Leveraging Visio for Adoption-Centric Reverse Engineering Tools

Qin Zhu Yu Chen Piotr Kaminski Anke Weber Holger Kienle Hausi A. Müller
Department of Computer Science, University of Victoria
{qinzhu,yuchen}@csc.uvic.ca *{pkaminsk,anke,kienle,hausi}@cs.uvic.ca*

Abstract

There are many reasons why reverse engineering research tools often fail to be evaluated or adopted in industry. Their rough user interfaces and poor interoperability are just two frequently mentioned issues. The aim of the ACSE (Adoption-Centric Software Engineering) project, conducted at the University of Victoria, is to investigate how some of these impediments can be overcome by building software engineering tools on top of Commercial Off-The-Shelf (COTS) products.

This paper outlines how to leverage Microsoft Visio for a software visualization and metrics tool. Software developers familiar with Visio only have to learn the reverse engineering specific functions introduced by our tools and can take advantage of their existing, domain-independent Visio knowledge. Thus, compared to a stand-alone application, this Visio-based tool leverages the cognitive support previously acquired by developers using Visio.

1. Introduction

As software systems grow older, software engineers increasingly find themselves faced with maintaining, extending or re-engineering largely undocumented legacy applications. Before any of these tasks can be attempted, an understanding of the software at a higher level of abstraction must be gained. Reverse engineering tools can help with this challenge, yet sport a surprisingly low adoption (and even evaluation) rate in the software industry. This low penetration can be partially attributed to a lack of polish in user interfaces and weak support for integration with other tools, comparing poorly to mainstream packages such as the Microsoft Office suite.

One way to overcome these impediments is by grafting reverse engineering functionality onto entrenched tools, a facet of an approach dubbed adoption-centric reverse engineering [1]. The base tools could be integrated software development environments (IDEs), but in general need not be related to programming, and include such applications as editors, shells, browsers, word processors, and personal information managers [2]. Some of these tools offer built-in extension facilities or development kits, making it easier to repurpose them.

One such flexible tool is Microsoft Visio. It is a member of Microsoft's Office suite so users can embed Visio drawings into other documents, customize its user interface in standard ways, and benefit from a widespread support base for that application. Visio is particularly well suited to our endeavor since it is built around a powerful diagramming engine to which it provides full programmatic access, though other office applications have their own attractions [1]. In this paper we illustrate how domain-independent, native Visio operations (such as panning, zooming, and automatic layouts), integrate seamlessly with end-user programmed, domain-specific operations (such as filtering or metrics), to provide reverse engineering functionality.

The paper is organized as follows. We describe Visio and its customization facilities in Section 2. In Section 3 we show how REVisio, our reverse engineering application, takes advantage of Visio's features to simplify development and lower the adoption barrier. Section 4 concludes the paper.

2. Microsoft Visio

Microsoft Visio, a member of the Microsoft Office suite, is an advanced drawing tool for all kinds of diagrams: flowcharts, block diagrams, building plans, maps, etc. While users can create diagram from scratch, Visio also provides a number of predefined templates that reconfigure the application for specific diagram types. Templates usually include stencils (though the latter can also be loaded independently), which are coordinated collections of master shapes. Master shapes are presented in a palette to be dragged by the user onto the canvas automatically instantiating the corresponding shape, allowing the users to focus on the content of the diagram rather than their drawing skills (or lack thereof). Visio also provides a variety of other shortcuts, wizards and navigation tools that further ease the user's cognitive workload.

2.1. Customization Options

Visio allows users (and developers) to customize its operation on many levels [3][4]. At the most basic level, users can employ their own masters, stencils and templates to customize the work environment. Fundamental drawing and transformation tools can be used to create or modify shapes, controlling their geometry, color and style. Any shape can then be promoted to a master and included in a stencil or template.

For more advanced customization, Visio exposes the masters' ShapeSheets. A ShapeSheet is a special-purpose spreadsheet whose cells are connected to the shape's properties, giving full control over the shape's geometry and behavioral constraints. ShapeSheets support formulas allowing complex relationships to be formed between cells, and provide limited access to certain operations defined on the Visio object model. However, each ShapeSheet is attached to a single shape, so it is impossible to write formulas that take other shapes on the canvas into account.

Finally, Visio is an automation server, exposing its object model for other software to use. The object model encompasses almost all data in Visio, including canvas and shape information, ShapeSheets, menus and dialogs, giving client applications full control over all aspects of Visio. They are able to modify existing elements in any way, as well as add new ones, or simply access information about the canvas' current contents. One popular use of automation is to construct a diagram according to some higher-level information provided by the user.

Automation clients can be written in any language that supports automation (or COM), including Visual Basic for Applications (VBA), Visual Basic (VB), C, and C++. Visio includes a VBA development environment and VBA macros can be embedded directly into Visio documents, thus simplifying both development and distribution.

A Visio solution brings together all the customized items that may be needed in the pursuit of a specific diagramming goal. Solutions combine templates, stencils, masters, customized ShapeSheets and various scripts and macros that use the automation facilities [5]. A typical solution maintains the original Visio paradigm of drag-and-drop diagram assembly from a palette of master shapes, while augmenting the shapes with "smarts" appropriate to the specific application. Users can indirectly affect the solution's shapes' properties through menus or data entry to automate the tedious or complex manual manipulation that would normally be required.

3. REVisio

REVisio is a partial adaptation of Rigi [6] (augmented with metrics) as a Visio application, in pursuit of the adoption-centric approach to reverse engineering tools. Rigi is a reverse engineering tool that extracts information from software source code, displays it in an interactive graph-like visualization, and allows it to be manipulated (manually or automatically) to discover the software's structure. Rigi has been under development at the University of Victoria for over a decade and is a feature-rich tool, yet its rate of adoption in the industry is lower than might be expected.

Visio, on the other hand, is widely used by software developers for design and analysis tasks. Our hypothesis is that those developers would hence be more likely to evaluate and adopt a Visio reverse engineering application due to its ease of installation and lower learning curve. Building on Visio also allowed us to shorten the development time of REVisio by reusing Visio's extensive diagramming facilities.

This section discusses how REVisio acquires information about software systems, visualizes the software's structure, and calculates and displays metrics. All application functions are implemented using a combination of automation, ShapeSheets and customized masters, and can be controlled from a customized tool-bar that integrates smoothly with Visio's user interface (Figure 2).

3.1. Data Import and Statistics

While REVisio is a Visio solution, it does not follow the typical drag-and-drop process for creating diagrams. Rather, drawings are initially automatically derived from imported data, and can then be navigated and modified by users with Visio's usual assortment of tools. The base data is first extracted from source code with established Rigi parsers and saved in Rigi Standard Format (RSF), Rigi View Graph (RVG) or Graph Exchange Language (GXL) files.

When a model is read in, REVisio presents the user with some statistics allowing them to estimate the com-

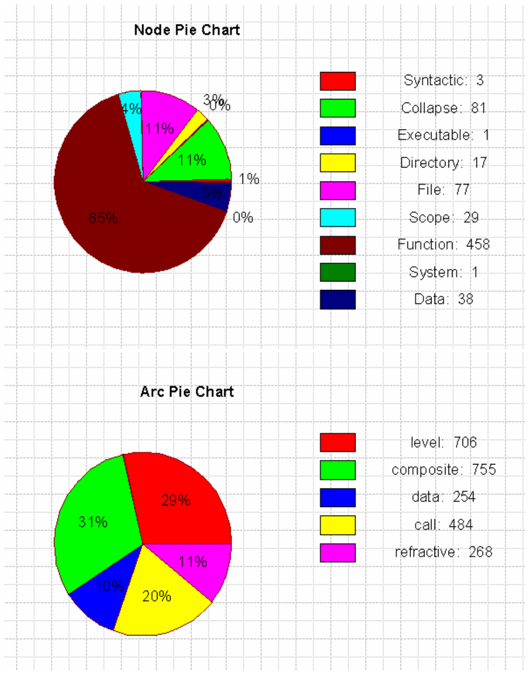


Figure 1. Node and arc pie charts

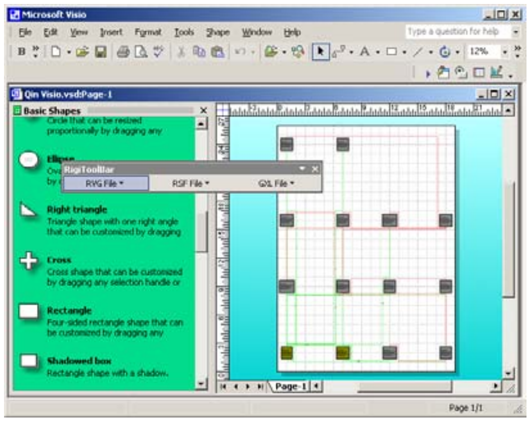
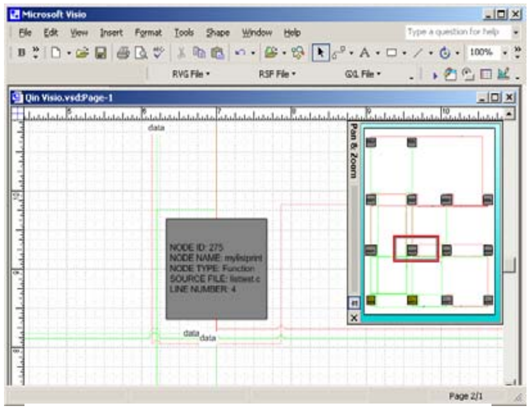


Figure 2. REVisio showing a Rigi graph

plexity of the system. Rigi models are attributed typed directed graphs. Components of the model are represented as nodes (or vertices), and relationships between components are represented as directed edges (or arcs). A simple example is a call graph of a software system: procedures are reified with nodes and calls between procedures are represented with arcs. To provide an overview of the graph's contents, REVisio displays annotated pie charts that show the total numbers of nodes and arcs of different types as well as their relative proportions (Figure 1). From these charts, the user can quickly intuit the model's size, the number of different types used in the model, and which ones are used most often.

3.2. Structure Visualization

Beyond these simple statistics, REVisio can display the graph read in from an RSF, RVG or GXL file. RSF files hold only structural information extracted from the target system, while RVG and GXL files are often augmented with view-specific attributes (e.g. displayed node location and color) that REVisio tries to preserve as much as possible. The nodes and arcs are shown using Visio shapes and dynamic connectors, Visio's automatically



NODE ID: 275
NODE NAME: mylistprint
NODE TYPE: Function
SOURCE FILE: listtest.c
LINE NUMBER: 4

Figure 3. Pan & Zoom tool in action

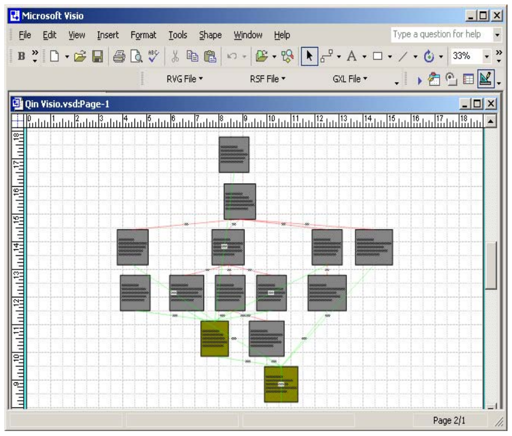


Figure 4. Tree layout of a small graph

routed rectilinear paths (see Figure 2). The resulting diagram can be extended and manipulated with all the standard Visio tools; two particularly useful ones are pan & zoom and automatic layout.

Visio's pan & zoom feature, shown in use in Figure 3, provides the user with a context view that lets them focus on diagram details while maintaining an overview of the structure. The context view also gives the user direct control over the focus of the zoomed-in view, allowing them to navigate large diagrams with ease.

Visio can also automatically lay out graphs according to a number of common algorithms, improving the organization of complex diagrams. For example, Figure 4 shows the previous diagram laid out in a tree style using straight connectors, with the page automatically resized to hold the resulting drawing. Larger graphs can be automatically laid out as well (see Figure 5), but the operation is not every efficient and its running time may quickly become prohibitive. (The graph in Figure 5 took about 10 minutes to lay out on a reasonably equipped machine, but we have not yet made detailed performance measurements or attempted to optimize the operations.)

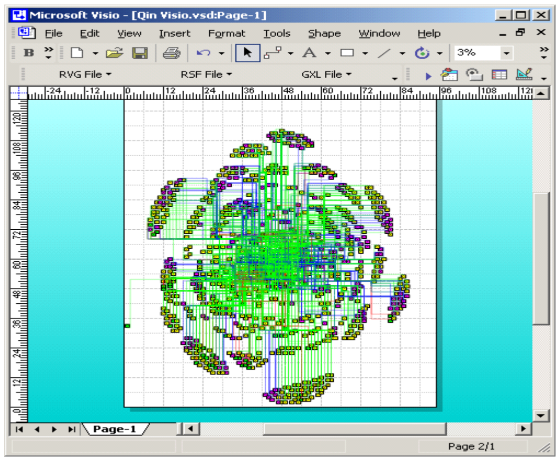

Microsoft Visio - [Qin Visio.vsd:Page-1]
File Edit View Insert Format Tools Shape Window Help
RVG File
RSF File
GXL File
Page-1
Page 2/1

Figure 5. Radial layout of a large graph

Inheriting these familiar diagram tools from Visio simultaneously makes REVisio easier to learn and use and shortens its development time. However, to support reverse engineering, REVisio introduces a few custom operations accessible through a toolbar menu. We allow the user to select nodes by type and to selectively hide them to reduce clutter. We also provide basic graph traversal operations, highlighting all parents or children of a given set of nodes based on a given relationship type.

These rudimentary functions are obviously inadequate for all but the simplest reverse engineering tasks. They serve merely as a proof of concept, to demonstrate that it is easy to script automated operations on the graph. We expect users to be familiar with Visual Basic and to write their own ad hoc scripts as the need arises.

3.3. Metrics Visualization

REVisio counterbalances the often overwhelming detail provided by software structure graphs with a variety of software metrics. We focus on classic metrics that produce numbers characterizing properties of software code [7]. A good metric can aid a reverse engineer to better assess certain characteristics of the subject software system and decide where to focus their attention. For example, metrics can indicate which parts of a software system have a high complexity or tight coupling.

We have initially implemented four metrics in REVisio:

- LOC (Lines of Code) counts the lines of code for each method or class.
- NMC (Number of Methods per Class) counts the number of methods for each class, thus indicating a class's size.
- CBO (Coupling Between Objects) counts the number of foreign classes referenced, either through field access or method calls. A large CBO value indicates that a class is highly dependent on other classes. [8]
- RFC (Response For a Class) extends the CBO by counting the number of foreign methods invoked by a class, and its value may therefore exceed the CBO value [8].

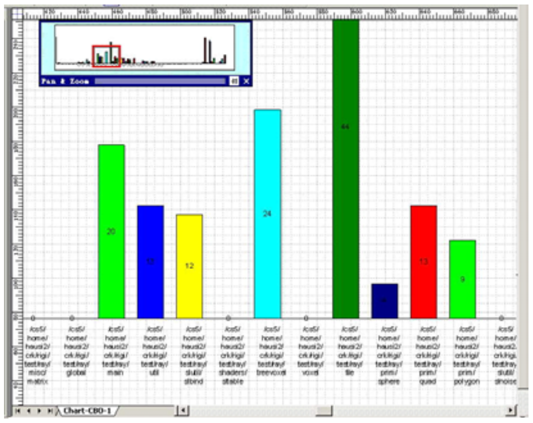


Figure 6. CBO bar chart

REVisio computes and displays the metrics based on the structural information read in from an RSF file. Figure 6 shows a sample CBO bar chart, with the class names on the X axis, and the CBO values on the Y axis. The chart was quickly implemented by scripting existing Visio masters and takes advantage of Visio's pan & zoom feature to provide an overview and quick navigation. Other metrics, even ad hoc ones customized to the user's target system or tasks, should be equally easy to implement.

4. Related Work

Our proposed approach is to craft software engineering functionality on top of COTS products that offer end-user programmability through either a scripting language or a more general automation API. In related projects in our group we extend Lotus Notes and PowerPoint to visualize and manipulate Rigi graphs [1]. Other research has leveraged COTS products as well to build software engineering tools.

Desert is an open tool environment consisting of several loosely coupled components [9]. One of these components is a specialized editor for source code and architecture documentation. This editor is based on Adobe FrameMaker and uses a wide variety of FrameMaker's functionality, such as syntax highlighting with fonts and colors, graphic insets, and hypertext links. FrameMaker is extended via the Frame Developer's Kit API.

Riva and Yang have developed a software documentation process that uses Rigi to visualize software artifacts [10]. They used Rigi's scripting capabilities to export this information to Visio as a UML model. The exporter writes files in Visio's XML vocabulary employing predefined UML master shapes. The authors take advantage of Visio's ability to produce HTML renderings to web-enable the documentation. It is interesting to note that their approach uses Rigi's scripting to write Visio XML files whereas our approach uses Visio's scripting to read in Rigi files.

Tilley and Huang report on their experiences with an industrial client in implementing a software visualization and documentation system in Visio [11]. Visio was selected after evaluating the visualization capabilities of several candidate tools. The authors were constrained in their technology choices by the client's policies. For example, "the company reasonably requested that professional support be available for whichever tools were selected. This requirement immediately ruled out almost all academic and research tools." Among the identified benefits of Visio was that the client already employed Visio in their development process and had a set of custom-developed stencils to represent their software artifacts.

Similarly to our approach, the authors used Visio to visualize software artifacts in a graph.

Systa et al. have extended Rigi with support to calculate and visualize various object-oriented metrics (among them RFC and CBO; see Section 3.3) [12]. Metrics data can be exported to Microsoft Excel spreadsheets. The spreadsheet is then used to visualize the metrics data with line diagrams similar to our approach. They also use an Excel macro to generate a correlation matrix for the metrics.

5. Conclusion

By building REVisio on top of Microsoft Visio, we not only saved development effort but also lowered the barrier to adoption that reverse engineering tools face. Users can leverage previously acquired cognitive support and take advantage of widespread training opportunities, flattening the learning curve. Integration with other office tools is enhanced, resulting in a uniform user interface and the ability to easily embed diagrams into other documents. Finally, Visio (and thus REVisio) supports scripting in the popular Visual Basic language. Ad hoc scripting is critical to reverse engineering activities, and VBA may prove easier to adopt than Rigi's Tcl.

Naturally, there are some tradeoffs to our approach. Visio's diagramming engine was originally meant for interactive use, so some aspects of it can be difficult to automate. For example, some masters automatically pop up dialog boxes or are limited to some small number of nested elements. These limitations can for the most part be circumvented with careful programming and shape customization. Performance, however, may prove to be a serious stumbling block. While VBA is usually fast enough for ad hoc scripting tasks, Visio itself was clearly not designed to scale gracefully to the thousands of shapes required to display models of even moderately large systems. Low performance is a factor in tool adoption and may negate any gains made by basing REVisio on a popular tool.

Further work is necessary to validate our claims of lowered adoption barrier, as well as to improve the usability and performance of REVisio. Nonetheless, we believe that REVisio explores an interesting and potentially worthwhile avenue of research, and demonstrates the promise of the adoption-centric reverse engineering approach.

6. References

[1] H. Müller, M.-A. Storey, K. Wong, "Adoption-Centric Software Engineering", http://www.acse.cs.uvic.ca.

[2] A. Walenstein, "Improving Adoptability by Preserving, Leveraging, and Adding Cognitive Support To Existing Tools and Environments", *3rd International Workshop on Adoption-Centric Software Engineering*, 2003.

[3] Microsoft Corporation, *Developing Microsoft Visio Solutions*, Microsoft Press, 2001.

[4] Microsoft Corporation, "Can Visio be customized?", http://www.microsoft.com/office/visio/faq.asp.

[5] D. A. Edson, *Professional Development with Visio 2000*, SAMS, 2000.

[6] H. Müller, K. Wong, S. R. Tilley, "Understanding Software Systems Using Reverse Engineering Technology", *The 62nd Congress of L'Association Canadienne Française pour l'Avancement des Sciences (ACFAS '94)*, 1994.

[7] N. Fenton and M. Neil, "Software Metrics: Roadmap", *The Future of Software Engineering track, 22nd International Conference on Software Engineering*, pages 359-370, 2002.

[8] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design", *6th ACM Conference OOPSLA '91*, pp. 197-211, 1991.

[9] S. P. Reiss, "Simplifying Data Integration: The Design of the Desert Software Development Environment", *18th International Conference on Software Engineering (ICSE '96)*, pp. 398-407, May 1996.

[10] C. Riva and Y. Yang, "Generation of Architectural Documentation Using XML", *9th Working Conference on Reverse Engineering (WCRE 2002)*, pp. 161-169, October 2002.

[11] S. Tilley and S. Huang, "On Selecting Software Visualization Tools for Program Understanding in an Industrial Context", *10th International Workshop on Program Comprehension (IWPC 2002)*, pp. 285-288, June 2002.

[12] T. Systa, P. Yu, and H. Muller, "Analyzing Java software by combining metrics and program visualization", *Fourth European Conference on Software Maintenance and Reengineering*, pages 199-208, 2000.