Problem Statement

Cramer Smith, Sam Lichlyter, Eric Winkler and Zach Schneider

October 26th, 2016

CS461: CS Senior Capstone

Fall 2016

Abstract

Developer tools are often complex pieces of software. Gathering and manipulating useful information for a programmer can often be a slow and costly process. By implementing Information Foraging Theory design patterns in the creation of these tools, the information collected may be more useful or produced faster. Information Foraging Theory is the theory and math behind the choices people make to maximize the value of the information they find versus the cost of getting that information. Our aim is to develop a tool that will act as a proof of concept to this idea and increase developer efficiency. Implementing multiple IFT design patterns, we will create a developer tool that helps enforce and maintain code structure.

I. Problem Statement

A. Problem Definition

Software developers often spend a significant portion of their time seeking information within their development environments. Tasks like re-factoring, debugging and writing correctly organized code can involve the developer spending valuable time scanning through their environments looking for examples, regions or specific functions. For example, if a developer is tasked with extending the functionality of existing code, they may have a design pattern they are expected to conform to. Finding the correct location in the project to place data access methods, GUIs, helper functions, etc. is essential in maintaining organized and understandable code. It may also take the developer a lot of time. The time spent navigating through code or seeking information in an IDE is not an efficient use of developer time and could be better spent elsewhere.

B. Proposed Solution

The proposed solution to this problem is to implement Information Foraging Theory design patterns within an IDE to assist in maintaining the organization and structure of large code bases. The solution will use a number of IFT patterns including Specification Matcher, Structural Relatedness, Impact Location, Path Search and Recollection IFT design patterns. The Specifications Matcher design pattern identifies source code files that contain valuable code snippets based on the fact that they fulfill the specifications of the developer. The Structural Relatedness design pattern would be used to identify other patches that contain valuable prey by looking at other patches that are structurally related to the current patch in which the developer has indicated interest. The Impact Location design pattern would be used to enable programmers to identify source code which is affected by the alteration of a different section of code. The notion of ”impact” is distinct from similarity, in that impact means the outcomes of the code may change, while similarity only means that the code may be related functionally. The Path Search design pattern would be used to automate searches across a path in a topology, collapsing a topology into a list of prey containing cues matching the predator’s information goal. The Recollection design pattern is used to quickly find a previously known class or method that is relevant to the task at hand. With these design patterns the IDE will be able to find and identify sections of code to help organize code within the projects scope. [1]

C. Performance Metrics

The effectiveness of this tool will be measured in two ways: if use of the tool saves developers time and if the code written by developers is cleaner and more organized than without the tool. User tests will be conducted, giving developers and computer science students a task which they must carry out programmatically. One group of users will attempt to solve the problem in the base version of a text editor, while the other will use an editor with the IFT extension installed. Success of the IFT tool will be achieved, in part, if the group that used the tool took less time overall to complete the task given to them. Additionally, code reviews will be conducted on the solutions produced by each respective test group. Success of the IFT tool will also be achieved if the code reviewers find that the solutions written with the text editor extension installed are organized more cleanly and logically that the solutions written without the extension.

REFERENCES

[1] T. Nabi, T. Sweeney, S. Lichlyter, D. Piorkowski, C. Scaffidi, M. Burnett, and S. Fleming. (2016) Information Foraging Theory. Oregon State University.

Client Signature

_____

Signature

_____

Date

Student Signatures

_____

Signature

_____

Signature

_____

Signature

_____

Signature

_____

Date