy to scaffold novice program tracing," *ACM Technical Symposium on Computer Science Education*, pp. 344–349, 2018.

[13] D. J. Gilmore, "Expert programming knowledge: A strategic approach," *Psychology of Programming*, pp. 223–234, 1990.

[14] P. L. Li, A. J. Ko, and J. Zhu, "What makes a great software engineer?" *IEEE/ACM International Conference on Software Engineering*, pp. 700–710, 2015.

[15] S. Baltes and S. Diehl, "Towards a theory of software development expertise," *ACM Joint Meeting of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 187–200, 2018.

[16] D. Wieringa, C. Moore, and V. E. Barnes, *Procedure writing: Principles and practices*. Battelle Press, 1998.

[17] A. Gawande, *The checklist manifesto*. New York: Picadur, 2010.

[18] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" *IEEE Working Conference on Reverse Engineering*, pp. 142–151, 2013.

[19] J. Cao, S. D. Fleming, M. Burnett, and C. Scaffidi, "Idea garden: Situated support for problem solving by end-user programmers," *Interacting with Computers*, pp. 640–660, 2015.

[20] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett, "Programming, problem solving, and self-awareness: Effects of explicit guidance," *ACM Conference on Human Factors in Computing Systems*, pp. 1449–1461, 2016.

[21] D. Loksa and A. J. Ko, "The role of self-regulation in programming problem solving process and success," *ACM Conference on International Computing Education Research*, pp. 83–91, 2016.

[22] A. J. Ko, T. D. LaToza, S. Hull, E. A. Ko, W. Kwok, J. Quichocho, H. Akkaraju, and R. Pandit, "Teaching explicit programming strategies to adolescents," *ACM Technical Symposium on Computer Science Education*, pp. 469–475, 2019.

[23] R. A. DeMillo, H. Pan, and E. H. Spafford, "Critical slicing for software fault localization," *ACM International Symposium on Software Testing and Analysis*, pp. 121–134, 1996.

[24] B. Vasilescu, V. Filkov, and A. Serebrenik, "Stack Overflow and GitHub: Associations between software development and crowdsourced knowledge," *IEEE International Conference on Social Computing*, pp. 188–195, 2013.

[25] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social Q&A sites are changing knowledge sharing in open source software communities," *ACM Conference on Computer Supported Cooperative Work & Social Computing*, pp. 342–354, 2014.

[26] M. Allamanis and C. Sutton, "Why, when, and what: Analyzing Stack Overflow questions by topic, type, and code," *IEEE/ACM International Conference on Mining Software Repositories*, pp. 53–56, 2013.

[27] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web?" *IEEE/ACM International Conference on Software Engineering*, pp. 804–807, 2011.

[28] "Piazza," retrieved May 6, 2021 from https://piazza.com/.

[29] M. Vellukunnel, P. Buffum, K. E. Boyer, J. Forbes, S. Heckman, and K. Mayer-Patel, "Deconstructing the discussion forum: Student questions and computer science learning," *ACM Technical Symposium on Computer Science Education*, pp. 603–608, 2017.

[30] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in Stack Overflow," *ACM Symposium on Applied Computing*, pp. 1019–1024, 2013.

[31] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: Integrating web search into the development environment," *ACM Conference on Human Factors in Computing Systems*, pp. 513–522, 2010.

[32] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack Overflow considered harmful? The impact of copy&paste on android application security," *IEEE Symposium on Security and Privacy*, pp. 121–136, 2017.

[33] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online Q&A forum reliable?: A study of API misuse on Stack Overflow," *IEEE/ACM International Conference on Software Engineering*, pp. 886–896, 2018.

[34] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on Stack Overflow," *IEEE Transactions on Software Engineering*, 2019.

[35] S. Baltes and S. Diehl, "Usage and attribution of Stack Overflow code snippets in GitHub projects," *Empirical Software Engineering*, pp. 1259–1295, 2019.

[36] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," *IEEE/ACM International Conference on Software Engineering*, pp. 344–353, 2007.

[37] J. Nielsen, *Usability engineering*. Morgan Kaufmann, 1994.

[38] P. H. Sellers, "The theory and computation of evolutionary distances: Pattern recognition," *Journal of Algorithms*, pp. 359–373, 1980.

[39] J. Reason, *Human error*. Cambridge University Press, 1990.

[40] N. Dell, V. Vaidyanathan, I. Medhi-Thies, E. Cutrell, and W. Thies, "'Yours is better!': Participant response bias in HCI," *ACM Conference on Human Factors in Computing Systems*, pp. 1321–1330, 2012.

[41] D. Hammer and L. K. Berland, "Confusing claims for data: A critique of common practices for presenting qualitative research on learning," *Journal of the Learning Sciences*, pp. 37–46, 2013.

[42] J. Saldaña, *The coding manual for qualitative researchers*. SAGE Publishing, 2009.

[43] M. B. Miles and A. Huberman, *Qualitative data analysis: An expanded sourcebook*. SAGE Publishing, 1994.