Context in Exploratory Programming: Towards a Theoretical Framework

Souti Chattopadhyay
School of Electrical Engineering and Computer Science, Oregon State University
chattops@oregonstate.edu

I. Introduction

Creativity theory states good designs are achieved by having a multitude of these designs [1]. Exploratory Programming is the process of trying out designs while writing software. Programmers have to evaluate these alternative implementations in order to implement new ideas [2] . These alternatives often have multiple objectives which might prompt a programmer to work towards multiple goals in episodes. Episodes are distinct periods when a programmer works towards a certain goal. These episodes may be interleaved where programmers compare different episodes [3]. However, there is little research which focuses on how programmers abstract meaningful and appropriate information from their exploration of alternatives to integrate into their current work. We refer to these meaningful abstractions as context.

Context is the coherent story of related alternatives. It answers questions such as: why the alternative is (or was) created, what constitutes an alternative, and what differences exist between alternatives. Programmers need to explore multiple related alternatives in order to create these context. An understanding of how programmers build this context unify the findings from studies on individual pieces of exploratory programming - evaluation, navigation and implementation - to provide a holistic view.

For my PhD dissertation, I will study, from a theoretical perspective, how programmers create, manage and re-create context. I will use theoretical frameworks to create a model which helps programmers build context in exploratory programming. I will investigate theories, strategies, models and tools that support exploratory programming. The concept of context creates a foundation that unites the disparate pieces of understanding the processes of exploration, evaluation, navigation in exploratory programming.

II. Context In Exploratory Programming

Alternative implementations have stories of their own. When a programmer evaluates these alternatives, he seeks to find these stories from the alternatives. Why is an alternative written in this way? How is this alternative different from that one? What was the meaning behind creating this alternative? The programmer seeks answer to these question in order to build a coherent story, a context, to identify the useful alternatives.

These alternatives are often created in separate episodes of programming. The artifacts created in one episode have a different shared story to tell. Figure 1 is a diagrammatic representation of contexts in three different episodes. Each shape inside an episode box is an artifact. The alternatives of the same artifact have the same shape. One can create context from alternatives from the same episode (eg. Context 1A,3) or different episodes (eg. E1). Related artifacts in the episode (not alternatives) form another kind of context (eg.1C, 2B). Sometimes programmers need to delve into a combination of alternatives and related artifacts across episodes to create a context.

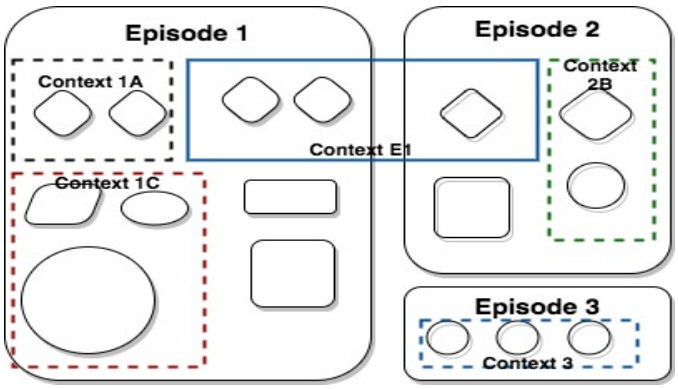

Fig. 1. Contexts in episodic development

The programmers build these context repeatedly when evaluating the artifacts and makes conscious decisions of which artifacts fit their purpose. They combine various artifacts built through continuous exploration to recreate a complete story before they implement. Understanding how they create this context is imperative to provide a conceptual framework to exploratory programming. It will also bolster the tools made to support processes like exploration, navigation and evaluation that constitutes exploratory programming.

III. Related Work

Researchers have conducted many empirical studies of information needs of programmers when designing and implementing their work. It is necessary to understand what causes these needs and how the information is truly managed and re-framed to implement [4], [5].

Tools like Hipikat and Mylyn aim to build context from the task the programmer is performing by measuring their Degree of Interest (DOI) and Knowledge (DOK) models [6], [7]. They analyze the artifacts modified by programmer to recommend relevant software development artifacts. But

978-1-5386-0443-4/17/$31.00

programmers create context even from artifacts that they have not encountered.

Tools like Variolite and Micro-Versioning build on how people explore artifacts. The PFIS-V model explains and predicts how people navigate to useful information by using Variational Foraging Theory [8] . Variational Foraging Theory stems Information Foraging Theory which states that programmers find information useful for their goal by analyzing the value of the information and the cost to navigate to it [9]. When the programmer encounters highly valuable information that is tangential to their goals it causes programmers to deviate from their planned course of action. Theory of Situated Actions states that in such unexpected cases, one acts on as much information available in that situation until they merge or re-frame their goals [10].

IV. Proposed Work

In this dissertation, I will develop a descriptive and explanatory theory of context by studying programmer behavior in the exploratory domain. This work is in the inception phase.

As I pursue this research, I will follow guidelines from prior formalizations and established theories to develop a grounded theory of building context. I will study how people create context from both planned and situated circumstances when they explore various related artifacts created across episodes. I will conduct empirical lab studies where participants will be asked to create a traffic flow simulation program [1]. I will conduct interviews and surveys to understand how programmers create context from the exploration of existing code.

I will further analyze how the process of context creation differs when they are exploring code that they have created themselves from those created by others. I will study how this context is created from alternatives, co-episodic artifacts and related artifacts. I will also explore which elements of code (structure, comments, construct naming etc.) programmers use to build the context, and their interplay. These findings will constitute the foundation of the theoretical framework.

Initial observation shows that the context a programmer creates is affected by factors like the programmers goals, phase of programming (implementation, ideation, comprehension etc.) intent behind performing the task (learning, fixing etc.) , Ownership, Awareness, and Lens the programmer is looking through (code history, current code base, or external sources like stack overflow etc.). I will investigate the role each factor plays in defining the usefulness of information and context.

Initial analysis suggests that programmers go through different stages of building context in exploratory programming. Figure 2 describes the five stages of creating context from existing sources until they successfully reach their goal which results in re-creating context. I will study these stages - how programmers **create** context from existing code, **combine** the various contexts to form a complete coherent story, how they **articulate** these contexts to understand if it fits their goals, and how they **organize** it in their minds to create a mental map which they use in case they backtrack or perform ad-hoc actions, and finally **re-create** context by integrating the useful artifacts.

[1]The detailed description of the study is available here: http://web.engr.oregonstate.edu/~nelsonni/docs/prompt.pdf

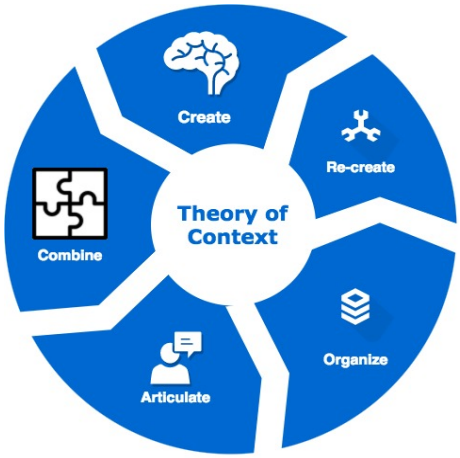

Fig. 2. The cycle of creating and re-creating context

V. Plans and Discussions

The theoretical framework resulting from the proposed work can be used to prescribe concepts and designs for tools, IDEs and interfaces. In the first two years, I plan to conduct the lab studies and build the foundation of the theoretical framework.

The following two years, I will create a model to capture programmer's manipulation of context. I will evaluate how accurately the model imitates the cognitive process of context building and management of these contexts. I will further analyze how context can extend the understanding of existing models and theories of exploratory programming.

In the conclusion of this work, I will create a tool that supports and improves the process of building context. I will evaluate how tools integrate with existing ones for exploration, evaluation and navigations to create deeper understanding and easier exploratory programming.

References

[1] J. Paul Guilford, "Intelligence, creativity and their educational implicationsj. p. guilford," 01 1968.
[2] B. e. Shneiderman, "Creativity support tools: Report from a u.s. national science foundation sponsored workshop," vol. 20, pp. 61–77, 05 2006.
[3] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Opportunistic programming: Writing code to prototype, ideate, and discover," *IEEE Softw.*, vol. 26, no. 5.
[4] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006.
[5] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '06/FSE-14, 2006, pp. 23–34.
[6] D. Čubranić and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," 2003, pp. 408–418.
[7] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," 2006, pp. 1–11.
[8] S. S. Ragavan, B. Pandya, D. Piorkowski, C. Hill, S. K. Kuttal, A. Sarma, and M. Burnett, "Pfis-v: Modeling foraging behavior in the presence of variants," 2017, pp. 6232–6244.
[9] P. L. T. Pirolli, *Information Foraging Theory: Adaptive Interaction with Information*, 2007.
[10] L. A. Suchman, *Plans and Situated Actions: The Problem of Human-machine Communication*, 1987.