

Graph-Centric Tools for Understanding the Evolution and Relationships of Software Structures

Yijun Yu Michel Wermelinger

Computing Department & Centre for Research in Computing
The Open University, UK

Abstract

We present a suite of small tools, implemented as a pipeline of text file manipulating scripts, that, on one hand, measure the evolution of any software structure that can be represented as a directed graph of software elements and relations and, on the other hand, visualise any three attributes of any set of software artefacts that can be related to the elements shown in the graph. We illustrate the applicability of the tool with our work on the evolution of the Eclipse architecture and the relation between bugs and components.

1. Introduction

Due to their generic and flexible nature, graph-based representations have been very popular in the reverse engineering literature and tools in order to represent software elements (e.g. functions) and their relations (e.g. calls between functions). In our own work we use graphs to represent the architecture of Eclipse: nodes represent plugins and arcs represent either a compile-time or a run-time dependency.

To support our research for understanding the evolution and relationship of software structures, we developed a suite of small tools that first extract the necessary data, then compute the necessary metrics (e.g., size, fan in/out), and finally visualise the results. However, we have taken care to make the suite relatively independent of our particular needs, in order to be useful to other researchers in a variety of contexts. Therefore, instead of developing the suite as a standalone application or as an extension for a particular IDE, we have adopted a simple pipeline architecture of scripts that manipulate text files. This makes it easier for researchers to interface their own tools with ours and to replace part of the pipeline in order to better suit their needs.

The overall architecture of our tool suite is shown in Figure 1 as a set of processes that convert input data files on the left into the output data files on the right. Among the processes, *fact extractors* obtain factual relations from artifacts of a single release of the software system and store

the relations in Rigi Standard Format (RSF) files. Using customised Crocopat scripts (*.rml), *fact mergers* combine facts about selected individual releases into a single fact base by expanding every relation tuple with an attribute of the release id. *Metric calculators* compute from the fact base a number of metrics, such as growth, volatility, etc. The *reporters* present the metrics and the architecture in a number of ways, including various visualisations. Finally, the *bug analysers* extract, merge, calculate and present the bug reports on top of architectural graphs.

Our tools can be downloaded from <http://computing-research.open.ac.uk/sead/archev>, including sample projects and datasets. The demo will emphasize the tool suite's architecture and what relations are described in each RSF file, so that the audience can understand how to adapt our tools to their needs.

2 Examples

We have reported the utility of these tools using Eclipse as a case study: in [1] we analysed how the architecture evolved, e.g. if architectural changes are confined to major releases, while in [2] we checked whether certain design principles apply over the life-time of Eclipse's architecture, e.g. if cohesion among components increased.

We extract one architectural graph per release (from 1.0 till 3.3.1.1), using Eclipse's XML metadata files about its plugins. This is sufficient for our purposes, besides being much more efficient than parsing source code.

In Eclipse, a module can be a single plugin or a component (group of plugins). The architecture is defined as a graph of compile-time or run-time dependencies between modules. Since any graph is represented by a binary adjacency relation in a RSF file, we wrote a simple converter to show a static graph in a tool like graphviz. However, we can also make use of animated graphs to show the evolution of the architectures throughout releases, using graph exploration systems such as Guess and CCVisu.

We also wish to see how bugs are related to components, e.g. which components have more critical bugs, and how

