Putting Information Foraging Theory to Work: Community-based Design Patterns for Programming Tools

Tahmid Nabi 1, Kyle M.D. Sweeney 1, Sam Lichlyter 1, David Piorkowski 1, Chris Scaffidi 1, Margaret Burnett 1, Scott D. Fleming 2

1 Center for Applied Systems and Software
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR, USA

2 Department of Computer Science
University of Memphis
Memphis, TN, USA
Scott.Fleming@memphis.edu

Abstract—The design of programming tools is slow and costly. To ease this process, we developed a design pattern catalog aimed at providing guidance for tool designers. This catalog is grounded in Information Foraging Theory (IFT), which empirical studies have shown to be useful for understanding how developers look for information during development tasks. New design patterns, authored by members of the research community for the catalog, concretely explain how to apply IFT in tool design. In our evaluation, qualitative analyses revealed the community-written design patterns compared well in quality to patterns that we had ourselves published in a smaller, peer-reviewed catalog.

Keywords—tool design; software engineering; applied theory

I. INTRODUCTION

Tools play a central role in enabling developers to find information efficiently during development tasks. For example, such tools include search and recommendation functions that can help a developer find the location of a bug in order to fix it [20][21], or to leave and view notes for one another [30]. To date, designers have relied primarily on intuition and empirical study for tool ideas. For example, one tool embodied the insight that developers often need to navigate through code-based on what lines of code could be called, and the tool included a novel static analysis to support navigation [19]. But this insight was gleaned only after lengthy empirical work [18].

Hence, tool designers could benefit from a synthesis of the literature in a form that highlights open areas and sparks insights. We took the first step toward this goal with a literature review [8] framed by Information Foraging Theory (IFT) [25], a theory that can explain and predict how developers seek information [20][21][24]. We examined software engineering papers and explained how programming tools revealed ways of applying IFT in practice [8], yielding 12 design patterns summarizing how those tools applied IFT concepts.

A key limitation of that preliminary catalog is that it only incorporated our own research group’s perspectives. Our current paper therefore presents an expansion of this design pattern catalog through a community-based process. Researchers from around the world contributed 16 additional design patterns, which broadened and deepened the range of ideas for how to apply IFT for tool design. We evaluated these new design patterns through a qualitative analysis.

II. BACKGROUND AND RELATED WORK

Information Foraging Theory (IFT) offers a framework for conceptualizing developer behavior [8][17][20][21][24]. To briefly summarize IFT as it applies to software engineering, developers hunt like predators for information in a topology, which consists of patches of code or other views connected by navigable links. Patches contain information features that have value in the context of the developer’s current task. A link has a certain cost, often measured in time or effort. Links may be annotated with certain cues (e.g., labels) indicating where those links lead. Developers try to maximize value relative to cost.

Prior empirical work extensively validated the benefits of applying IFT. For example, tools motivated by IFT can predict where developers will navigate and offer links to reduce foraging cost [20][21][24], organize files visually to minimize navigation cost [12], summarize code to reduce effort of program understanding [2], help developers find appropriate versions of programs during reuse [16], and search for needed API documentation [31].

We seek to go beyond the design of individual tools and establish how IFT provides broad guidance for tool design in general. Our approach is to synthesize insights from tools into design patterns, which are general, reusable solutions to common design problems [4][9]. Design patterns are abstract enough to generalize solutions among multiple situations and approaches [10][23][29], but concrete enough to aid developers in everyday design work [26]. Design patterns exist for security-related features [32], agent-based and service-oriented architectures [15][28], object-oriented systems [3][9][34], embedded systems [1][7], and visualization tools [13]. Our literature review extended this list by showing that design patterns could describe insights to guide programming-tool design [8]. For example, our catalog included the Dashboard pattern, which refers to an information patch in which a developer can become aware of links that lead to continually changing patches that have high value.

Researchers have taken a variety of approaches for evaluating design pattern catalogs. Some applied qualitative analyses (e.g., [15][32]), others obtained feedback from pattern authors (e.g., [6][11]), and still others applied patterns and observed their benefits and weaknesses (e.g., [14][27]). We used the first of these three approaches in our evaluation.