Towards aiding within-patch information foraging by end-user programmers

Balaji Athreya, Chris Scaffidi
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR, USA
athreya86@gmail.com, cscaffidi@eecs.oregonstate.edu

Abstract—Many tools help professional programmers with the difficult problem of finding information during code maintenance. The empirical success of these tools can be explained by Information Foraging Theory (IFT), which predicts how a person seeks information by navigating through an information system based on the visual weight of information features presented to the person.

Motivated by the success of these tools, we investigated the reasonable expectation that end-user programmers would likewise benefit from tools that increased the relative visual weight of important information features. We prototyped and evaluated two tools, each of which uses an existing algorithm to identify the most important lines of code. One prototype highlights important lines of code; the other prototype hides unimportant lines of code. An empirical study revealed that increasing the relative weight of important information features by highlighting did positively impact the amount of information foraged and the rate of information gained; on the other hand, decreasing the relative weight of unimportant information features by hiding had a modest negative impact. These results reveal opportunities for enhancing existing IFT-based foraging models and applying them to design more effective end-user programming tools for coding, debugging, and code reuse.

Keywords—end-user programming; code maintenance; information foraging

I. INTRODUCTION

Reusing and maintaining code requires finding information in the code, which can be time-consuming and difficult [14]. Models based on Information Foraging Theory (IFT) (e.g., [19][20]) have helped to clarify how programmers go about this process. First, a typical programmer locates a part of the code (such as a function or method) that might contain needed information. Then the programmer mentally processes information in that part of the code and decides where to look for more information, if necessary. The result is a series of visits to multiple locations. IFT refers to these locations as “patches” and the pieces of information within the patches as “features.” It posits that people try to minimize the sum of two kinds of costs incurred for obtaining a given set of information: the cost of navigating to patches and the cost of extracting features from within patches.

Two key strategies for helping programmers with information foraging therefore become apparent.

One strategy is to reduce the cost of navigating to patches—a strategy that is evident in many programming tools that provide navigable links between patches (e.g., [5][12]). Such tools have proven useful for both professional and end-user programmers, typically by helping programmers to visit fewer patches by navigating to useful patches more directly.

Another, alternative strategy is to reduce the cost of extracting information from within each patch. This strategy, unlike the first, has received much more attention in the context of professional programmers than in that of end-user programmers. Some tools for professional programmers perform code summarization that identifies and consolidates the most important statements while eliding the less important statements (e.g., [22][25]). Other tools provide automatic statement highlighting that emphasizes the most important statements (e.g., [4][6]).

All such tools incorporate heuristics for identifying important statements, and they can be understood in terms of IFT as manipulating information features by emphasizing important statements relative to unimportant statements. To date, manipulation of information features has received limited empirical validation for use in helping end-user programmers with processing within patches. The main work in this regard has been on tools that use highlighting to help end-user programmers find errors during debugging (e.g., [23][28]).

Nonetheless, in the broader sense of helping people to actually understand patches (not just “find the bugs”), it is questionable whether increasing the relative weight of important information features will be as helpful for end-user programmers as for professional programmers. One reason to doubt the effectiveness of this approach is that end-user programmers’ code is rarely designed for reuse or maintenance and is often incomprehensible—Nardi writes, “It is not clear whether users who modify existing example programs could ever really come to understand the programs they modify” [17]. Such code might not have many information features important enough to highlight, or perhaps the important features are not as identifiable in end users’ code as in professional programmers’ code. On the other hand, end-user programmers only write code as a secondary task in their work,