Supporting Knowledge Acceleration for Programming from a Sensemaking Perspective

Abstract
Programmers spend a significant proportion of their time searching for and making sense of complex information. However, they often lack effective tools to help them make sense of the information, turn it into knowledge, or share it with their respective communities. In this position paper, we aim to help programmers collect, navigate, and organize knowledge to meet their goals while capturing this knowledge and making it useful for later programmers with similar needs. We describe barriers and challenges to creating this sustainable cycle, and we explore the design space and opportunities for effective tools and systems.

Author Keywords
Empirical Studies of Programmers; Development Tools; Sensemaking; Collaboration; Knowledge Transfer; Exploratory Search.

ACM Classification Keywords
H.5.2 [User interfaces (User-centered design)]: Miscellaneous

Introduction
Programming is highly cognitively demanding, requiring programmers to perform many activities at the same time, for which there is little direct support. For example, programmers often must understand existing code written
by others, determine how to write new code based on a large number of constraints and requirements, identify relationships and design rationale of code in order to make changes that work correctly, select among a set of application programming interfaces (APIs, also called libraries, toolkits or software development kits – SDKs), each having different properties and constraints. In fact, researchers have identified hundreds of questions that programmers ask and want to find answers for [28, 15], from basic questions like “what is the syntax for writing a while loop in Python” to more strategical and high-level ones like “how to progressively adopt React.js (a JavaScript library developed by Facebook) in my existing project”.

All of these cognitive tasks can be classified as attempts by the programmers to gain knowledge about their code, APIs, requirements, etc. Although there are many tools that focus on helping programmers find that knowledge (e.g., [16, 2, 24, 31, 25, 34]), there are surprisingly few that help programmers leverage that knowledge after it is discovered.

It is well known that programmers do not like to comment their code (for example, to add in “design rationale” for why the code is the way it is) or do extra work that they do not envision as being of immediate benefit to their main task of getting code to work [30]. However, recent research (e.g., [12, 6, 11, 13, 3, 1, 5, 8]) in other domains have identified needs and mechanisms relating to people performing complex cognitive tasks on the Internet, which have a high benefit and low cost for the individual as well as providing benefits for others later. For example, people spend a significant amount of time and effort trying to capture information and save it in structured ways. Wikipedia lists more than 20 commercial social bookmarking systems in which users provide keyword tags for web pages; in 2008, Delicious alone reported more than 5.3 million users and 180 million unique bookmarked URLs. For note-taking and organizational tools, Wikipedia lists more than 60 systems; Evernote, one of the more popular, claims more than 200 million users. Research has demonstrated the usefulness of cues from others to help an individual find what they want. Such approaches range from social navigation to social filtering to social bookmarking to social search [7, 19, 17, 33, 20, 18]. Our preliminary findings show that programmers similarly are often also collecting and organizing information for themselves, and programmers are often willing to help others when there is a clear benefit, such as answering Stack Overflow questions. We propose to leverage these trends to make it easier for programmers to collect, navigate, organize, and share knowledge. In particular, the specific research questions of our proposed research include:

- After a programmer develops knowledge for themselves, what about what they have done is generalizable for others?
- How do we capture that in a sustainable and incentivized way that will have a low cost and high perceived benefit for the initial programmer?
- How can we combine the knowledge from multiple programmers who are investigating related issues so that it will be useful for later programmers?
- How can we extract design rationale, discovered constraints, requirements and features, facts about code and APIs, and other useful information from the accumulated knowledge?
- How can we organize and present the accumulated knowledge in a way that would help the initial and later programmers?
- Where can such knowledge be surfaced so that programmers will encounter it when it is useful to them?

Our approach to start answering these research questions
will elicit theories and practices in cognitive psychology, human-computer interaction (HCI), and software engineering, and will include both new studies and novel tools. The studies will help better understand programmers’ knowledge needs and activities. The tools will include extensions for browsers, text editors and integrated development environments (IDEs) to help programmers gather (“forage” [22]), navigate, organize, and share that knowledge. We envision that programmers will be able to generate and discover the knowledge they need at an accelerated pace with our supporting tools acting as a programming copilot.

Preliminary Pilot Studies
In preparation for the research and to better motivate the problem, we conducted retrospective walkthroughs with 16 experienced programmers about a complex programming task they had recently done as part of their normal programming. The goal was to identify common activities and leverage points related to the cost structure of sensemaking. While significant prior work has examined the sensemaking process [26, 4, 14, 6, 12, 23], including in the context of programming [10, 9, 32], our focus here was on identifying common activities and their key costs and benefits that could inform tools to help the initial programmer while at the same time capturing their cognitive work to benefit others.

Participants engaged in a variety of tasks involving significant sensemaking activities, ranging from learning the React.js library for JavaScript (reactjs.org) to issuing bug reports, and used a variety of tools to annotate and save the information they encountered, including browser tabs and bookmarks, note-taking tools such as Workflowy, Notes, Evernote, Google Docs, or even building their own custom websites. Although the specific information needs varied across tasks, they generally fell into the two high-level sensemaking categories of gathering information and organizing that information into useful models to take action [26, 4, 23]. Interestingly, as a result of using a web browser to gather this information and trying to keep track of it, we also noticed a need to navigate and keep track of the various branches of options being considered, with participants encountering and queuing up possibilities to consider later [21, 29]. The types of tasks we observed programmers doing to try to accomplish these activities included:

Deciding on a framework, API, approach, pattern, or code example to use. Prior work has shown that 34% of a programmer’s search sessions are aimed at finding or learning about an API [27], with each option having different strengths or limitations depending on the programmer’s goals and constraints (e.g., existing coding stack, style preferences, level of expertise, etc.). We observed two participants at the beginning stages of a project, both having trouble deciding what framework or API to use. One programmer noted difficulties with foraging for information: “The hardest part is figuring out if an API actually has the functionality I’m looking for.” Programmers often organized the information they collected to foster comparisons. One participant stated, “Because there are so many tools [for web development], you almost want to read something that tells you, React is a good idea. Even though it’s hard at first, this is why it’s powerful.” They often started following one lead but switched to another when they ran into difficulties, suggesting the need for support for branching and backtracking navigation.

Learning the structure of an unfamiliar framework, API, or unfamiliar code. A majority of our interviewees cited learning unfamiliar code as a significant challenge, including new concepts (e.g., “Promise” in next-generation JavaScript) or terms (e.g., “event-driven”). One participant stated, “A lot
Sensemaking activity | Task types involving sensemaking
--- | ---
**Deciding on a framework, API, approach, pattern, or code example to use** | **Learning** the structure of an unfamiliar framework / API; understanding unfamiliar code | **Implementing** a specific feature/functionality

| Foraging | Identifying pros and cons, important constraints, and contexts of use | Annotating unfamiliar concepts or code; capturing useful explanations | Marking pages where code was copied; annotating how it was adapted

| Navigating | Keeping track of and switching among different options | Supporting deep diving into explanations and resuming main thread | Queuing potential options to implement and switching among them

| Organizing | Building a comparison table of functionality | Filtering foraged tutorials and examples by users’ goals | Visualizing trees of potential options

| Sharing | Exposing design decisions as design rationale for other programmers | Exposing best practices where they are relevant (e.g., in code or on relevant search pages) | Shortcutting bad “rabbit holes”; showing different styles of solutions

**Table 1:** Design space we plan to explore, with example opportunities in each cell.

Design Space
These initial results suggest a design space to profitably explore ways to understand and help programmers forage, navigate, and organize different resources they encounter in order to achieve their own programming goals, and to share the results with others. Table 1 shows a preliminary depiction of this design space, with different types of programming tasks commonly occurring in our preliminary pilot study as the columns, and different activities that people engage in while interacting with information for these tasks as the rows. Within the cells are examples of opportunities...
for helping support that activity for that task. For example, when deciding on a mapping API to use, the programmer might gather important constraints (e.g., whether it supports geolocation; how much does an API call cost; whether it supports overlays) and the contexts in which it can be used (e.g., if it plays well with the React framework). We propose an important advance in the addition of the sharing row, in which our main focus is the implicit sharing where a programmer’s cognitive work might be used by others later. For example, if one programmer has already gathered the constraints and contexts in which they decided to use one API over others, those constraints and contexts might be surfaced as design rationale in that user’s code, or they might be surfaced if another user did a similar search or visited the same web pages as the initial programmer.

One of the key goals of our research is to make this table more complete by filling out the rows and columns and the opportunities within each cell. The top three rows will help us find leverage points to create tools that capture the work that people do during sensemaking and are perceived as being useful by themselves; the last row embodies how that captured work can be made useful for others.

Conclusion
Programming has been highly cognitively demanding. We propose to go beyond previous approaches to understand and support sensemaking in programming by focusing on the opportunities to capture the cognitive work that programmers engage in while making sense of information in order to help them and others with similar knowledge activities. We aim to better understand the cost structure of sensemaking in programming, and use it to build tools that directly reduce the cost and/or increase the benefit to the initial users in making sense of information. To do so we will engage in two primary thrusts: empirical studies to better understand the activities, information needs, results and cost structure of sensemaking across various types of programming-related activities, and the development and evaluation of prototype tools to explore the design space of supporting those activities.

REFERENCES


