Feature-centric Environment*

David Röthlisberger, Orla Greevy and Adrian Lienhard

Software Composition Group
University of Berne, Switzerland
{roethlis, greevy, lienhard}@iam.unibe.ch

1. Introduction

The task of locating the parts of the code that are relevant to a feature in object-oriented systems is widely recognized as a non-trivial task and a body of reverse engineering research collectively referred to as *feature identification* has emerged [1, 2]. A software engineer frequently needs to understand which parts of a system implement a feature to carry out maintenance activities, as change requests and bug reports are usually expressed in terms of features [4]. The main focus of *feature identification* research to date is in a reverse engineering context. Despite the fact that research has highlighted the usefulness of *feature identification* techniques for program comprehension, very little of this effort has found its way into the software engineer's development environment.

In this paper, we demonstrate a tool providing a perspective of a system that reflects how features are implemented to support maintenance activities. By integrating this tool in a development environment we support feature understanding while performing maintenance activities. This environment, called Feature-centric Environment, compares several features visually, provides a detailed view for a single feature and integrates a code browser focusing on a single feature of a software system. All these different views are enriched with metrics, they are interconnected and the user is able to interact with them.

In the following we introduce the Feature-centric Environment and its different views.

2. The Feature-centric Environment

The Feature-centric Environment provides three different views of features: The Feature Overview, the Feature Tree and the Feature Artifact Browser. All these three views are enriched with the *Feature Affinity* metric introduced in a previous work [3]. Applying this metric guides and supports the software engineer during the navigation and understanding of one or many features. We assign a color that represents its Feature Affinity value to the visual representation of a method used in a feature . Our choice of colors correspond to a heat map, *e.g.,* colors from cyan to red.

Compact Feature Overview

The feature overview visualizes more than one feature. The software engineer can decide how many features she wants to visualize at the same time (see Figure 1 (1)). For every chosen feature, a list of all methods used in the current feature is provided. Every method is displayed as a small colored box where the color represents the feature affinity value, the list is sorted according to this metric value. Clicking on a method opens the feature tree where all occurrences of the selected method are highlighted.

Feature Tree

In the feature tree view we present the method call tree, captured as a result of exercising one feature (see Figure 1 (2)). The first method executed for a feature (*e.g.,* the "main" method) forms the root of this tree. Methods invoked in this root node form the first level of the tree, hence the nodes represent methods and the edges are message sends from a sender to a receiver. As in the feature overview, the nodes are colored according to their feature affinity value. The tree is collapsed to the first two levels at the beginning, but every node can be expanded and collapsed again afterwards. Like this, the user can conveniently navigate even large call trees of a feature. Every node of the tree provides a button to look up the method of this node in the feature artifact browser.

Feature Artifact Browser

The source artifacts of an individual feature are presented as text in the *feature artifact browser* (see Figure 1 (3)). It displays only the classes and methods actually used in the feature. Classes and methods not par-

* In *International Workshop on Visualizing Software for Understanding* (Vissoft 2007) (tool demonstration), 2007.

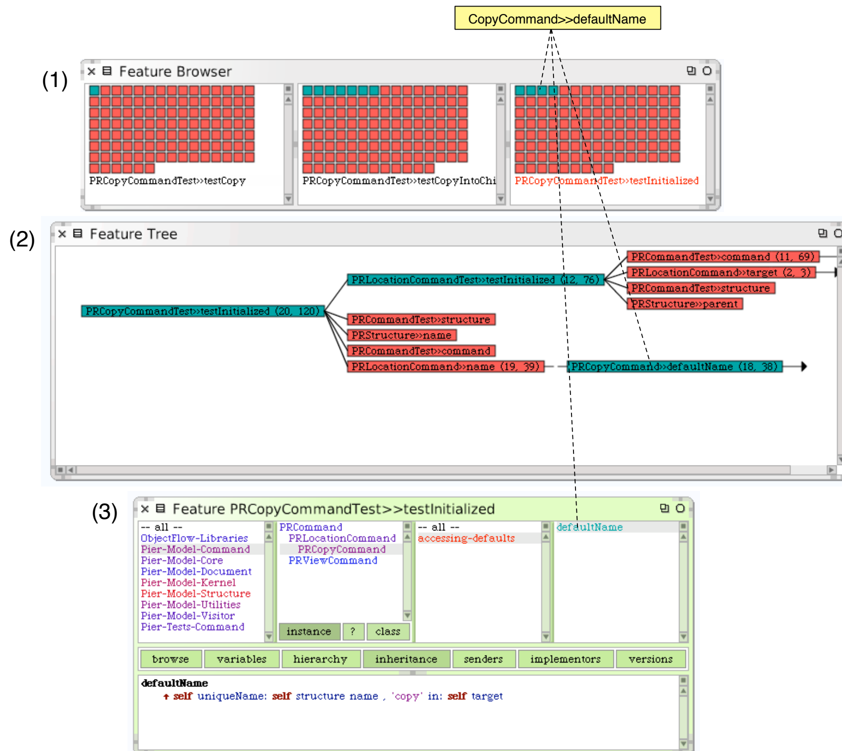

Figure 1. The Visual Components of our Feature-centric Environment

ticipating in the runtime behavior of a feature are not displayed. This makes it much easier for the user to focus on a single feature of the software. The *feature artifact browser* is an adapted version of a standard class browser available in the Squeak environment containing packages, classes, method categories and methods in four panes on the top, while the lower pane contains the source code of the selected entity. This version of the class browser not only presents static source artifacts, but also the feature affinity metric values by coloring the names of classes and methods accordingly.

3. Description of the Demonstration

The demonstration will show how a developer uses the Feature-centric Environment to perform a maintenance task, *i.e.,* correcting a defect in a software system. First, we will select some specific features of a software system to visualize them in the Feature-centric Environment. These features are normally unit test cases of a system that will be exercised to gather execution information, *e.g.,* methods being invoked. Second, we will show how to use the visualizations of these features to correct a defect. The goal is to quickly locate the method which is actually responsible for the defect. For that we will specifically use the Feature Overview to pre-choose good candidate methods to contain the defect, then we will navigate these candidate methods in the Feature Tree View to get context about how the methods are used in a specific feature. Finally, we will analyze the source code in the Feature Artifact Browser to evaluate and possibly correct a faulty method.

We acknowledge the financial support of the Swiss National Science Foundation for the project “Analyzing, capturing and taming software change” (SNF Project No. 200020-113342, Oct. 2006 - Sept. 2008).

References

[1] G. Antoniol and Y.-G. Guéhéneuc. Feature identification: a novel approach and a case study. In *Proceedings IEEE International Conference on Software Maintenance (ICSM 2005)*, pages 357–366, Los Alamitos CA, Sept. 2005. IEEE Computer Society Press.

[2] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Computer*, 29(3):210–224, Mar. 2003.

[3] O. Greevy and S. Ducasse. Correlating features and code using a compact two-sided trace analysis approach. In *Proceedings of 9th European Conference on Software Maintenance and Reengineering (CSMR'05)*, pages 314–323, Los Alamitos CA, 2005. IEEE Computer Society.

[4] A. Mehta and G. Heineman. Evolving legacy systems features using regression test cases and components. In *Proceedings ACM International Workshop on Principles of Software Evolution*, pages 190–193, New York NY, 2002. ACM Press.