Understanding Legacy Features with Featureous

Olszak, Andrzej; Jørgensen, Bo Nørregaard

Published in:
Reverse Engineering (WCRE), 2011 18th Working Conference on

DOI:
10.1109/WCRE.2011.64

Publication date:
2011

Document version
Publisher's PDF, also known as Version of record

Citation for pulished version (APA):
Olszak, A., & Jørgensen, B. N. (2011). Understanding Legacy Features with Featureous. In Reverse Engineering (WCRE), 2011 18th Working Conference on (pp. 435 - 436). IEEE Computer Society Press. https://doi.org/10.1109/WCRE.2011.64

Understanding Legacy Features with Featureous

Andrzej Olszak and Bo Nørregaard Jørgensen
The Maersk Mc-Kinney Moller Institute, University of Southern Denmark
Campusvej 55, 5230 Odense M, Denmark
{ao, bnj}@mmmi.sdu.dk

Abstract—**Feature-centric comprehension of source code is essential during software evolution. However, such comprehension is oftentimes difficult to achieve due the lack of correspondence between functional features and structural units of object-oriented programs. We present a tool for feature-centric analysis of legacy Java programs called Featureous that addresses this issue. Featureous allows a programmer to easily establish feature-code traceability links and to analyze their characteristics using a number of visualizations. Featureous is an extension to the NetBeans IDE, and can itself be extended by third-party plugins.**

Keywords-features; feature-centric analysis; modularity

I. INTRODUCTION

One of the important factors driving software evolution is the feedback from program users. It is the users who request adding, modifying and repairing functional features in a program. This in turn forces programmers to use features as the units of program comprehension and modification [1]. However, these units do not always correspond to the structural units that a program's source code is decomposed into.

Majority of object-oriented programs are not designed for locality and separation of features. This results in feature implementations being scattered over structural units and tangled with each other in terms of them. These complex relations make it difficult to comprehend and modify features locally and in separation from each other [2].

In order to facilitate and structurize the process of feature-centric comprehension of software, we created Featureous – a tool for *feature-centric analysis* of Java programs.

Featureous provides a feature-location mechanism, based on dynamic analysis, which allows one to easily establish traceability links between features and source code of a legacy program [3]. These links then become a basis for a number of analytical views depicting characteristics of feature-code correspondences in a program. The views are theoretically grounded in our three-dimensional conceptual framework built around the properties of *perspective*, *abstraction* and *granularity* [5].

Featureous is implemented as a plugin to the NetBeans IDE and is tightly integrated with existing support for Java development (i.e. project execution, code editing, code navigation). Furthermore, we provide APIs that allow for extending Featureous by third-party researchers and programmers without the need to modify the core of the tool.

II. FEATUREOUS TOOL

Feature-centric analysis is a set of techniques dedicated to investigating correspondences between user-identifiable features and legacy source code [5]. The goal of feature-centric analysis is to help identifying and understanding the implicit boundaries, overlaps and dependencies among feature implementations. In order to be feasible for large codebases, feature-centric analysis requires an efficient method of establishing feature-code traceability links.

Feature location. Featureous incorporates the dynamic feature location approach defined in [3]. For each feature of a legacy program, a programmer has to annotate its entry points – the methods through which a program's control flow enters implementation of a feature. As these methods are the only ones that need to be annotated, our approach is feasible for large and unfamiliar codebases. In rough comparison to manually locating whole feature implementations, which is reported to require on average 60 minutes per feature in *dbviz* application (18 features, 13 KLOC) [4], our approach is observed to require on average only 8 minutes per feature in *JHotDraw SVG* (29 features, 62 KLOC) [3].

After annotating feature-entry points, Featureous instruments a program with a tracing aspect at load-time. The aspect identifies methods, classes and objects used by individual features at run-time. This information is then saved as feature-trace files and serves as an input to further analysis. The captured traces are shown in the *feature explorer window* of Featureous, as depicted by (*A*) in the overview of the tool's user interface in Fig. 1.

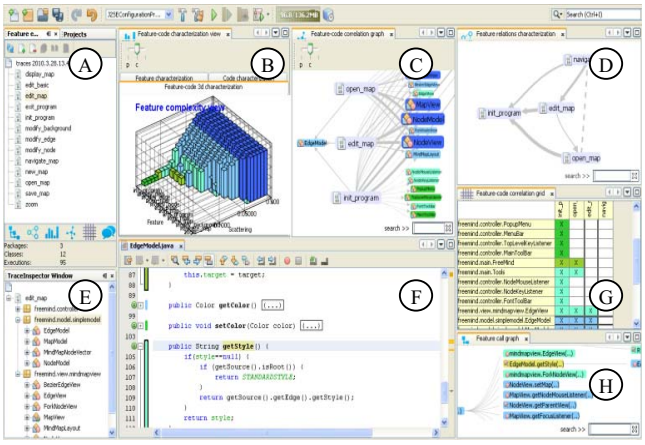

Figure 1. Overview of the user interface of Featureous.

1095-1350/11 $26.00 © 2011 IEEE
DOI 10.1109/WCRE.2011.64

Analytical views. Featureous provides a number of analytical views [5] for investigating the captured traceability links.

The *feature and computational-unit characterization* view (*B*) uses three-dimensional bar chart to visualize the *scattering* (X axis) of individual features (Z axis) and *tangling* of their computational units (Y axis). Thereby it summarizes the complexity of feature-code correspondences. Our coloring scheme assigns *green* to computational units used exclusively by a single feature, *dark blue* to core units used by most of the features and two shades in-between to any intermediate units.

The *feature-code correlation graph* view (C) and *feature-code correlation grid* view (G) correlate features and computational units. This allows for investigating how concrete structural units of a legacy program implement features. In Fig. 2, we present an example usage of the *feature-class correlation graph*, where we compare how the four decompositions of the KWIC application [8] implement features.

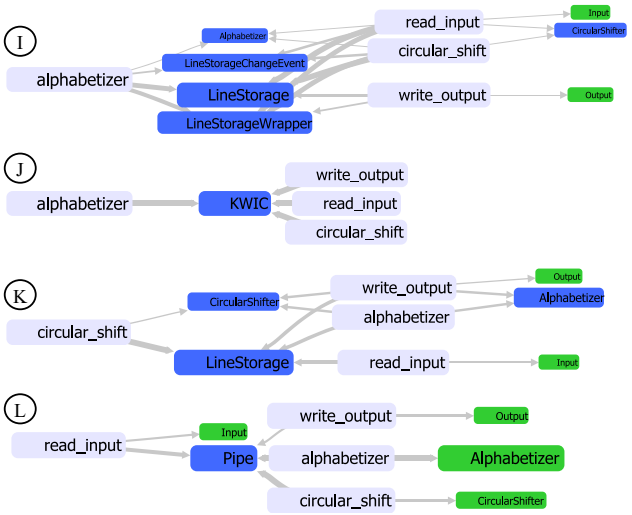

Figure 2. Feature-class correlation of four decompositions of KWIC: (I) implicit invocation, (J) shared data, (K) abstract data type, (L) pipe and filter.

In Fig. 2 we can see that the four features of KWIC ("alphabetizer", "circular_shift", "read_input", "write_output") are completely tangled with each other in the *shared data* design (*J*). In contrast, the *pipe and filter* design (*L*) grants the most independence to feature implementations, having only the Pipe class shared among features. This decomposition also exhibits lowest average values of class-granularity *scattering* and *tangling* metrics [5], i.e. 0.10 and 0.08 respectively.

The next view of Featureous called *feature relations characterization* (*D*) visualizes dynamic relations (based on object instantiation and co-usage) among features. This allows an analyst to identify run-time dependencies between features. Such dependencies are an important factor to consider when investigating deployment-time functional customizability of an application and identifying excessive dependencies not reflected in a program's problem domain.

Finally, the *feature-code traceability* (*E*), *feature call graph* (*H*) and *code-feature traceability* (*F*) views provide fine-grained navigable links between features and source code. Herein, it is possible to list and navigate units contributing to a feature's call graph, as well as features contributing to concrete fragments of source code being edited.

Experiences with Featureous. To this date, we have applied Featureous in a number of activities during software evolution.

Firstly, we experienced Featureous to be a significant aid in understanding how the features of the *SVG* application based on the *JHotDraw* framework are implemented. Featureous proved useful for both top-down and bottom-up comprehension strategies. The gained understanding of feature boundaries and relations facilitated adoption of an evolutionary change [5].

Secondly, we have applied Featureous to investigating implementations of control concerns in a component-based climate control system for greenhouses called PREDICT [6]. There, control features were implemented by so-called control components, directly reflecting the physical devices found in greenhouses. Using Featureous, we identified semantic coupling between some of the control features by investigating their co-usage of data objects.

Most recently, we have applied Featureous to guide feature-oriented restructuring and modularization of the NDVis neuroscience application [7]. The tool helped us to understand the legacy codebase and to design a remodularization process that resulted in increased independence of feature implementations and improved representation of the reusable core of the application.

III. SUMMARY

In this paper, we have presented Featureous – a tool for feature-centric analysis of legacy Java programs. We have discussed the feature location mechanism and the analytical views provided, as well as our experiences with the tool. Featureous is freely available at http://featureous.org .

Our future plans are to equip Featureous with a *feature-oriented remodularization workbench* plugin. We envision such an extension to integrate our remodularization approach [3] with the analytical capabilities of the tool to provide an interactive prototyping environment for feature-oriented restructuring of legacy Java source code.

REFERENCES

[1] C.R. Turner, A. Fuggetta, L. Lavazza, and A.L. Wolf, "A conceptual basis for feature engineering," J. Syst. Softw., vol. 49, 1999, pp. 3–15.

[2] D. Röthlisberger, O. Greevy, and O. Nierstrasz, "Feature driven browsing," ICDL '07, New York, NY, USA: ACM, 2007, pp. 79–100.

[3] A. Olszak and B. N. Jorgensen, "Remodularizing Java programs for improved locality of feature implementations in source code," Science of Computer Programming, In Press, 2010.

[4] http://www.cs.columbia.edu/~eaddy/concerntagger/

[5] A. Olszak and B. N. Jørgensen, "Featureous: an integrated environment for feature-centric analysis and modification of object-oriented software," International Journal on Computer Science and Information Systems, Vol. 6, Nr. 1, 2011, s. 58-75.

[6] http://ecosoc.sdu.dk/coe/PREDICT

[7] http://ndvis.sourceforge.net/

[8] D. Garlan and M. Shaw, "An Introduction to Software Architecture," Technical Report, Carnegie Mellon Univ., Pittsburgh, PA, USA, 1994.