Poster Abstracts at Eurographics/ IEEE-VGTC Symposium on Visualization (2010)
D. Auber, G. Melançon, T. Munzner, and D. Weiskopf (Chairs)

SolidSX: A Visual Analysis Tool for Software Maintenance

D. Reniers[1] and L. Voinea[1] and A. Telea[2]

[1]SolidSource BV, Eindhoven, the Netherlands [2]University of Groningen, the Netherlands

Abstract

We present SolidSX, an integrated tool for visual analysis of large software systems. SolidSX integrates static code analysis (parsing and metric computation) and multiple linked views such as treemaps, table lenses, and hierarchical edge bundles in a single environment, thereby simplifying the work of software developers interested in correlating several structure and metric aspects in understanding large software projects. We outline the features of SolidSX which make it an effective instrument in contexts where tool integration and ease-of-use are key requirements.

1. Introduction

Software maintenance accounts for over 80% of the costs of the lifecycle of modern software systems. Of these, over 40% represent understanding the software. Many visualization tools support software understanding by techniques such as compound graph layouts for hierarchy-and-dependency data; treemaps and tables for software quality metrics; and annotated text views for source code. Yet, most such tools know very limited acceptance in the software industry. Key reasons for this are limited scalability in terms of visualization and/or size of handled code bases, long learning curves, and poor integration with software analysis and development toolchains.

We present here SolidSX, a visual tool for software understanding in maintenance which attempts to remove the above-mentioned difficulties. SolidSX follows a visual analytics approach. It tightly integrates several visual techniques (hierarchical edge bundles (HEBs), treemaps, table lenses, and annotated code views) with several reverse-engineering and analysis techniques (code parsers and code quality metric engines) in a single environment. We present next the main features of our tool and outline the design decisions taken which ensure the scalability, ease of use, and integration requirements.

2. Data Acquisition and Analysis

SolidSX uses a simple dataflow architecture (Fig. 1, for details see the tool's manual [Sol09]). Input source code, .NET assemblies, or Java bytecode, are processed by several built-in analyzers to extract a compound (hierarchy-and-dependency) attributed graph. Nodes model software artifacts: modules, assemblies, files, directories, classes, and functions. Edges model both aggregation (containment) and a wide range of dependency types: calls, type uses, variable reads/writes, inheritance, interface implementation, and header/package inclusion. Nodes and edges may have a variable number of key-value attributes, *e.g.* names, types, signatures, and software metrics. No restriction is put on the graph's structure or attributes, *e.g.* several hierarchies can co-exist. Extracted data is persistently stored in a simple, fixed-schema, SQLite database or XML file, which allows fast retrieval, user-defined searching, and interoperability with other tools.

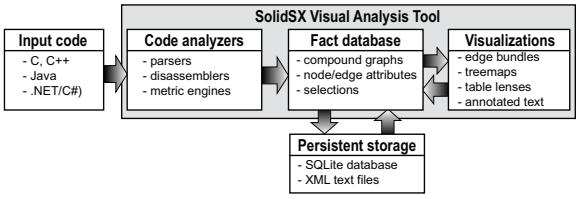

Figure 1: *SolidSX tool dataflow architecture*

Setting up static code analysis is notoriously complex, error-prone, and time consuming. We succeeded in *completely* automating this process by using customized versions of several static analyzers: Recoder (for Java) [Lud09], Reflector (for .NET/C#) [Red09], and our own SolidFX parser (for C/C++) [TV08]. Besides structural information, we modified these analyzers to compute code quality metrics, such as complexity, cohesion, coupling, fan-in, and fan-

out. This allows end users to generate visualizations combining structure, dependency, and quality metrics data in a matter of seconds up to minutes from code bases of up to hundreds of thousands of lines by simply providing an input directory with source code. This design decision proved key to user acceptance.

3. Visualizations

SolidSX offers several views (Fig. 2 top): classical tree browsers, table lenses of node/edge attributes, treemaps, and the novel HEB compound graph layout [Hol06]. All visualizations have carefully designed *presets* which allow one to use them with no additional customization, and are implemented in C++ using OpenGL 1.1. They all depict the same fact database created by the code analysis step. Users can also create node/edge *selections* in any view by either direct interaction or custom queries (implemented as SQL/XML queries). These two mechanisms realize the linked view concept, which enables users to easily create complex analyses of correlations of structure, dependencies, and metrics along different viewpoints. Figure 2 top) illustrates this on a C# system of around 45000 lines of code (provided with the tool distribution [Sol09]). The HEB view shows function calls over system structure: caller edge ends are blue, callee edge ends are gray. Node colors show McCabe's code complexity metric on a green-to-red colormap, thereby enabling complexity correlation with the system structure. We see that the most complex functions (warm colors) are in the module and classes located top-left in the radial layout. The table lens view shows several function-level code metrics, and is sorted on decreasing complexity. This allows one to see how the different metrics correlate with each other. Alternatively, one can select *e.g.* the most complex or largest functions and see them highlighted in the other views. The treemap view shows a flattened system hierarchy (modules and functions only), with functions ordered top-down and left-to-right in their parent modules on code size, and colored on complexity. The visible 'hot spot' indicates that complexity correlates well with size. Constructing the *entire* scenario, including the code analysis, takes about 2 minutes and under 20 mouse clicks.

4. Integration in Development Pipelines

Arguably the most important feature for user acceptance of SolidSX is integration ease: The tool comes can be used fully standalone on C, C++, .NET/c#, and Java code bases, but is also integrated in the Visual Studio IDE (Fig. 2 bottom). The latter links code editors and views in both ways by mouse clicks, and is one of the first (and few) examples of truly integrated visual analytics solutions in software development. The tool provides a simple Python scripting interface which allows integration in other IDEs or toolchains, such as Eclipse, KDevelop, or Qt Creator. The open SQL and XML data interchange formats further simplify integration at data exchange level. SolidSX was used in several

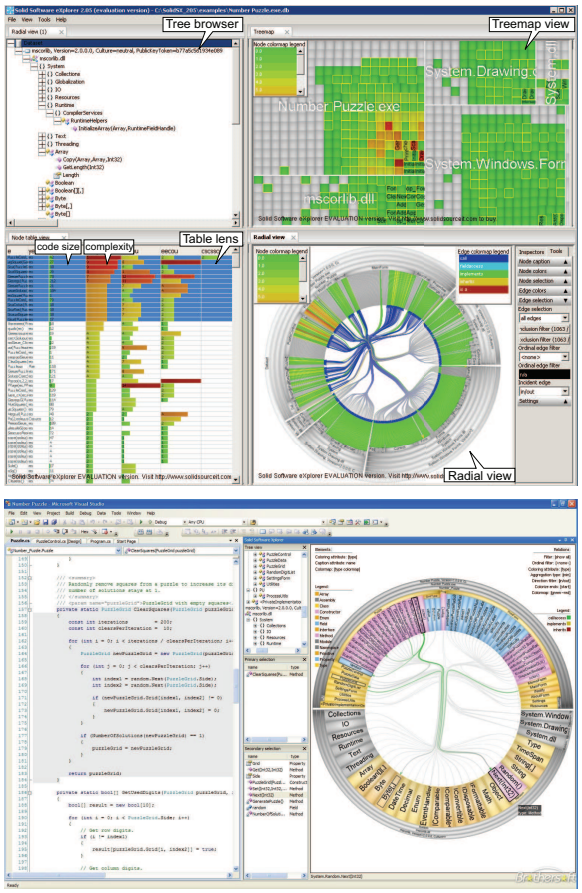

Figure 2: *Top: SolidSX views; Bottom: Visual Studio integration*

industrial reverse-engineering and program comprehension projects, as described on the tool's webpage [Sol09]. Ongoing work includes integrating multiscale bundling features, a recent research result [TE10], in the HEB view. SolidSX is freely available for academic and research users.

References

[Hol06] HOLTEN D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. *IEEE TVCG* (2006), 741–748.

[Lud09] LUDWIG A.: Recoder java analyzer, 2009. http:recoder.sourceforge.net.

[Red09] REDGATE INC.: Reflector .NET api, 2009. http:www.red-gate.com/products/reflector.

[Sol09] SOLIDSOURCE: SolidSX software explorer, 2009. http:www.solidsourceit.com/products/SolidSX-source-code-dependency-analysis.html.

[TE10] TELEA A., ERSOY O.: Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum (Proc. EuroVis'10) 29*, 3 (2010). to appear.

[TV08] TELEA A., VOINEA L.: SolidFX: An integrated reverse-engineering environment for c++. In *Proc. ACM SOFTVIS* (2008), pp. 165–172.