Softwarenaut: Exploring Hierarchical System Decompositions

Mircea Lungu, Michele Lanza
Faculty of Informatics, University of Lugano, Switzerland
mircea.lungu@lu.unisi.ch, michele.lanza@unisi.ch

1. Introduction

Softwarenaut is a tool aimed at top-down exploration of large software systems. Using it, the reverse engineer can obtain various architectural views of a system by interactively navigating a hierarchical decomposition of the system. In order to accomodate many possible decompositions of a system Softwarenaut was designed as a framework which provides visualization and exploration services that can be applied on various hierarchical decompositions of the system.

2. Exploration mechanisms

There are multiple hierarchical decompositions of a system and for most of them the exploration mechanisms which are needed do not depend on the decomposition type. From the mechanisms that Softwarenaut employs we mention the exploration primitives, the multiple perspectives and the visual annotations.

Multiple perspectives. Softwarenaut provides three coupled, complementary perspectives on the analyzed systeem. As Figure 1 shows, the perspectives are:

- *Exploration Perspective.* Presents a graph-like representation of the visible modules (nodes in the graph) and their interaction (edges in the graph).
- *Detail perspective.* Offers detail on demand for the current selected entity by providing alternative views on it.
- *Map Perspective.* presents the positions in the hierarchy of the visible modules in order to offer a sense of context and orientation.

Navigation Primitives. There are three principal operations that the framework provides during the navigation.

- *Expand.* By expanding a node the view is updated and the node is replaced with nodes representing its children.

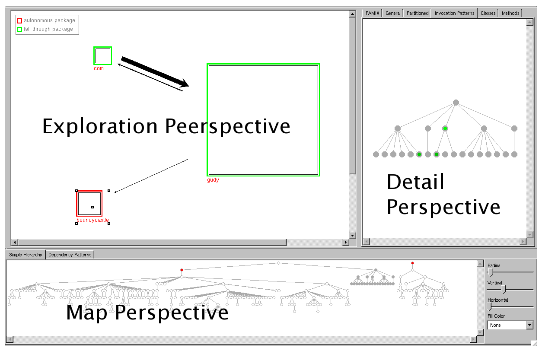

Figure 1. Package structure of Azureus explored with Softwarenaut

- *Collapse.* By collapsing a node corresponding to a package, the node, together with all the nodes representing the siblings of the package are removed from the view and replaced with a node representing the parent package.
- *Filter.* Because filtering is an important part of exploration, Softwarenaut implements multiple types of filters.

Visual Queries for Annotating Navigation. The framework provides a mechanism for specifying queries which have visual results used to annotate the elements in the view. One application of the visual query system is providing information on the most interesting exploration operations recommended at a given point in exploration.

3. Types of Analysis

Depending on the available data, on the desired depth of the analysis and on the purpose of the reverse engineering process, various types of analysis are posible. In this section we briefly present several types of analysis that Softwarenaut can perform.

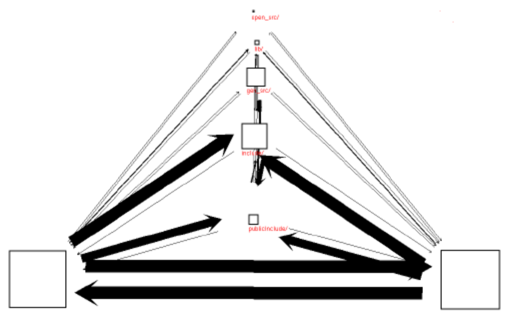

Figure 2. Exploration Perspective in one industrial case study.

Package Dependency. When analyzing a Java system, the modularization units with the highest granularity are the packages. Because even for a medium-sized software project, a package dependency graph is overloaded with information, Softwarenaut makes use of the implicit convention used by developers of grouping the related packages together in superpackages. Based on this assumption, the relations between the lower level packages can be aggregated to higher levels obtaining views which present less information but can be iteratively refined.

Figure 1 presents Softwarenaut exploring the package hierarchy of Azureus, an open source file sharing system of approximately 2500 classes.

Directory Include Relationships. For C/C++ projects an initial overview of the system can be obtained by considering the include relationships between files aggregated at directory level. This type of analysis is important when in analyzing very large industrial systems because the data can be obtained by a lightweight parsing of the system.

Figure 2 presents the Exploration View of Softwarenaut exploring a large industrial C/C++ system (more than 2MLOC) . The nodes represent modules (directories) in the system, and the edges represent include relations aggregated from the contained files. The area of the nodes is proportional to the corresponding module's size and the width of the edge is proportional to the number of dependencies abstracted in it.

With these visual conventions in place, the figure shows that, besides having similar size, the two bottom modules interact in a symmetric way with the other modules in the system. A reverse engineer might want to understand the causes of such a similarity[1].

1 Starting from this view, the authors discovered that the files contained in the two modules were duplicated. Note that the visual conventions were essential in detecting the symmetry

Semantic Cluster Interaction. Latent Semantic Indexing (LSI) is an information retrieval technique used primary in web search. By using LSI to compute the similarity between documents, Kuhn et. al [?] create a hierarchical clustering of the classes in the system on a semantic base. Their aim is to identify concepts in the system.

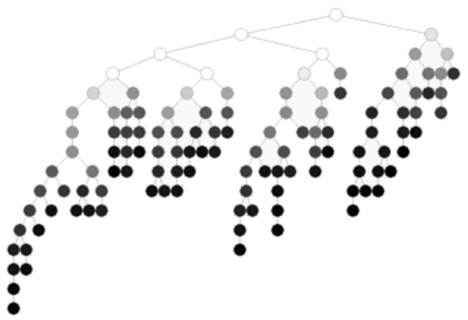

Figure 3. Map Perspective representing a hierarchical clustered decomposition. The more semantically cohesive a cluster, the darker its color

Because the concept detection process can not be automated the solution is to manually inspect the cluster hierarchy. In our previous work [?] we how exploration with Softwarenaut is useful in detecting concepts in a semiautomatic way in a system.

4. Presentation Description

During the presentation, we will interactively reverse engineer a software system using Softwarenaut. The system will be a large (> 1000 classes) software system. We believe that doing the analysis in an interactive way and involving the participants will provide us with important feedback on the strong and weak points of the tool and the methodology.

Softwarenaut is open source software, written in Smalltalk. The tool is available for all the major platforms (Windows, Linux, OS X) and can be freely downloaded from: `http://www.inf.unisi.ch/phd/lungu/softwarenaut` —