Context in Programming: An Investigation of How Programmers Create Context

Souti Chattopadhyay
chattops@oregonstate.edu
Oregon State University

Nicholas Nelson
nelsonni@oregonstate.edu
Oregon State University

Thien Nam
namt@oregonstate.edu
Oregon State University

McKenzie Calvert
calvertm@oregonstate.edu
Oregon State University

Anita Sarma
anita.sarma@oregonstate.edu
Oregon State University

ABSTRACT

A programming context can be defined as all the relevant information that a developer needs to complete a task. Context comprises information from different sources and programmers interpret the same information differently based on their programming goal. In fact, the same artifact may create a different context when revisited. Context, therefore, by its very nature is a "slippery notion."

To understand how people create context we observed six programmers engaged in exploratory programming and performed a qualitative analysis of their activities. We observe that the interactions with artifacts and a mapping of meaning from those artifacts for a programming activity determines how one creates context.

KEYWORDS

Programming context, programmer behavior, qualitative studies

ACM Reference format:
Souti Chattopadhyay, Nicholas Nelson, Thien Nam, McKenzie Calvert, and Anita Sarma. 2018. Context in Programming: An Investigation of How Programmers Create Context. In *Proceedings of 11th International Workshop on Cooperative and Human Aspects of Software Engineering, Gothenburg, Sweden, May 27, 2018 (CHASE'18),* 4 pages.
https://doi.org/10.475/123_4

1 INTRODUCTION

Programming, in particular the act of coding, does not occur in isolation. It involves ideating and exploring different solutions and using different types of information (in the codebase and from online/external resources) to complete a task. These relevant pieces of information, along with the programmer's prior knowledge, creates a context that the programmer uses to solve the task.

In software engineering, 'context' has been described as the perspective gained from all relevant information obtained from these different sources. Although we intuitively understand context; it is a "slippery notion" [5]—hard to formally describe and define. It is dynamic in nature, morphing with every action and evolving along with changing goals.

CHASE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

Context has been researched in both software engineering and ubiquitous computing, resulting in two disparate views of context—representational and interactional context [5].

The *representational view* describes context as delineable and stable [1, 12, 15]. To operationalize this view, researchers have attempted to encode context through artifacts, tasks, and environmental factors. For example, Gasparic et al. [7] discuss how context can be modeled by environmental factors like who is working, what is the environment, and which artifacts are involved. However, focusing only on how context is represented leaves gaps in our understanding of how context is created, how it is affected by developers' goals, and how it changes with evolving goals.

The *interactional view* defines context as a relational property that exists between objects or activities; and one cannot be viewed disjointed from the other [5]. Because of this interconnected relationship, contextual factors must be defined dynamically for each activity, and the sequence in which they occur is important.

In this paper, we present a study of six programmers and observations about how programmers create context by interacting with artifacts. Our qualitative analysis provides evidence that context crosscuts activities and artifacts, calling for the need of merging the representational and interactional views of context.

2 RELATED WORK

Context, while an intuitive concept, is very hard to define due to its highly dynamic nature. Schilit and Theimer [16] defined context in the "context-aware" computing domain as being various factors of a user; such as location, time, and identity.

Brown [3] proposed a definition which included the characteristics of the user, the environment, and the application. However, these definitions are inclined towards static models of the factors that affect context, and assume very little interplay between them.

Models like Mylar [10] and Hipikat [4] capture various factors like the nature of programming tasks, artifacts, and the development environment. These models attempt to identify the context that directly or indirectly contributes to the programmers ability to construct meaningful software.

This kind of *representational* perspective of context is accurate for modeling software systems where the state of the program is the primary focus. However, these models can only answer *"what can be used to represent context?"* rather than *"what is context?"*

Pascoe [12] and Abowd et al. [1] propose perspectives that focus on the *interactional* nature of context—where context is a subset of

the physical and conceptual states of the user—the user's emotions, attention, and informational states.

However, *interactional* context is difficult to operationalize due to the transitory nature and focus on users. Although Spyglass [17] was developed with this view of context to create a recommendation system; a model that accounts for the interactions is still missing.

3 METHODOLOGY

We conducted a lab study to observe six graduate students with professional programming experience performing a programming task. Their details are presented in Table 1. We provided participants P1-P5 with a prompt requiring planning, designing, and developing software to simulate a traffic intersection. This prompt had previously been used in other studies [11]. This prompt was easy to understand but nontrivial to implement, allowing us to observe participants' use of context.

We additionally observed participant P6 working on a real-world problem involving development of his own IDE. We do so to contrast the results of participants who were working on a given prompt, from those that occur when a programmer is working on their on own programming task.

Table 1: Participant Demographics

Participant	Gender	Programming Language[i]	Programming Experience[ii]
P1	M	Scala	>10 yrs.
P2	M	C/C++	7 yrs.
P3	F	Python	5 yrs.
P4	F	Java	7 yrs.
P5	F	Python	>10 yrs.
P6	M	JavaScript	>10 yrs.

[i] Selected by participant. [ii] Across all programming languages.

Our participants provided a brief, but diverse subset of programming styles; P1 adhered to the Test-Driven Development (TDD) model, and P6 displayed a strong affinity for tinkering [2].

We time-boxed the study to one-hour, to prevent participant fatigue. Participants used their preferred development tools and programming language. They were also provided blank paper and given the option to think-aloud. P1 chose to think aloud, which we used to validate our interpretations of P1's actions. We collected a diverse dataset: audio, screen recording, external notes.

To analyze the data, thefi rst author unitized the screen recordings and audio transcripts into continuous time segments. Each unit contains a logically consistent group of related interactions that represent a small part of their programming task.

For each of these units, we identified the programming activity, the artifacts used, and their frequency. Table 2 lists the 11 programming activities we use in our analysis. The codeset builds on [18], with slight modifications tofi t our task prompt: we changed `READING QUESTIONS` to `READING TASK PROMPT (A4)`, and added `UPDATING DOCUMENTS (A1)`.

Using this codeset, thefi rst and the third authors obtained an Inter-Rater Reliability (IRR) score of 97.14% and thefi rst and the fourth authors obtained an IRR score of 92% when coding random segments from the selected participants dataset (which constituted ~14% of the raw data). The remaining 86% of the data was coded individually by the same authors.

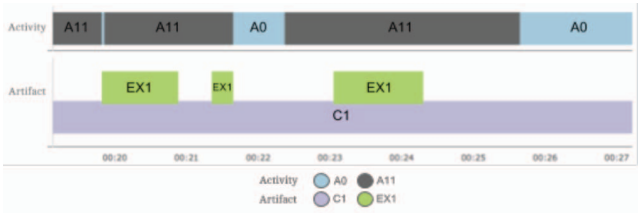

Figure 1: Time sequence of Activities and Artifacts from P1

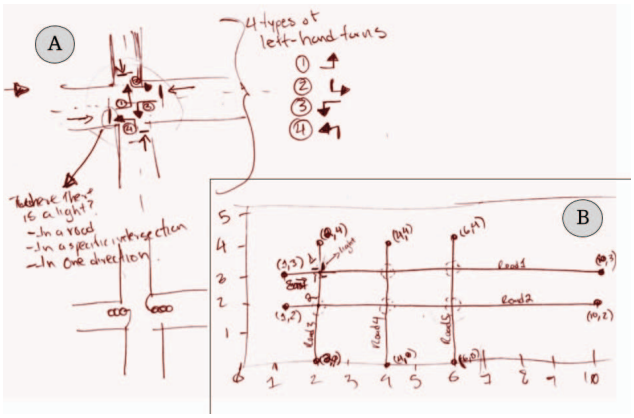

Figure 2: External artifacts created by P1 during study; (A) shows a model of the interactions within a traffic intersection, (B) represents a potential layout for roads and intersections on a grid.

4 STUDY RESULTS

Here we present our observations of how participants' interact with the artifacts during specific activities. Our focus was to develop an understanding of how the context building process occurs, and gain insights into which factors play a role in shaping context.

We observe the types of artifacts that participants refer to during different programming activities. The participants interacted with 6 different types of artifacts: *source codefiles* , *documents* (task prompt), *IDE tools* (e.g.: *New Class* dialog box, terminal), *web resources* (e.g. search results, Q&A pages), *externalized artifacts* (e.g. paper diagrams), and *other* (e.g. calculators).

For each participant, we refer to source codefi les in the format `C1`,`C2`, etc., the prompt document is `D1`, IDE tools follow `T1`,`T2`, etc., web resources follow `W1`,`W2`, etc., and external artifacts as `EX1`. The numbering of these tags represents the sequential order in which participants encountered or interacted with them.

4.1 Artifacts span heterogeneous medium

During our study, participants not only accessed code artifacts, but also tools within the IDE, online resources, and external documents to help them in their programming task. Here we discuss the different types of artifacts and the frequency in which they're used.

Participants, as can be expected, accessed the code artifacts most frequently when involved in their coding activity (`A0`). During their hour-long session, P1 accessed three source codefi les 87 times and P6 accessed four source codefi les 147 times.

Table 2: Programmer Activities Codeset

Code	Programming Activity	Activity Description
A0	CODING	The subject interacts with source code in the editor.
A1	UPDATING DOCUMENTS	The subject altered, or updated, the task prompt document.
A2	NAVIGATION	The subject navigates tofi nd thefi le or pages they need.
A3	READING TASK PROMPT	The subject read and inspected the task prompt document.
A4	SEARCHING	The subject uses search to locate resources to support his problem solving.
A5	READING SEARCH RESULTS	The subject reads the search results to decide which results to click.
A6	PROCESSING SEARCH RESULTS	The subject decides how to deal with the search results.
A7	VIEWING WEB RESOURCES	The subject views online programming support resources.
A8	DEBUGGING	The subjects tries tofi x the problem in the code.
A9	RUN (incl. test running)	The subject uses tools to run the program and view the results.
A10	IDLE	The subject does nothing for a specific period (≥ 3 *seconds*)

The next most frequently accessed artifacts were on-line resources. Participants often used these resources to learn how to use a feature or recall implementation details. For example, P6 accessed seven web resources 91 times—he used blog posts to learn how to correctly use the "`draggable`" feature.

We found that participants also frequently engaged in creating external artifacts (e.g., notes, updating the traffic prompt, and creating diagrams). Figure 1 shows a plot of P1's activities and the artifacts he interacts with for an 8-minute segment of his session. We found that P1 (based on his think aloud data) was ideating about possible solutions. If we simply analyze his on-line interactions it shows that he was interacting with the codefile `intersection.scala (C1)` while his programming activity switched between coding `(A0)` and being idle `(A1)`. However, this belies the fact that he was ideating different solutions during this period; he was modeling the traffic movement at the intersection (Fig 2.A) and the data model to represent the intersections (Fig 2.B).

Our observations indicate that when defining context and how it is built, we need to also consider artifacts that are outside the IDE. Not doing so [7, 10], leaves gaps in our understanding.

4.2 Programming activity guides interaction with artifacts

The programming activity guided the kinds of artifacts, and medium, that were accessed and also how participants interacted with them.

For example, from Figure 3.A, when interacting with the code artifact `cards.js (C1)`, was involved in two separate activities (coding `(A0)` and searching `(A4)`. To update a feature, P6 produced new code in `cards.js (C1)`, but then realized that he needed to update all the other parts of the program that were affected. He searched for the particular term within the code and switched back to coding as necessary. While both of these activities, coding `(A0)` and searching `(A4)`, required P6 to interact with the code artifact `C1`, the type of interaction varied. When coding `(A0)` between `00:15:57–00:16:29`, P6 primarily typed in short bursts that were interspersed with scrolling interactions. Whereas, during subsequent searching `(A4)`, P6 mostly scrolled following the highlighted instances of the term he searched and intermittently copied text.

The interactions with artifacts provides the information which guides an activity forward, sometimes causing a switch to different activity. In Figure 3.B, P6 looked through a `stackoverflow.com` (SO) page `(W3)`. He is viewing web resources `(A7)`, scrolling and intermittently pausing to read the answers on the page until he finds a probable answer. Then he switches to coding `(A0)`, and for the next two minutes, continued to switch between the activities coding `(A0)` and viewing web resources `(A7)`. Toward the end of this sequence of activities, P6 scrolled quickly to a desired section and copied the text before returning to code.

While the above examples are intuitive and simple, they shed light on how activity and interactions with artifact are closely tied together. They provide evidence to the notion that not only does context arise from the activity, but the programming activity and the goal behind it guide the specific type and amount of information obtained from the artifact.

4.3 Interaction includes Reflection

Participants evaluated the information within the artifacts that they accessed. Typically these evaluations coincided with prolonged interactions (e.g, long periods of scrolling or highlighting parts of an artifact). After such evaluation periods participants started to code. If participants found errors then they reflected on the information that they extracted from the artifacts.

We observed such cycles of *Evaluation-Reflection* in all sessions. Figure 3.B presents such a cycle for P6. He read through multiple answers on the SO page `(W3)`, evaluating each one (marked by *'Evaluation'* arrow). After selecting the most appropriate (perceived) solution, he started to code `(A0)` and run the code `(A9)`; Following these steps multiple times as well as revisiting the selected answer on the page `(W3)`. When the selected solution proved to be incorrect, he went back to the same `(W3)` page and spent 30 seconds (twice the time of previous visits) reflecting on the solutions and their appropriateness.

We saw a similar loop for P1; He sought help when implementing a feature (case class) as seen by his comment: *"I don't know how to do this"*. Thus, he read the scala documentation, evaluating the information on the page. After scrolling through the page multiple times he reflected that the solution would be too difficult and implementing it was not worth the effort for the task at hand: *"How do I do this smartly? . . . fi ne, I'll just do it with strings"*.

5 DISCUSSION AND FUTURE DIRECTIONS

Limitations: Like many other research papers based on formative analysis, our study has limitations. First, this was an exploratory

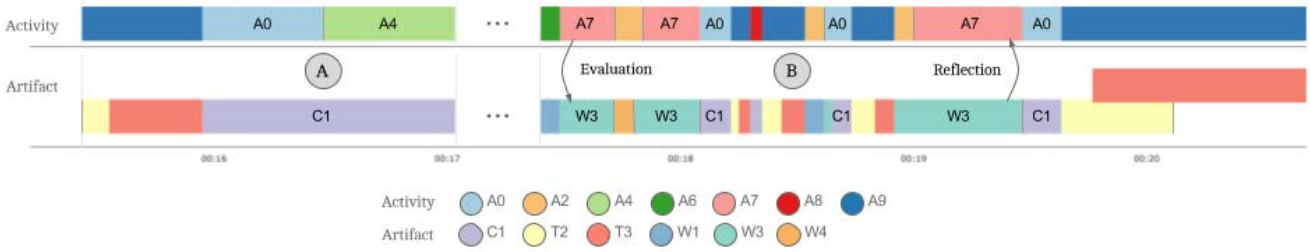

Figure 3: Time sequence of Activities and Artifacts from P6 during study; (A) shows a sequence of `CODING (A0)` and `SEARCHING (A4)` across the same `C1` source codefi le, (B) represents a cycle of evaluation, action, and reflection.

study that only included six participants recruited through convenience sampling. All participants had at leastfi ve years of programming experience, with three participants having three or more years of professional programming experience. Second, we collected different types of data across the participants– screen sharing was captured for all participants, but P4 included video of the participants face during the study and P1 provided think-aloud audio. Since our primary goal was to observe the interactions between participants and the artifacts they utilized during programming, screen capture data was the primary focus during our studies.

Future Directions: Our initial analysis reveals two future directions that we plan to pursue.

5.0.1 Using Information Foraging Theory to inform context creation. Context has been largely defined as all 'relevant information' that an individual needs to complete a task. We believe that Information Foraging Theory (IFT) constructs [6, 14] can help us model how programmersfi nd information in an artifact and decide whether it is relevant as they carry out their tasks. IFT explains that the consumption of information is similar to how animals hunt for food—following scent of their prey and choosing more valuable prey through a cost-benefit analysis. We observed that participants engaged in such evaluation of artifacts and the information contained within when building context.

IFT has been applied in the programming domain to study how varying goals [13] affect the perceived value of information and how the (perceived) cost of 'consuming' information varies across different types of artifacts (e.g., web site vs. Q&A forum) [9]. We plan to build on these works to explore how IFT can help model programmers context building behavior.

5.0.2 Mapping and Memory Modeling. Our observations show that participants accessed artifacts and reflected on the information contained in them. Past work has alluded to how individuals perform "sensemaking" of information contained withing artifacts tofi t their (task) goal [8], but the process by which individuals map the information theyfi nd to the problem or solution space has not been modeled. Moreover, once a context has been built parts of the context (information) can be useful and recalled from memory for another (programming) task. Further studies are needed to understand how individuals recall snippets of context from other tasks to aid their current one or how temporality of actions may cause decay in memory and the ability to recall context.

REFERENCES

[1] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing*. Springer, 304–307.

[2] Laura Beckwith, Cory Kissinger, Margaret Burnett, Susan Wiedenbeck, Joseph Lawrance, Alan Blackwell, and Curtis Cook. 2006. Tinkering and Gender in End-user Programmers' Debugging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 231–240. https://doi.org/10.1145/1124772.1124808

[3] Peter J Brown. 1995. The stick-e document: a framework for creating context-aware applications. *Electronic Publishing-Chichester-* 8 (1995), 259–272.

[4] DavorČubrani ć and Gail C Murphy. 2003. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th international Conference on Software Engineering*. IEEE Computer Society, 408–418.

[5] Paul Dourish. 2004. What we talk about when we talk about context. *Personal and ubiquitous computing* 8, 1 (2004), 19–30.

[6] Scott D Fleming, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. 2013. An information foraging theory perspective on tools for debugging, refactoring, and reuse tasks. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 2 (2013), 14.

[7] Marko Gasparic, Gail C Murphy, and Francesco Ricci. 2017. A context model for IDE-based recommendation systems. *Journal of Systems and Software* 128 (2017), 200–219.

[8] Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Jill Cao, Kyle Rector, and Irwin Kwan. [n. d.]. End-user Debugging Strategies: A Sensemaking Perspective. *ACM Trans. Comput.-Hum. Interact.* ([n. d.]), 5:1–5:28.

[9] Xiaoyu Jin, Nan Niu, and Michael Wagner. 2017. Facilitating end-user developers by estimating time cost of foraging a webpage. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2017, Raleigh, NC, USA, October 11-14, 2017*. 31–35. https://doi.org/10.1109/VLHCC.2017.8103447

[10] Mik Kersten and Gail C Murphy. 2006. Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 1–11.

[11] Nicolas Mangano and André vander Hoek. 2012. The design and evaluation of a tool to support software designers at the whiteboard. *Automated Software Engineering* 19, 4 (01 Dec 2012), 381–421.

[12] Jason Pascoe. 1998. Adding generic contextual capabilities to wearable computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on.* IEEE, 92–99.

[13] David Piorkowski, Scott D Fleming, Christopher Scaffidi, Margaret Burnett, Irwin Kwan, Austin Z Henley, Jamie Macbeth, Charles Hill, and Amber Horvath. 2015. Tofi x or to learn? How production bias affects developers' information foraging during debugging. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on.* IEEE, 11–20.

[14] Peter Pirolli and Stuart Card. 1995. Information Foraging in Information Access Environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. 51–58.

[15] Bill Schilit, Norman Adams, and Roy Want. 1994. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on.* IEEE, 85–90.

[16] Bill N Schilit and Marvin M Theimer. 1994. Disseminating active map information to mobile hosts. *IEEE network* 8, 5 (1994), 22–32.

[17] Petcharat Viriyakattiyaporn and Gail C Murphy. 2010. Improving program navigation with an active help system. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 27–41.

[18] Yi Wang. 2017. Characterizing Developer Behavior in Cloud Based IDEs. In *Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on.* IEEE, 48–57.