Understanding Code Architectures via Interactive Exploration and Layout of Layered Diagrams

Vineet Sinha* Elizabeth L. Murnane* Scott W. Kurth§ Edy S. Liongosari§ Rob Miller David Karger

*Architexa {vineet, emurnane}@architexa.com
§Accenture Technology Labs {scott.kurth, edy.s.liongosari}@accenture.com
MIT CSAIL {rcm,karger}@mit.edu

Abstract

Visualization tools that target helping developers understand software have typically had visual scalability limitations, requiring significant input before providing useful results. In contrast, we present Strata, which has been designed to actively help users by providing layered diagrams. The defaults used are based on the package structure, and user interactions can allow for overriding these defaults and focusing on relevant parts of the codebase.

Categories and Subject Descriptors: D.2.2 [Design Tools and Techniques]: User interfaces; D.2.6 [Programming Environments]: Graphical environments, Interactive environments.


Keywords: Program Comprehension; Software Architecture; Reverse Engineering; Automated Software Engineering.

1. Introduction

With the growth in the size and complexity of software projects, teams are facing a number of challenges in understanding and getting a grasp of such projects. While design documents can help in providing an overview of these projects, such documents are often outdated, incomplete, or simply do not exist. Even though creating an applications’ architecture efficiently is expected to promote effective architecture reviews and evaluations [1], having the architecture available and synchronized with the application implementation is currently difficult and costly [3].

A common solution to such difficulties is to use tools for automatically generating such diagrams from the code itself, but generated diagrams are often not intuitive and have scalability limitations, especially in tasks related to comprehension [1]. While the creation of high-level overviews of a project typically needs manual input to improve their quality, available tools require expert assistance in providing any form of architectural diagrams and additionally only provide limited assistance in creating diagrams focusing on different aspects of the implementation.

In this demonstration we present Strata, which builds high-level diagrams of dependencies in software projects and actively helps developers in exploring, understanding, and getting an overview of the underlying project. Strata builds a layered visualization representing the entire project. In doing so it uses the project’s package structure as a default for dependency and module aggregations. Dependency cycles are also broken to maximize the number of dependencies going downwards (as demonstrated to be effective [5]). Since such guesses by the tool can be incorrect, simple interactions with the diagram allow the user to change the layout. Further, with such guesses, a developer can now use Strata to interactively either focus on a relevant portion of the project or remove irrelevant portions. Developers can explore and find relevant (potentially crosscutting) concerns within the implementation and use them as modules for future explorations.

The focus of the approach is on providing a mechanism to get a rapid high level visualization without developer intervention and on providing mechanisms for users to find relevant portion of the code that they might be interested in. By providing a lightweight method for developers to define and use modules, developers can use Strata to understand code and describe this understood knowledge to Strata for use in enhancing the default visualization.

2. Walkthrough

Consider a developer working on the JEdit project [2]. JEdit is an open-source Java based editor consisting of a fairly extensive plugin and scripting framework with over 500 classes and 150,000 lines of code. When working with such projects it is difficult to find a starting point in the code and to get an overview of the various components. Strata provides support for such situations by allowing developers to explore through overview visualization of the code. The developer needs to only right-click on the project in the Eclipse IDE and open Strata from the context menu, which presents him with a view similar to Figure 1.

Figure 1. Top level view when a developer opens Strata on the entire JEdit codebase. (left: the default view, and right: when mouse is moved over the org package)
The figure is designed to look similar to commonly created layered architecture diagrams. It shows that the top-level project consists of a number of modules with the org module being the largest. Strata shows the size of modules in tooltips, and the developer will likely realize that some of the modules like installer, gnu and com consist of fewer classes in the project. Strata’s layered view puts each module in a layer that depends on one or more modules in layers below it. In this case, the org module depends on a module below it, and moving the mouse over the org module shows arrows indicating dependencies on a number of modules. An experienced Java developer will likely recognize that the module bash is likely related to scripting support, the installer module is likely related to an installer for the editor, and the com and gnu modules likely consist of overrides to the externally provided functionality provided in these modules. Since developers are mostly interested in the code for the current project, Strata by default shows only the dependencies in the provided source, and does not include code provided in external libraries.

At this point the developer can further continue his exploration. He can choose to select and remove the smaller modules from the view. Doing so results in Strata automatically expanding the org module, showing that it consists of the gjt.sp module and the objectweb module. Again, the module names are not very useful, but they represent the best guess that Strata starts with, and in such cases just noticing the size of a module can recommend an exploration path to the developer. Strata again decides that there are few modules being shown to the developer and expands the largest module – the gjt.sp module. This module consists of the jedit and the util module, and the jedit module is further automatically expanded to give Figure 2.

Looking at the figure, a few quick observations can be made: the objectweb module does not seem to have any dependencies to or from it to the rest of the shown code, the util module that is below the jedit module likely has a number of code elements depending on it, and the jedit module consists of a number of modules dealing with, among other things, the gui, a textarea, and search support. Mousing over the diagram shows the presence of a few upward arrows indicating the dependency cycles among these modules and shows that the involved modules are possibly working closely with each other.

Again, the developer can decide that certain dependencies represent minor temporary inconsistencies with the design, and he can choose to ignore them and ask the tool for an updated layout or can continue exploring with the tool by asking it to remove some of the modules.

Once at a detailed enough exploration level the developer can ask Strata to show the dependencies in the code or to view the source of the involved modules. Beyond diving into details of modules, a developer can explore in a number of intuitive ways. He can select a grouped dependency and ask to expand the modules that are part of the dependency. Further, a developer can also select a module and ask Strata to show all modules that either depend on it or that the selected module depends on.

3. Implementation

Strata is an Eclipse plug-in and has been built on top of the Relo project [4]. Strata was originally developed as Snapshots at Accenture Research, then refined at MIT CSAIL, and has since been re-implemented by Architexa as part of its Source Explorer.

References