Building a Software Metrics Visualization Tool using the Visio COTS Product

by

Yu Chen

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

© Yu Chen, 2006
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part,
by photocopy or other means, without the permission of the author.

Building a Software Metrics Visualization Tool using the Visio COTS Product

by

Yu Chen

Supervisory Committee

Dr. Hausi A. Müller, Supervisor (Department of Computer Science)

Dr. Jens H. Weber, Department Member (Department of Computer Science)

Dr. Daniel M. Germán, Department Member (Department of Computer Science)

Dr. Shihong Huang, External Examiner (Florida Atlantic University)

Supervisory Committee

Dr. Hausi A. Müller, Supervisor (Department of Computer Science)

Dr. Jens H. Weber, Department Member (Department of Computer Science)

Dr. Daniel M. Germán, Department Member (Department of Computer Science)

Dr. Shihong Huang, External Examiner (Florida Atlantic University)

Abstract

Software metrics tools help people analyze, measure, and understand selected features and attributes of software systems. In recent years, more and more software metrics tools have incorporated sophisticated visualization capabilities to present software metrics results more effectively to end users. Despite advanced graphical user interfaces, adoption of custom stand-alone metrics tools is still a problem. Moreover, increasing functionality and complexity exacerbates this problem. To address this problem, we investigated how to build a software metrics research tool using Microsoft Visio—a Commercial-Of-The-Shelf product that is familiar to most end users. As the industry leader for business drawing, Visio offers a complete solution for engineers and technicians who need to create detailed schematics. Hence, Visio is an ideal candidate for visualizing metrics data. With built-in scripting capabilities, Visio is end-user programmable and therefore developers are able to extend its functionality and design custom solutions. Our assumption is that industrial

collaborators, who are already familiar with Visio, are more amenable to investigate and evaluate a software engineering research tool based on Visio than an idiosyncratic tool built from scratch. Moreover, Visio has a large user base and thus a Visio-based research tool has a better chance of being adopted in practice.

In this thesis, we report on our investigations of using Visio for building a software metrics research tool, called Vimex. In particular, we describe the design and implementation of Vimex as well as its integration with REVisio—a reverse engineering tool built on top of Visio. Finally, we report on our lessons learned in these investigations, discuss current limitations of this approach, and outline avenues for future research.

Table of Contents

List of Figures

Acknowledgments

I would like to thank my supervisor, Dr. Hausi A. Müller, for the opportunity to work in his research group at the University of Victoria on the Adoption-Centric Reverse Engineering (ACRE) project, and for his continued encouragement and support even while I worked at IBM Toronto and IBM Victoria. I also would like to thank many members of the Rigi group for their help and support, including Qin Zhu, Grace Gui, Holger Kienle, Piotr Kaminski, Scott Brousseau, Tao Liu, Jing Zhou, Ingrid Zhou, Feng Zou, and Tony Lin. I also would like to acknowledge the generous support I received from IBM Corporation and the Natural Sciences and Engineering Research Council (NSERC) through Dr. Müller's research grants and through teaching assistantships from the Department of Computer Science at the University of Victoria. Finally, I would like to thank my friends and family who encouraged and supported me over the years.

Dedication

To my parents

Chapter 1 Introduction

"When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of the meager and unsatisfactory kind."

—*Lord Kelvin* [Kelv1894]

1.1 Motivation

With the tremendous growth of the computer industry over the past few years, software products have become part of everyday life. The increasing importance of software places more demands on the software life cycle, management of software development projects, and on the overall quality of the software products. To design and manage software projects successfully and to evolve software effectively, software engineers need scientific ways to identify, create, revise and evaluate software processes and products. Software measurement is one way to characterize attributes of software products and processes and mapping them to numbers or symbols.

Software metrics are numerical data related to software processes and products, which support activities such as software project management and product assessment. The data can reveal characteristics of a software system and help engineers understand system properties to aid ongoing processes and decision-making. By collecting information from software processes and products and expressing the information as numerical values, engineers can compare and evaluate properties and qualities of different processes and products. Typically, software metrics are used to assess the complexity of software, the quality of artifacts, or the reliability of products [Lanz01]. With the support of software metrics, project managers and software engineers can reduce the cost and effort of software projects and make more accurate predictions and better informed decisions during the entire life cycle of the software products.

Often software professionals create their own metrics in order to accommodate their special needs, constraints and problems. However, developing well-designed metrics is non-trivial. Conversely, ill-defined metrics pose the threat of producing dubious results that can then misguide the software engineer. Thus, it is important for metrics researchers to find good strategies and techniques for evaluating software metrics and metrics (research) tools.

The following questions or concerns may arise when building metrics research tools to assist software engineers in gathering information and computing numerical values for software properties:

- How do we convince industrial collaborators to investigate and evaluate a metrics research tool?
- How do we convince potential customers to adopt and use such a tool?
- What is the target audience for a particular metrics tool?

- Given a target audience how do we characterize the strengths and weaknesses of a metrics research tool?

There are many potentially desirable characteristics for a successful metrics tool. In this thesis, we concentrate on the following characteristics of a research tool:

1. adoptability;
2. attractiveness (or appeal); and
3. appropriateness.

In the following, each characteristic is explained in more detail.

Adoptability

With adoptability we mean the act of favorable evaluation and acceptance of a research tool by industry. Many tool characteristics contribute to adoptability of a software engineering tool [Wong99], including:

- familiarity of potential end users with the tool;
- integration of the tool with other tools;
- ease of integration of the tool into actual development processes; and
- learnability (i.e., how difficult it is for a novice, an intermediate user, or an expert to become proficient using the tool).

Different tools come with different features and capabilities, such as the user interface, prescribed methods, and end-user programmability. These features affect how users learn and use the tool. If users feel that using the tool is difficult or that it takes a long time to get proficient, they will abandon it and seek out other tools. Therefore, adoptability should be a primary design goal. This is particularly important for a metrics tool that strives to help people to analyze, measure, and understand software products quickly.

Attractiveness

Attractiveness (or appeal) is another key characteristic that indicates how attractive a tool is to users with respect to its look-and-feel and functional capabilities (i.e., usability and usefulness of a tool). If a tool delivers desired information in appropriate levels and formats, it will be attractive to users. It is critical that the input and output formats are compatible with the development environment and process employed by the users. Thus, a metrics tool should be able to accept input in various formats, including source code as well as XML-based and textual intermediate forms (e.g., GXL and RSF). The output format also has to appeal to end-users. To present metrics data in an intuitive and attractive manner to end-users while also assisting them in achieving their goals is an important requirement.

Appropriateness

Appropriateness recognizes that different user groups have different needs and tries to satisfy their varying requirements. When designing a metrics tool, we need to understand that users have different levels of experience and therefore interact differently with the tool. For example, software developers are more interested in code-centric measurements to assess the quality of the source code, while project managers are more concerned with higher-level quality and process measurements to assess the status of a project as a whole. Thus, a successful metrics tool should be able to accommodate a variety of users with respect to usefulness and expertise.

1.2 Approach

Given the three characteristics outlined above (i.e., adoptability, attractiveness, and appropriateness)—AAA for short—we ask ourselves: how can we construct a metrics tool that satisfies AAA? Also, is there a general approach to building software tools which produces tools with these kinds of properties?

Developing a stand-alone tool is the normal approach for software engineering researchers. This approach comes with the following advantages: developers have full control over the tool's source code, the development environment is flexible, and developers can create their own standards or definitions to support special functions. Its disadvantages are a longer development period and thus more development costs and effort.

Another development approach is to use available components to assemble a metrics tool. There are many parsing, visualization, and metrics calculation components that could be readily reused in a component-oriented approach [Kien06]. If the quality of the components is high, the reliability of the tool will likely be higher than that of a "home-grown" stand-alone tool. However, integrating different visualization components to form a common look-and-feel is not trivial to accomplish. An even more radical approach is to leverage a single COTS (Commercial-of-the-Shelf) product to build a metrics research tool. The idea is to build a software engineering tool by extending a COTS product. If the underlying COTS product is already integrated into the development process and used by the targeted developers, then the tool's adoptability is significantly improved. This approach to tool building has been advocated by the ACRE (Adoption-Centric Reverse Engineering) and ACSE (Adoption-Centric Software Engineering) research projects conducted under the direction of Dr. Müller at the University of Victoria [MSW03, MWW03, BLMS04].

Today, COTS products, such as web authoring, diagramming or word processing tools are common in a software engineer's work environment—Lotus Notes, Microsoft Office, Microsoft Visio, Adobe Acrobat or Adobe GoLive come to mind. Such COTS products are available on many platforms, have large user bases, import and export various document formats, feature polished user interfaces, come

with version control and CSCW (Computer Supported Collaborative Work) capabilities, and are extensible and end-user programmable. By building software tools on COTS products, research tool developers get all this functionality and quality for free, and at the same time the development effort is greatly reduced.

One great advantage of building tools on a COTS product is the large user base that a COTS product enjoys. Many tools developed by academics fail to be evaluated and adopted in industry due to the unfamiliar user interface of these tools, missing customer support, and the lack of interoperability with other tools with respect to control and data integration [THP03]. In contrast, if a software engineering research tool is built using a COTS product, the researchers have a better chance to find a target audience in the large user base of the underlying COTS product that is willing to evaluate their tool.

Considering the alternatives, we chose the COTS-based approach to build our software metrics research tool. Following the idea proposed by the ACRE and ACSE projects, we built our tool on top of a popular COTS product to gain development, user base, and interoperability support from the underlying COTS product.

There are several COTS products to choose from when building a metrics research tool. One important consideration is how to present the resulting measurements. Metrics data are typically exported into plain text files. Data in a textual format is easy to generate and manipulate, and possesses comprehensive and detailed information. However, textual information is often not the most effective way to present metrics data to the user. Visual techniques can make complicated information easier to understand [STSH02], and can be more intuitive than plain text, thus helping people to understand complex data better [CFYS02]. Thus, ideally our underlying COTS product should be able to manipulate numbers effectively, come

with diagramming capabilities, and feature a graphical user interface. The spreadsheet metaphor is an ideal representation for manipulating metrics data in a declarative manner. Diagramming tools are ideal for visualizing metrics data and integrating metrics data with other software views (e.g., UML diagrams).

Finally, COTS products offer a variety of mechanisms to address the requirement of meeting different user needs, including declarative extensions, scripting interfaces, plug-ins, components and APIs. In our approach, we provide a framework with different logic and presentation layers to meet the needs of two types of users: high-level users (e.g., managers and system architects) and low-level users (e.g., programmers and testers).

1.3 Outline of the Thesis

This thesis consists of eight chapters. Chapter 2 provides background and related work on software engineering, software metrics, and software metrics tools. Chapter 3 introduces the "host" of our metrics tool—Microsoft Office Visio 2003—and provides a discussion on why we have chosen it. Chapter 4 describes the architecture of our metrics tool. Chapter 5 provides implementation details on our metrics tool, which is named Vimex. Chapter 6 discusses the integration of Vimex with the REVisio reverse engineering tool. Chapter 7 compares Vimex with other metrics tools. Finally, Chapter 8 summarizes the contributions of this thesis and proposes avenues for future research.

Chapter 2 Background and Related Work

This chapter introduces background knowledge for this thesis and summarizes related work that inspired our research. We first present some software metrics background and then describe the notion of Adoption-Centric Software Engineering (ACSE). Related work on metrics and visualization is reviewed in the final section of this chapter.

2.1 Software Metrics

"A major difference between a 'well developed' science such as physics and some of the less 'well-developed' sciences such as psychology or sociology is the degree to which things are measured."

— Roberts [Robe79]

Software metrics (also known as software measurement) are ways to measure the quality of software artifacts and software development processes. Software metrics provide mathematical descriptions of software and process quality by mapping empirical objects to numerical ones while preserving the inner relations and structures. The mathematical descriptions are useful, for instance, in estimating the

complexity of software, tracking project progress, understanding development stages and predicting potential defects.

With the help of software metrics, software engineers and project managers can get quality measures early in the development cycle and make better decisions concerning the project's progress and resources' allocation during the lifetime of software projects [NFMN00]. It is especially desirable that software metrics indicate where and what to inspect so that time and effort can be saved.

The following are some of the many different software metrics that can be employed at different stages of the software life cycle:

- **Requirement Analysis Metrics:** Requirement analysis, which is a major activity of requirement engineering, is concerned with determining the information needs and the functional and technical requirements for a potential system or product [PDPB04]. Many crucial questions arise during requirements analysis. Can the requirements be fully realized? Are there potential risks in completing the functionality specified by the requirements? Requirement analysis metrics assist project managers and quality assurance engineers in identifying the risks and refining the requirement specification during the early stages of the project.
- **Specification Metrics:** Based on the specification of the functionality of the software, specification metrics help project managers and developers predict more accurately what the software would require in cost or time. These measures can be very helpful in managing and scheduling the software project [Arif93].
- **Design Metrics:** By computing appropriate metrics during the design, managers and software developers are able to infer more characteristics

about the software that they are developing. They can predict and identify modules that are potentially error-prone, aid designers in evaluating different choices, and discover potential issues that may cause difficulties later on in coding and maintenance [DMRT01].

- **Code Metrics:** Code metrics compute software attributes and quality measures of source code artifacts. They reflect the properties of the source code directly and are used widely by researchers and software engineers to examine and improve the quality of the software product.
- **Test Metrics:** During the test phase, the performance, reliability, and security of a software product are tested. Applying test metrics regularly demonstrates the effectiveness of the software testing and change management process [Brad03]. With these metrics data, fault-prone modules can be identified more easily and efforts can be concentrated on improving the quality of these modules.
- **Maintenance Metrics:** Software maintenance is an activity that needs to be performed for all deployed systems [BVT03]. Maintenance metrics provide answers with respect to the reliability and maintainability of a software product. However, these attributes are difficult to measure because of the indistinct requirements from which they are derived. The advantage of having early indicators of future software problems outweighs this inconvenience [Schn01].

Software metrics can be divided into two broad categories: software product metrics and software process metrics. Software product metrics are the measurements that are taken on the actual software product itself, such as code metrics. Software

process metrics are the measurements related to the software development and maintenance process, such as effort, cost and specification metrics.

Product metrics include external product metrics and internal product metrics. External product metrics provides properties visible to the users of a product, such as functionality metrics, performance metrics and usability metrics. Internal product metrics cover the properties visible only to the development team, such as size metrics and complexity metrics [Meye98].

Product metrics can also be classified as object-oriented metrics and non object-oriented metrics. The traditional metrics are non object-oriented metrics that focus more on the functionality of a software system than on its architectural characteristics. With the emergence of object-oriented programming, object-oriented metrics are widely used in object-oriented analysis and design. Typical object-oriented metrics include metrics about the complexity of the system hierarchy and objects coupling, such as software package metrics (i.e., cohesion and coupling metrics).

Overall, the use of software metrics in software engineering can help software engineers to understand software, to manage software projects, and to guide process improvement [BMP95].

2.2 Adoption-Centric Software Engineering

Research tools in software engineering often fail to be evaluated and adopted in industry [MSW03]. A likely reason is that people are reluctant to evaluate or use tools with which they are unfamiliar. Adoption barriers encountered by stand-alone research tools include: unfamiliarity, rough and unpolished user interfaces, poor

interoperability with existing tools, poor adaptability for complex work requirements, and lack of commonly expected document management functionality.

The key research question of the ACSE project at the University of Victoria is: how to build software engineering research tools that are easier for people to evaluate, accept, and use? The goal of this research is to find ways of building software engineering research tools that leverage good cognitive support and good interoperability with other tools and the target work environment. The premise is that improved cognitive support and interoperability will lead to more favorable industrial research tool evaluation and eventually better adoption.

For software products, the main factors of cognitive support consist of familiar user interfaces as well as similar functionalities and processing procedures. Cognitive support can be improved effectively by providing familiar environments or platforms to users, such as an environment in which a user works daily or intimately. The ACSE project assumes that when we develop software engineering tools on top of familiar environments or platforms, which feature popular user interfaces and functionalities, the cognitive support of the resulting research tools is greatly improved.

Interoperability is the ability to work with other entities (e.g., individuals, objects and tools). Here, interoperability refers to the ability of a system or tool to work with other systems or tools without special effort on the part of the user. For software products, it can be categorized into two major aspects: functionality and data.

Interoperability of functionality means a software product can use the functionalities (or part of the functionalities) provided by other software or vice versa. Interoperability of data refers to the ability of software products to exchange data, such as the ability of importing or exporting several data formats. Today, the most popular data carrier format is XML (Extensible Markup Language). XML is a W3C

standard based on tags that allows information and services to be encoded with meaningful structure.

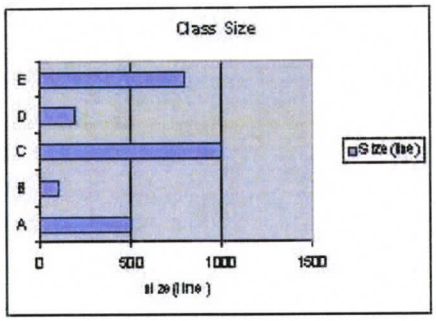
Because "home-grown" stand-alone tools typically come with their own functionalities and data sets, it takes significant time, expense, and effort to provide effective cognitive support and interoperability to integrate the tools with other tools and the user's work environments. In contrast, COTS products, such as Microsoft Office or Lotus Notes, are widely used and have standardized modules and middleware technologies that allow them to interoperate and cooperate easily and effectively. For example, an Excel spreadsheet can be exported to Word and PowerPoint by copying and pasting.

We believe that, by building software engineering research tools based on a COTS product, the resulting tool will automatically come with popular middleware technology and document management capabilities as well as a large user base (i.e., the user base of the underlying COTS product), thereby, enhancing its cognitive support and interoperability.

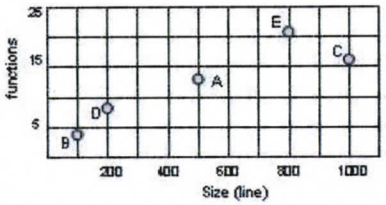
2.3 Data Visualization

Data visualization is an important technology in many disciplines, including software engineering. Transforming data into graphical forms for easier interpretation and perception by humans is the common goal of all visualization communities. When large volumes of data are collected, processed, and represented graphically, people can often quickly and effectively discover characteristics, patterns, and trends that are not readily apparent in other formats such as plain text [TKVM00].

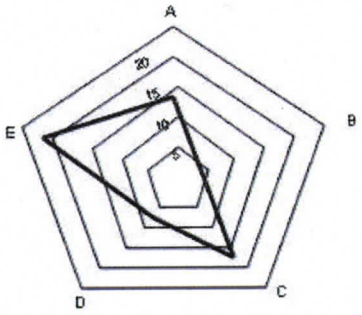
Many graphical techniques, such as charts and diagrams, are used for data visualization. Figure 2.1 depicts a few samples of data visualization techniques.



(a) Bar chart



(b) Grid



(c) Radar chart

Figure 2.1: Graphical data visualization techniques

Of course data visualization can also be employed for presenting metrics data. With data visualization, the traditional, tedious data analysis and statistics processes are amplified and improved when using appropriate graphical renderings and an effective approach to manipulate the underlying data (e.g., functions attached to spreadsheets or tools' built-in macros). With data visualization, the metrics data are easier to understand and explore, thus increasing the understandability of the structure and attributes of software products.

2.4 Related Tool Support

In this section, we discuss existing software engineering tools that are related to our work. We first describe existing examples of tools that have been built following the ACSE methodology, which inspired us for this research, and then present several metrics visualization tools.

2.4.1 ACSE Project

The ACSE project, conducted at the University of Victoria under the direction of Dr. Müller, concentrates on the problems of evaluation and adoption of software engineering research tools in industry. Several theses and papers have been published under the auspices of this project and this section reviews two selected subprojects: Live Documents and ACRENotes.

2.4.1.1 Live Documents

An ACSE Live Document is a document with a state that can adapt automatically and intelligently to its context. It is implemented on top of öffice platforms. A Live Document is data-driven and synchronizes code and documentation automatically, keeping documentation up-to-date. As a case study, live documents have been used to enhance the documentation capabilities of the Rigi reverse engineering tool [WKM02].

2.4.1.2 ACRENotes

ACRENotes is an adoption-centric reverse engineering tool that was developed on top of Lotus Notes [Ma04, MKKW03]. By managing reverse engineering data (e.g., in GXL format) as documents, ACRENotes leverages the Lotus Notes' database and provides graphical views to present the data. ACRENotes benefits greatly from Lotus Notes' groupware features, allowing team members to cooperate effectively and efficiently. With the help of LotusNotes' scripting languages, ACRENotes users can create their own functions and automations for complex tasks.

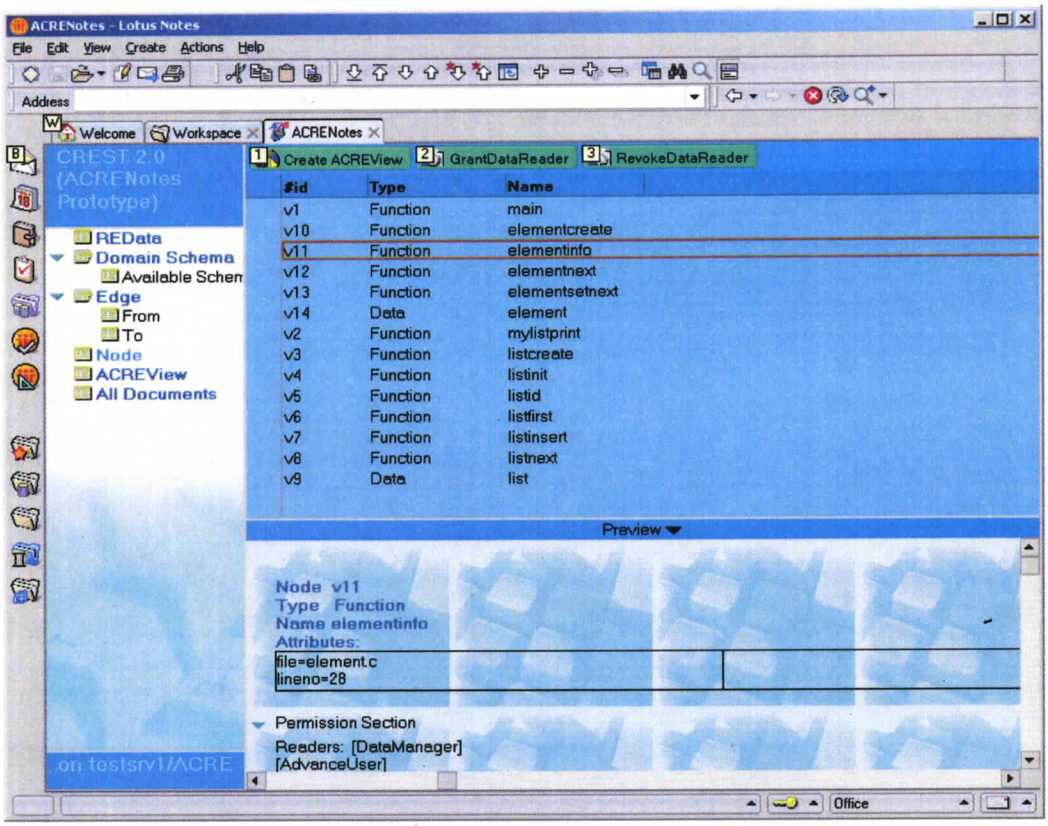


Figure 2.2: Main User Interface of ACRENotes [Ma04]

Figure 2.2 depicts a screenshot of ACRENotes. The top right window shows a document that contains a textual view of a software system graph. More detail about the selected graph entity (here the "v11" node) is provided in the bottom right window.

2.4.2 Metrics Visualization

This section shows approaches to metrics visualization, and discusses several tools that have metrics capabilities in the domains of reverse engineering, component-based software engineering, and the semantic web.

2.4.2.1 Imagix 4D

Imagix 4D is a well-established commercial reverse engineering tool which helps software developers to understand complex legacy software written in C and

C++ [Imag03]. Imagix 4D calculates quality metrics about the source code and displays the results using its metrics tool.

Func # : McCabe Cyclomatic Con (10 / 30)	Lines in Function	Stmts in Function	Vars in Function
Tk_WmCmd	840	805	0
CanvasWidgetCmd	916	843	0
Tk_WinfoCmd	348	322	0
MenuWidgetCmd	412	362	0
TkTextTagCmd	393	343	0
ExpandPercents	377	350	0
TextWidgetCmd	257	251	0
ConfigureSlaves	305	238	0
ListboxWidgetCmd	247	236	0
Tk_PackCmd	296	280	1
TkCanvPostscriptCmd	429	339	0

Figure 2.3: Metrics view from Imagix 4D [Imag03]

Imagix 4D does not visualize the metrics results, but uses a color scheme to represent the values (cf. Figure 2.3). For example, symbols in the graph model can be colored to indicate their metrics values. Users can also view the metrics information within the context of the software's structure.

2.4.2.2 CodeCrawler

CodeCrawler is a language independent reverse engineering tool that combines metrics and software visualization [Lanz03b]. Figure 2.4 depicts the System Complexity View in CodeCrawler.

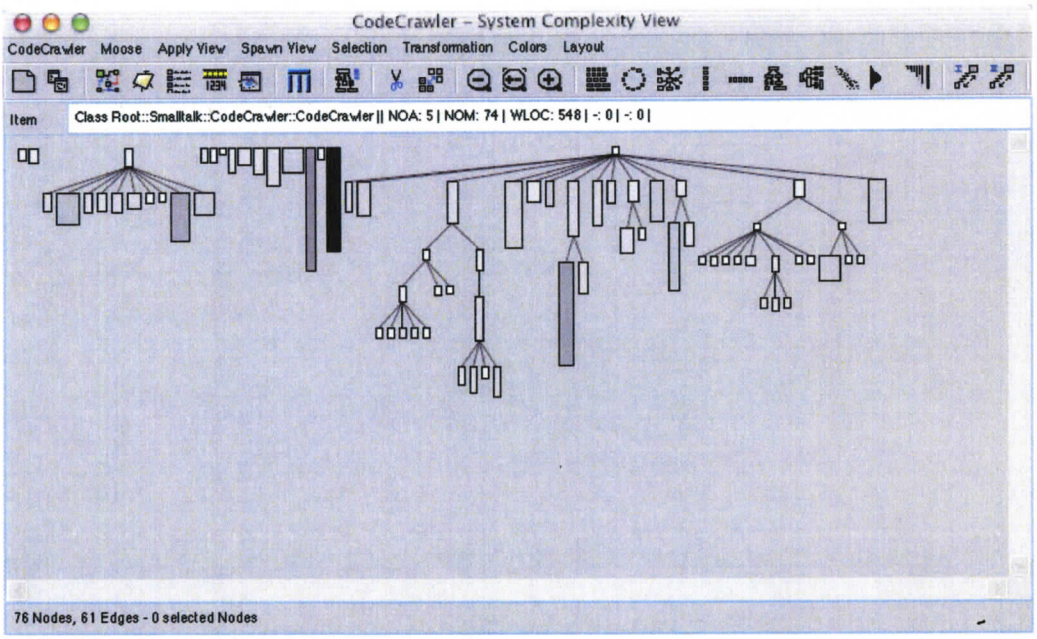


Figure 2.4: CodeCrawler—System Complexity View

In the graphical view, metrics data is visualized with different properties of arcs and nodes. For instance, the width of the nodes represents the number of class attributes, the height of the nodes represents the number of class methods, and the color of the nodes represents the number of lines of code of a class [Lanz03a].

2.4.2.3 Software Metrics and Visualization for Component-Based Software Engineering

Software Metrics and Visualization for Component-Based Software Engineering is a project at the University of Newcastle, Australia. It investigates size measures and prediction models for component-based software development, empirical evaluation of software measures, and an integrated visual environment for analyzing and tracking measures [SQHE01].

Large software systems may have hundreds of components. To estimate the effort for software project management, software engineers must know the size of the components and subsystems. Thousands of numbers in numeric form are difficult for

humans to comprehend and use. In this project different visualization techniques are being explored to simplify the comprehension of metrics data. For example, Figure 2.5 shows the use of a cityscape metaphor for depicting the software size of components in a system.

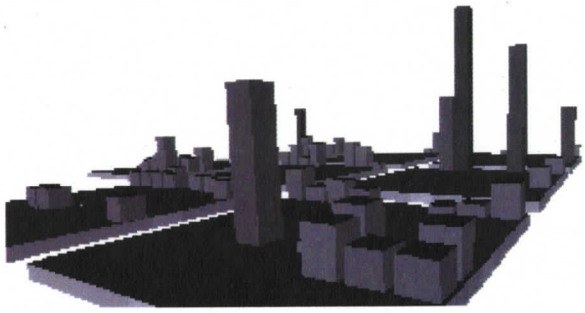


Figure 2.5: Software Metrics and Visualization project—Size visualization

2.4.3 Microsoft Visio add-ons

SemTalk is a modeling tool that helps users to contribute to the semantic web through a Visio-based graphical editor [Sem01]. It eases the creation and management of meta-models. The SemTalk Object Engine is the core of SemTalk and is used to define the meta-model for Visio Shapes and the connections among types of shapes. Models can be exported or imported in RDF (Resource Description Framework) format.

2.5 Summary

This chapter reviewed useful selected background for this thesis, including the need for software metrics, the notion of Adoption-Centric Software Engineering, and the utility of data visualization techniques. Software metrics are helpful for software engineers throughout the software's life cycle. ACSE is a promising approach for potentially improving the adoptability of software engineering tools. Data visualization enhances people's analysis capabilities of metrics data. Finally, related

projects about software metrics and ACSE were presented. The following chapter introduces our adoption-centric approach of designing a metrics tool on top of Visio.

Chapter 3 Approach

The ACSE approach for building software engineering research tools advocates using a COTS product as a basis for development. In this chapter, we argue that Microsoft Visio is an ideal platform for building a metrics tool. According to the ACSE approach, a commercial host environment is needed to ease evaluation and eventual adoption of research tools. In addition, a host tool with a sophisticated graphical user interface is preferred to enable the effective visualization of metrics data. As arguably the most powerful diagramming tool on the market, Microsoft Visio, with its robust environment and many customization options for end users to accomplish complex and specialized tasks, seems ideal for our purposes.

3.1 Microsoft Visio

Microsoft Visio is a key component of the Microsoft Office suite. As a drawing and diagramming tool, it helps users in designing and creating a variety of diagrams, such as flowcharts, block diagrams, building plans, or maps.

Visio comes with facilities, such as predefined templates, stencils and master shapes, to ease and automate the design and the drawing of diagrams. Templates are

saved configurations of different diagram types to help users get started. Stencils are collections of master shapes that users may utilize in their designs. Master shapes are listed in a separate pane on the side of the canvas so that users can drag and drop them into the canvas to generate shapes automatically.

Visio also provides many customization features and options for users. A basic customization approach for users is to save their own designs as templates, stencils, and masters for future use and later deployment. One of the most powerful customization options of Visio is the notion of a "ShapeSheet", a spreadsheet containing all the properties of a shape, including its geometry information, events and behaviors. By specifying values or defining formulas in the ShapeSheet, the designer can fully control a shape. For example, formulas allow the user to establish multiple relations among shapes. If the property of one shape changes, the other related shapes gets adjusted accordingly. Visio's functionality can also be accessed programmatically via its API. Through Visio's object model, which encompasses shape, canvas, ShapeSheet, toolbar and menu, client applications can access any objects provided by Visio and control them programmatically.

3.2 Why Visio?

As described in the previous section, Microsoft Visio is a powerful tool for designing and creating diagrams. Even though, without further evaluation, it is not guaranteed that Visio indeed is a suitable host component for our metrics tool. In order to make a more informed decision, we assess Visio by looking at the desirable characteristics that a host tool should possess (cf. Chapter 1):

- **Large user base:** The users of the host tool are the potential users of our metrics tool. Thus, the host tool's users, who have gained experience with

its functionality and user interface, are able to leverage their acquired cognitive support directly for our metrics tool.

- **Familiar user interface:** Tools have to be both useful and usable. The latter is mostly influenced by the tool's user interface. Moreover, the learning curve for the new tool is significantly improved for the user, if the tool's user interface is grounded in familiar paradigms. As a result, the new tool will more likely be evaluated and adopted than a stand-alone custom tool.
- **Customization:** Different users have different requirements for using a research tool, and in particular different needs for how to integrate the tool into their existing working environment [Till95].
- **End-user programming:** This is an advanced customization feature, which is particularly useful to automate operations. Automation can save users' time and effort on repetitive and tedious tasks. Many tools use scripting languages to provide end users with the flexibility to modify the existing functionality, or to create new functions. Furthermore, users may customize the user interface as they see fit.
- **Interoperability:** Software engineering tools rarely operate on their own. Thus, it is necessary for them to interact with other tools on data or control levels.
- **Graphical viewer (and editor):** To present results visually to the user, it is desirable that the host tool is already equipped with graphing and diagramming functionality.

Analyzing these criteria for Microsoft Visio 2003, we found that Visio satisfies all of them:

- **Large user base:** Visio is used by millions of users worldwide.
- **Familiar user interface:** Visio's user interface is similar (but not the same; for example, the drag-and-drop of the stencils onto the canvas is rather idiosyncratic), to popular Microsoft Office products, which constitutes an important standard in document manipulation. For example, a Word user can do basic operations in Visio (e.g., loading of files and selection of graphical objects) without any difficulty.
- **Customization:** Visio features many customization mechanisms to cater to the different requirements and needs of its users, such as templates, stencils, masters, and the ShapeSheet.
- **End-user programming:** Visio has a built-in scripting language called VBA (Visual Basic for Applications) for creating macros to automate tasks.
- **Interoperability:** Visio provides mechanisms for control, data and presentation integration [Micro03]. Visio's data can be imported and exported in several formats. Its graphs can easily be imported into Word or PowerPoint documents by copying and pasting. Through the Office Object Model, users can interoperate with Visio by initializing a Visio file from other Office products or launch other Office products from Visio.
- **Graphical viewer (and editor):** Visio is the leading tool in the diagramming tool market, supporting many different kinds of diagram types, such as UML and building plan.

There are several other candidate COTS products with similar characteristics, including Microsoft Word, Microsoft Excel, or Lotus Notes. Comparing these tools against Visio, we found that Visio is more suitable for building a visual metrics tool. In the following we briefly compare Visio against Word/Excel and Lotus Notes.

Microsoft Visio vs. Microsoft Word and Excel

As members of the Microsoft Office suite, Visio, Word and Excel have many features in common, such as middleware technology, an end-user programming environment, and the look-and-feel of the user interface. Compared to Word and Excel, Visio has a clear advantage with respect to diagram viewing and editing. In addition, Visio's customization options provide more diagramming choices and functionality to visualize different aspects of metrics.

Visio vs. Lotus Notes

Lotus Notes is a groupware product with strong end-user programming capabilities. However, it lacks support with respect to diagram editing and manipulation as well as interoperability with other tool environments. The memory footprint of Lotus Notes is also considerable. In contrast, Visio has powerful diagramming capabilities. It also provides interoperability with other Office tools (e.g., users can migrate data between Visio and Excel).

3.3 Development Environment

Microsoft Office provides VBA (Visual Basic for Applications) as the default development language. VBA is a cross-product programming language and environment. It is designed specifically to provide development capabilities inside an off-the-shelf application. VBA is a subset of VB (Visual Basic) and inherits the Office object library and the application instance of VB. VBA is a "light-weight" VB that resides inside a COTS product. In contrast to VB, one does not need an application instance to run a VBA application. Every Office application has both the VBA engine and an IDE (Integrated Development Environment). Users who are already familiar

with VB's development environment need little training to use the IDE in the Office products to customize, extend, and integrate applications with VBA.

As a member of Microsoft Office, Visio has the following features. It is a robust, practical and viable development platform for creating solutions that integrate with business systems, databases, networks, Microsoft Office, and other VBA-compliant applications. By extending the third-party add-ons of Visio, users can create their own solutions based on Visio and the Microsoft Office development platform. VBA codes are wrapped with Visio documents so that they are easy to save and exchange, and the user does not need to manage separate VBA code files.

3.4 Summary

In this chapter, we introduced the host COTS product for our tool, Microsoft Visio, the criteria for choosing the host tool, and the means by which Visio meets our requirements. After that, we discussed the development environment provided by Visio including the built-in VBA language. The next two chapters present the design and implementation of our metrics tool, Vimex, which we built on top of Visio.

Chapter 4 The Metrics Tool Vimex

This chapter discusses the architectural design and components of our metrics tool, Vimex. To separate concerns, we structured Vimex into three layers (*Metrics Core Engine layer, Application layer,* and *Presentation layer*). Given our goal of meeting end users' needs by allowing extensive customization, we introduced layers to separate metrics calculations, customization, and visualization. This chapter also discusses two granularity levels (overview/detailed level) in the application layer for two different user types (high/low-level), and the tool's metrics provided in the detailed level.

4.1 Architecture

To separate the core metrics functions from customization and visualization, we designed three layers for our metrics tool—*Metrics Core Engine layer, Application layer, and Presentation layer*—as depicted in Figure 4.1.

- The **Metrics Core Engine layer** defines the metrics data model and provides basic operations for metrics calculations.

- The **Application layer** connects the Core Metrics layer with the Presentation layer. It is structured further into two levels to meet different user requirements.
- The **Presentation layer** encapsulates visualization technique for visualizing metrics data. Based on the information produced by the Application layer, it generates graphical depictions of the metrics results.

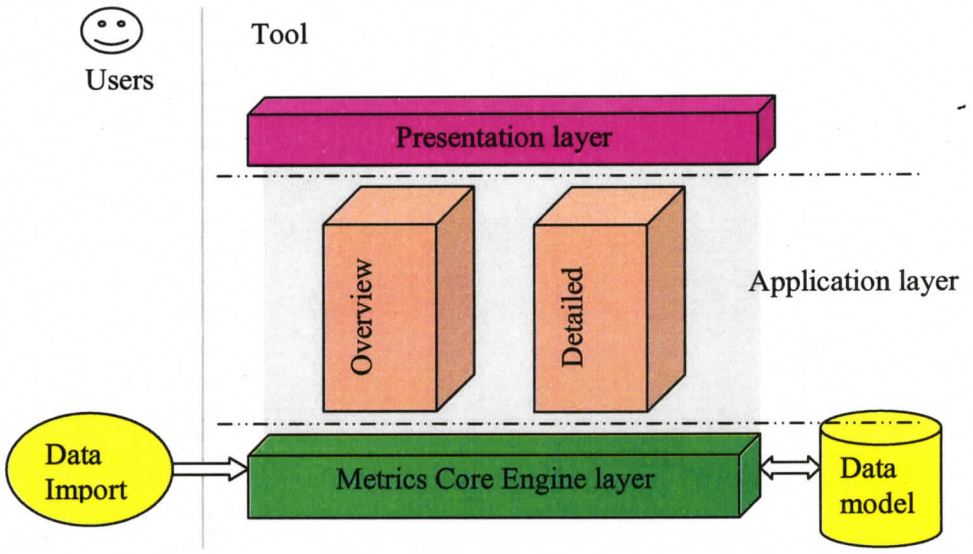


Figure 4.1: Architectural design of Vimex

4.2 Metrics Core Engine Layer

The Metrics Core Engine, as depicted in Figure 4.1, is the core of our Vimex metrics tool. It includes the metrics calculation engine and the metrics data model. The engine is responsible for importing data and performing metrics calculations. The metrics data model is the major data structure in Vimex. The Metrics Core Engine provides the most basic metrics functions. This layer by itself constitutes a basic

metrics tool without visualizations and customizations. It can be also seen as a reusable library that other researchers can leverage when building their own tools.

4.3 Application Layer

The Application layer connects the Metrics Core Engine with the Presentation layer. It supports of two levels of granularity in order to cater to the requirements of different end-user communities.

Software metrics are important resources that help software developers and project managers to understand and assess features of software products and projects. Due to the different focuses of different users, there are different user expectations and needs. As the designers of a metrics tool, we should always consider the possibility of different user requirements and provide a range of suitable solutions. The questions are who the potential users are, and how they rely on a metrics tool to accomplish their tasks. Generally, the end users of a metrics tool would be: software project managers, system architects, software engineers, programmers, and testers. Their objectives can be summarized into two categories: (1) collecting information about a software project for planning or managing purposes; and (2) capturing certain objects' information for development or maintenance purposes. These two different objectives address different types of requirements for a metrics tool: (1) the first objective requires more system-wide properties with little specific information; (2) the second objective requires more details on metrics at the object/class level. We can also presume that the users with the first objective may not have profound metrics knowledge, or that it is not necessary for them to possess profound metrics knowledge. Therefore, metrics details should be left out or abstracted for these users.

To provide different metrics granularity required by users, we introduce two levels within the Vimex Application layer: *overview level* for high-level users, and *detailed level* for low-level users. Each level is discussed in the following sections.

4.3.1 Detailed Level

The detailed level provides metrics calculation based on individual metrics or individual objects. It is designed for users that need specific metrics data to assist in specific development or maintenance tasks. In the detailed level, two main types of views are provided:

(1) Views that are based on **individual metrics**. The potential users of this type of views could be software engineers and developers who need to know how the subject system behaves with respect to a specific characteristic. For example, *Coupling between Objects* (CBO) is a metric that represents the count of the classes to which a certain class is coupled [SCCK94]. With CBO metrics calculated from all classes in a software system, it would be easier to identify the density of the inter-relations among classes, and to locate the classes that are highly related (i.e., higher CBO values) to other classes as well as classes that are more independent of other classes (i.e., lower CBO values). Software engineers and developers can make better decisions based on the CBO values—they should pay extra attention to those classes with high CBO for maintenance or modification purposes.

(2) Views that are based on **individual objects**. The potential users of these views need specific information about a particular object in a software system. They are most likely the software developers or programmers who want to modify a particular object. The metrics data of a single object can be depicted as a Radar chart or a Kiviat diagram, which is a well-known technique for depicting relations among multiple data sources. One example of using the Kiviat graph is depicting the usage of

various resources in computer systems [KKPK73]. Such visualization techniques are easier to compare and evaluate over different time periods than plain text.

The following metrics are provided in the detailed level:

- **Lines of Code (LOC)**—the number of lines of a class.
- **Depth of Inheritance Tree (DIT)**—the length from the node to the root of the tree.
- **Number of Children (NOC)**—the number of classes derived from a specified class.
- **Methods per Class**—the number of methods of a class.
- **Coupling Between Objects (CBO)**—the count of the (non-inheritance-related) classes to which a specific class is coupled. Two classes are coupled when methods declared in one class use methods or variables of the other class.
- **Response For Class (RFC)**—the number of methods that can be invoked in response to an object of a class, including methods implemented within the class and the methods accessible due to inheritance.
- **Weighted Methods per Class**—the number of methods implemented within a class.
- **Lack of Cohesion of Methods (LCOM)**—the average percentages of the methods in a specific class using data fields in that class, the percentages are then subtracted from 100%.
- **Method Inheritance Factor**—the ratio of the inherited methods to the total number of methods.
- **Attribute Inheritance Factor**—the ratio of the inherited attributes to the total number of attributes.

All of the above metrics associate a metrics value with an individual class (or object). Different metrics reflect different characteristics of a class. The combination of certain metrics values along with suitable visualization techniques can be an effective means to better understand and assess an individual class that is part of a software system.

4.3.2 Overview Level

Many users (especially high-level users) want a brief understanding about the current status of the project to glean a quick overview, perform a cursory inspection, make personnel assignments, etc. These users do not care for detailed metrics information, such as the meaning of every individual metrics value and how they are calculated. If the metrics tool provides too many details, the user would have to spend too much time wading through them. What is needed is a simple and concise presentation that shows the current key system attributes relevant for this particular type of user.

Currently, Vimex provides estimations of the following software attributes based on metrics calculations [Morr89, SCCK94]:

Complexity—complexity is estimated by several metrics, such as Lines of Code, Methods per Class and Depth of the Inheritance Tree.

Reliability—reliability can be associated with the cohesion of a module. High cohesion usually means high reliability.

Reusability—reusability can be reflected as Coupling between Objects and Number of Children. Excessive coupling between classes will prevent reuse of them. Generally, the larger the number of subclasses, the greater the reuse of the base class' code.

Extensibility—the number of Methods per Class indicates the class' extensibility. If a class has a larger number of methods (especially abstract methods that must be overridden), it is potentially more difficult and error-prone to extend it.

Maintainability—maintainability can be estimated by measuring the Coupling between Objects and the Cohesion within Classes. The more connections between classes exist, the higher the chance that changes will propagate. As a result maintenance can be expected to be more difficult. Cohesion indicates a class' independence. High cohesion means that a class is relatively independent of other classes, so it can be expected to be easier to maintain.

Testability—testability involves three factors: Coupling, Number of Classes and Response for a Class. The higher the class coupling, the more rigorous the testing should to be. A large number of classes and high coupling among classes means that more integration tests are necessary.

To obtain values for the above software attributes different metrics values have to be combined. However, different metrics have different value ranges, which can make it difficult to compare them to each other, and to summarize them effectively. To solve this problem, all metrics values in Vimex are converted into percentage numbers. Therefore, choosing a right base (divider) is very important for the conversion. The following rules are applied when combining metrics:

- For metrics values that are counts of objects (e.g., CBO), their percentage values are calculated by counts of objects divided by the total number of objects for that type (e.g., if the type is class, it means total number of classes in the system). For example, the system's CBO percentage value would be

$$systemCBO\% = \frac{Average(CBO)}{totalNumberOfObjects} * 100\%$$

where *Average(CBO)* is the average value of all classes' CBO values, and *totalNumberOfObjects* is the count of all classes in current system. For example, one class' CBO value of 15 means that it relates to other 15 classes. If the total number of classes is 50, then CBO%=15/50*100%=30%.

The percentage values are used in estimating the software attributes, however they are not metrics per se.

- For metrics (i.e., Methods per Class) values that are not calculated based on a total count of a type of objects, the maximum value of that metrics is used. The formula for Methods per Class is:

$$MethodsperClass\% = \frac{Average(MethodsperClass)}{Max(MethodsperClass)} * 100\%$$

For each software attribute, one or more metrics can be calculated, and then an average percentage number is calculated based on the metrics numbers. The best practice for computing the average percentage number is picking the same divider for the same metrics (or if possible, all involved metrics), especially when the attributes are used for comparisons.

These software attributes can provide an effective project overview for high-level users such as project or budget managers, who want to understand quickly the current status or trends of the project. One single attribute value may be difficult to interpret in isolation. Often it is more meaningful when the same attribute is tracked and compared over different releases, milestones, or development stages. This approach can reveal certain evolution features of a software project so that past activities can be summarized and future activities planned.

4.4 Presentation Layer

The Presentation layer handles the visualization of the metrics data. Depending on the user-level information provided by the Application layer, the Presentation layer generates appropriate views. The overview level is connected to system property views, and the detailed level uses various metrics charts. The actual presentation format is inferred at the Presentation layer depending on the Application layer's user level. The presentation layer is also responsible to map the metrics data to appropriate graphical objects and properties of these objects. This transformation is dependent on Visio's object model and is discussed in more detail in the next chapter, which covers implementation details.

4.5 Summary

In this chapter we presented the design of the Vimex architecture, which consists of three layers: Metrics Core Engine layer, Application layer, and Presentation layer. To meet different client group requirements for software metrics, we introduced two user levels for the Application layer: overview level and detailed level. The next chapter presents implementation details of the Vimex tool.

Chapter 5 Implementation of Vimex

This chapter presents selected implementation details of our Vimex metrics tool including data import in various formats, visualization techniques, and customization mechanisms.

5.1 Data Import

In the design of Vimex we concentrated on metrics calculation and presentation. The architectural design introduced in Chapter 4 shows that Vimex does not parse source code but rather imports data streams in various formats from external sources.

5.1.1 Data Formats

In this thesis, the term *source data* refers to well-formatted data that can be imported by the Vimex Metrics Core Engine layer for metrics calculations. These well-formatted data streams, whose formats are introduced in the following subsections, are normally generated by stand-alone parsers or reverse engineering tools.

5.1.1.1 RSF

The first type of source data formats is RSF, which stands for Rigi Standard Format. It is the primary data file format for the Rigi reverse engineering environment

[Wong98]. Many other tools provide support for RSF (i.e., importing and exporting of RSF streams). An RSF stream contains software artifact information extracted from source code by reverse engineering tools. Typical information recorded by RSF includes variables, functions, classes, subsystems and their relations. There are two variations of RSF: *unstructured* and *structured* [Wong98]. Unstructured RSF (also known as three-tuple RSF) consists of a sequence of triples. An RSF triple can represent an object, the relationship between two objects, or a value binding to an object's attribute. Structured RSF is different from unstructured RSF. We choose structured RSF as one of our source data formats.

The following sample is a fragment of a structured RSF file:

```
type    163!listnext    Function
call    163!listnext    295!elementnext
file    163!listnext    list.c
```

The above fragment describes a "Function" object called "listnext" (id = 163), a call from the "listnext" object to the Function "elementnext" (id = 295), and the location of "listnext" (i.e., file "list.c").

5.1.1.2 GXL

Although RSF is a suitable data format for Vimex, is quite popular in the reverse engineering and program understanding communities, and has withstood the test of time, it is not a W3C recommended standard data format. Moreover, even though an RSF stream consists of a schema and data portion, it is not an XML (eXtensible Markup Language) format. RSF was conceived before XML was defined by W3C. This limitation reduces the interoperability of our metrics tool and restricts it from accessing additional data resources. To ease this limitation, Vimex supports two XML data source formats: GXL and SVG.

XML is a popular data carrier that was created to define, validate and share documents on the Web. GXL (Graph eXchange Language) is an XML-based exchange format for graph data (i.e., data consisting of nodes and edges, or entities and relationships) and consists of XML elements to describe nodes, edges, and attributes [HWS00]. By supporting GXL in Vimex, the interoperability between Vimex and other software engineering tools is significantly improved.

The following data stream represents a sample GXL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gxl SYSTEM "gxl.dtd">
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
<graph id="SimpleGraph">
        <node id="v1">
                <type xlink:href="schema.gxl#Function" />
                <attr name="name">
                        <string>main</string>
                </attr>
                <attr name="file">
                        <string>listtest.c</string>
                </attr>
                <attr name="lineno">
                        <int>18</int>
                </attr>
        </node>
        <node id="v2">
                <type xlink:href="schema.gxl#Function" />
                <attr name="name">
                        <string>mylistprint</string>
                </attr>
                <attr name="file">
                        <string>listtest.c</string>
                </attr>
                <attr name="lineno">
                        <int>4</int>
                 </attr>
        </node>
        <edge id="e12" from="v1" to="v2">
                <type xlink:href="schema.gxl#call" />
        </edge>
 </graph>
</gxl>
```

Figure 5.1: Sample GXL file

From this sample GXL code, we can see that the main tags of GXL elements are "<node>" and "<edge>" with attributes attached to them. Thus, the information in GXL and RSF files is similar. There are translators to convert between RSF and GXL streams [Kien01].

5.1.1.3 SVG

SVG (Scalable Vector Graphics) is an XML-based W3C standard to describe two-dimensional graphics [Jack02]. Compared to GXL, which only encodes the relational structure of a graph, SVG supports graphical objects (e.g., curves, text, and images), and can thus be used to store the layout of a graph. In addition, SVG is interactive and dynamic, and supports scripting. SVG documents can be viewed with many COTS products including Web browsers, Visio, Word, and PowerPoint.

Thus, the availability and interoperability of SVG significantly enhances Vimex's usability and interoperability. By importing and exporting SVG files, Vimex is able to interact with other ACRE tools, including the SVG graph visualization engine [GKM05].

5.1.2 Importing Data

Source data can be imported into Vimex in two different ways: (a) using the toolbar option "File-Open" to read source data; or (b) by specifying the source data during the metrics calculation process. Vimex supports only one data source at a time.

The data model of Vimex follows the data structure of a relational database. There is an ID number associated with each object. Data from the source data file are stored in two dynamic arrays. Once created, no write operation is allowed on these two arrays (unless a new source data file is imported).

Figure 5.2 shows the code snippet of handling RSF, GXL and SVG data files:

```
Dim fd As CommonDialog
Set fd = New CommonDialog
With fd
  .CancelError = False
  .Filter = "RSF File (*.rsf)|*.rsf|any file (*.*)|*.*"
  .FilterIndex = 1
  .DialogTitle = "select file to open"
  .ShowOpen

  If .filename <> vbNullString Then
    Select Case UCase(Right$(.filename, 3))
      Case "GXL"
        ImportGXL (.filename)
        InitMetrics
      Case "RSF"
        ImportRSF (.filename)
        InitMetrics
      Case "SVG"
        ImportSVG (.filename)
        InitMetrics
      Case Else
        MsgBox "the file format is not supported"
    End Select
  End If
End With
```

Figure 5.2: Code snippet for handling source data files

Every metrics calculation may introduce new object attributes or relations among objects. Thus, additional arrays will be created to store the newly calculated metrics data as well as the corresponding object IDs. To output metrics results, it would be necessary to trace objects' data from the object array and the metrics data from a metrics data array. Therefore, an object's ID is an important key to link data from different arrays together.

5.2 Visualizing Metrics Results

In general, Vimex processes one set of data and generates another set of data. The way the output data is represented is important for a metrics tool. With many available visualization techniques, choosing an appropriate one is difficult. We prefer

simple but effective ways to present metrics data so that users are not unnecessarily distracted or overwhelmed.

Accordingly, four types of visualizations were chosen for Vimex:

- Multiple metrics data for single objects;
- Single metrics data for multiple objects;
- Single metrics data for the same object over different time periods; and
- Statistical data.

After an assessment of Visio's diagrams and shapes, we selected four types of charts to present the metrics results:

- Bar charts are used to represent single metrics data of multiple objects;
- Radar charts (and their variations) are used to display multiple metrics data on a single object;
- Line charts are suitable for data over different time periods; and
- Pie charts are chosen for presenting the statistical data.

An example of a bar chart is given in Figure 5.3: Chart for Response for Class, which displays "Response for Class (RFC)". In this chart, the X-axis lists classes that are involved in the calculation; the Y-axis indicates the response volume for each class. Classess with higher response volume can be easily identified as they have taller bars.

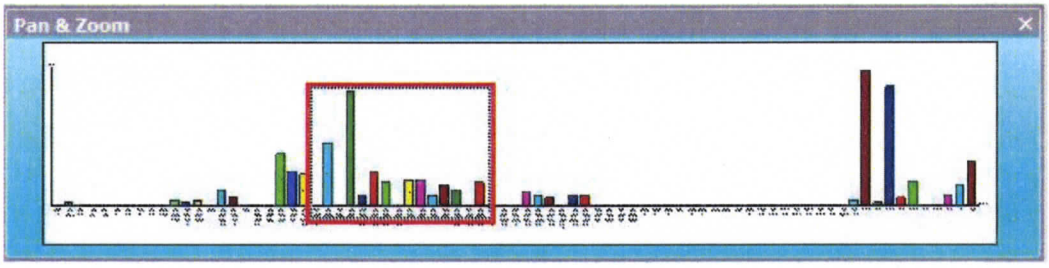


Figure 5.3: Chart for Response for Class (RFC)

Some of the advantages of using charts include:

- Charts have an attractive look and feel;
- Charts are intuitive to read and understand;
- Charts help users to quickly locate the data in which they are interested in; and
- Charts are a good starting point to conduct quick assessments, evaluations, and comparisons.

Exploration of source code statistics can provide the user with relevant information, such as:

- How many different data types are in the subject system?
- Which types occur most frequently?
- How are objects' metrics distributed over value ranges?
- Do the metrics results computed with low level source code artifacts differ from metrics results computed at a higher level of abstraction (i.e., coupling and cohesion at subsystem or package level)?

To help users to understand the complexity of the software system better, we provide statistical data that complement our metrics calculations. In Vimex, two kinds of summaries are provided:

(1) Summaries of source artifacts (e.g., number of data types). Two kinds of artifacts are distinguished: entities and relations. Each kind of artifact has its own data types. Statistical charts can be generated to summarize artifacts and artifact types. Figure 5.4 shows a typical statistics chart of the entity types in a target system.

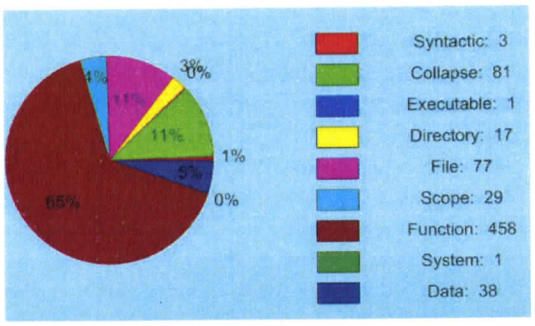


Figure 5.4: Entity Types in a System

(2) Summaries of metrics data. Figure 5.4 depicts a statistics generated from the metrics data in Figure 5.3: Chart for Response for Class. In this chart, there are 79 objects distributed over a value range from 0 to 52. It is easy to observe that most objects have a low value. However, the distribution is presented in a more concise manner by the statistical data, which shows that, for instance, 89 percent of the objects fall in the first quarter (0% -25%) of the value range (shown as red slice).

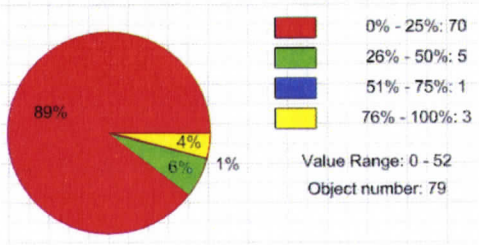


Figure 5.5: Data statistics over metrics results from Figure 5.3

5.3 Extending Visio with Metrics Master Shapes

By building a software metrics tool on top of Visio, one can take advantage of Visio's customization functionalities to provide seamless functionality that is consistent with the cognitive support that Visio users have already acquired [ZCKW03]. A typical example of cognitive support is Visio's drag-and-drop metaphor that is employed to create graphical objects on the canvas interactively.

Thus, our metrics tools should adhere to this metaphor (e.g., when the user creates a new metrics chart).

5.3.1 Drag and Drop

The drag-and-drop metaphor is a general diagramming and drawing technique in Visio and includes three steps: (1) open a stencil file; (2) select a master from it; and (3) drop the master onto a page to create an instance of the master [Micro03].

By creating masters for software metrics, metrics calculations are hidden beneath the drag-and-drop action: when a user drags and drops a metrics master, a pre-defined mechanism is triggered and a visualization of the metrics is generated automatically.

A typical drag and drop scenario is described in the following example: the process of generating a CBO metrics includes four steps: (1) open a new Visio file and the stencil which contains the metrics masters; (2) drag the "CBO" master and drop it onto the page; (3) a dialog will pop up and let the user select the data source to import (if the data model is empty); and (4) a CBO bar chart is generated automatically based on the source data. Figures 5.6-5.9 illustrate these four steps.

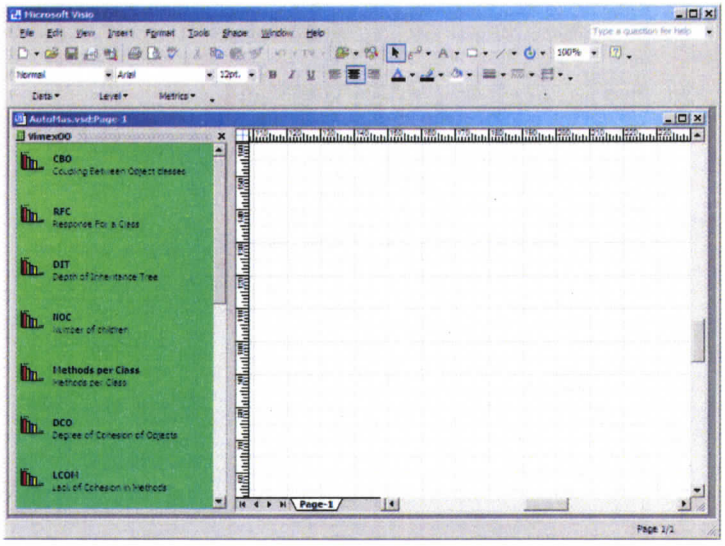


Figure 5.6: Drag and Drop—Step 1

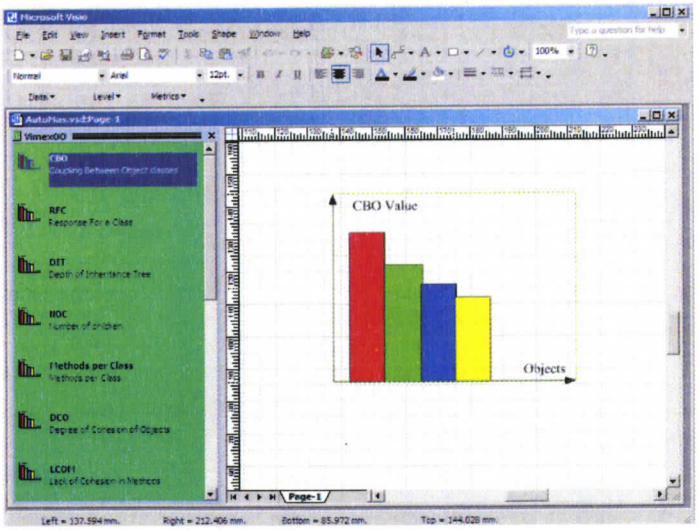


Figure 5.7: Drag and Drop—Step 2

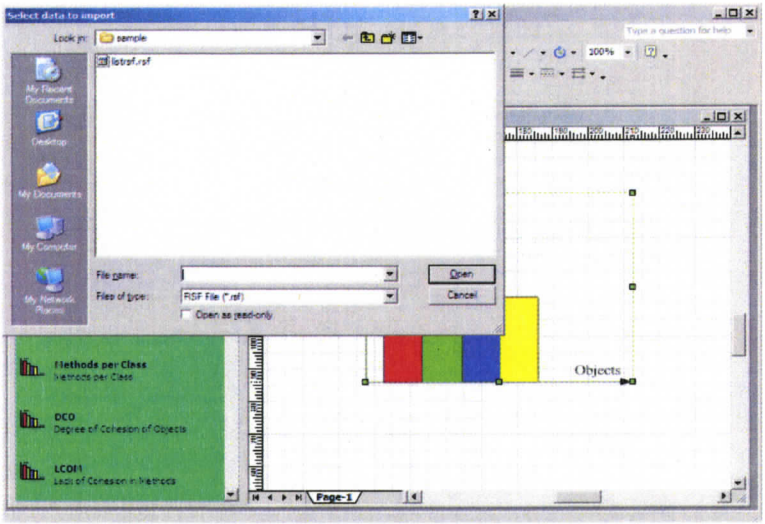


Figure 5.8: Drag and Drop—Step 3

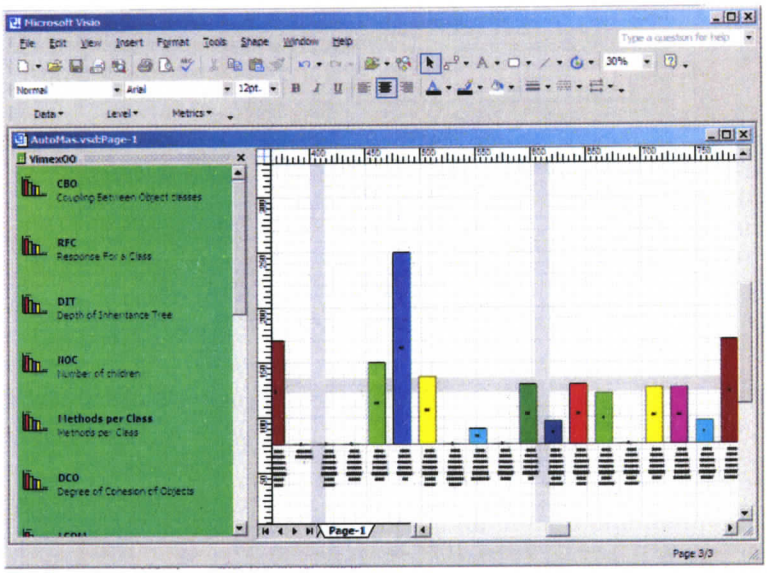


Figure 5.9: Drag and Drop—Step 4

The drag and drop scenario has been implemented with Visual Basic scripting and includes the following three design and implementation steps:

(1) Designing a master for a particular metric. Visio provides charting masters but, unfortunately, they cannot be edited directly and the data they can hold are limited. For example, the built-in bar chart in Visio can display only 12 bars even if the user defines more than that. Therefore, we had to design our own chart masters.

(2) Define the action triggered upon dragging-and-dropping the master. We found that Visio provides a drop event for master shapes, with which we can bind our action. An interface for the drop event can be found in the master's ShapeSheet—under the edit mode of the master by opening the master's ShapeSheet. In the "Events" section, there is a cell called "EventDrop", which will be triggered when the master shape is dropped onto a page. If we put

CALLTHIS("myProcedure",)

in the "EventDrop" cell, the procedure named "myProcedure" will be called in case of the drop event. The empty argument of the procedure indicates that the call will pass the newly created shape as the only argument. If the procedure needs more arguments,

one may list them after the empty argument (i.e., CALLTHIS("myProcedure", , arg1,arg2, ...)).

(3) Create a procedure called "myProcedure" with the corresponding argument list (i.e., myProcedure(shpObj as Visio.shape)), and a metrics calculation function.

5.3.2 Custom Properties

When metrics values are calculated, some meaningful information such as "the highest value" may be computed as well, which could become a useful reference for users in subsequent activities. A pertinent question is: How can such pieces of temporary data be stored in such a manner that they can be referred to later on? The most appropriate place for storing such data is the ShapeSheet. Every shape has an associated ShapeSheet and its contents can be accessed programmatically [Micro03]. A ShapeSheet provides a section named "Custom properties," which accepts user-defined data. "Custom properties" is by default an invisible section. Once this section is inserted into the ShapeSheet, the user can add data cells to it as depicted in Figure 5.10. If the "Custom properties" are defined in a master, all shapes generated by the master will inherit the same set of custom data cells.

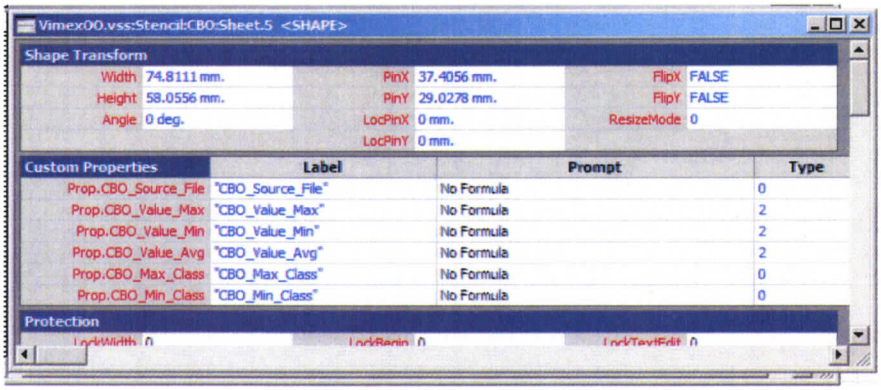


Figure 5.10: ShapeSheet with "Custom properties"

The following code snippet shows how to programmatically access "Custom properties" where "Shape" in the code snippet refers to the current metrics master shape:

```
' save max CBO value to cell CBO_Value_Max
Shape.Cells("Prop.CBO_Value_Max") = MaxCBO
' save the average CBO value to cell CBO_Value_Max
 Shape.Cells("Prop.CBO_Value_Avg") = AvgCBO
```

Supporting Visio's typical drag-and-drop procedure to create drawings is more intuitive for users and leverages cognitive support that users have already acquired. By taking advantage of ShapeSheet capabilities to define events and store data, users who are familiar with Visio can obtain information about how to expand the data, and how to extend the behavior of Vimex in a straightforward manner.

5.4 Summary

This chapter discussed selected implementation details of our metrics tool. Vimex imports source code artifacts in RSF, GXL, and SVG formats. Charts are then used to visualize the obtained metrics data. To facilitate metrics' creation, metrics masters are designed to effect a calculation using a simple "drag and drop" operation. In addition to metrics calculations, Vimex can compute data statistics to help users understand the complexity of the input data. By attaching metrics data to metrics charts, data can be computed and stored for reference purposes.

Chapter 6 Integration with REVisio

Vimex is a metrics tool prototype designed to help with software engineering activities. Since such activities require a larger toolset for the manipulation of diverse information, it is important that Vimex can be integrated with other (reverse engineering) tools. In this chapter, we describe a tool integration case study to show how Vimex can be integrated with REVisio (Reverse Engineering Visio). Vimex aids REVisio in providing better metrics support for software reverse engineering activities. This integration is a first step towards investigating how Vimex is leveraged in concert with other software engineering tools.

In the following, we briefly describe the functionality of REVisio, how we integrated Vimex with REVisio, and the resulting benefits of the integration.

6.1 REVisio

REVisio is a Visio application implemented under the auspices of the ACRE project at the University of Victoria [ZCKW03]. This reverse engineering tool partially implements the functionality of the Rigi reverse engineering environment [MHKK88, SWM97]. REVisio was realized using an adoption-centric tool-building approach [MWW03]. The ACRE project is based on the assumption that industrial developers are more likely to evaluate and eventually adopt a research tool that is

built using a COTS product with which they are already familiar, than an unfamiliar idiosyncratic stand-alone tool. In particular, ACRE assumes that research tools built using COTS products afford more cognitive support and provide more interoperability than traditional standard-alone research tools.

Just like Vimex, REVisio imports reverse engineering information from text files in formats such as RSF (Rigi Standard Format), RVG (Rigi View Graph), and GXL (Graph eXchange Language). To visualize the imported reverse engineering information, graphs are generated in REVisio based on the data extracted from the source files. A graph's nodes and arcs are rendered with Visio's basic shapes (i.e., rectangles and dynamic connectors, respectively). Additional graph-editing capabilities similar to the ones in the Rigi environment are supported (e.g., selection of forward trees) that allow to manipulate the graphs for reverse-engineering tasks.

6.2 Integrating Vimex and REVisio

Software metrics can provide important input to guide the software engineering process. REVisio is a reverse engineering tool that helps users to visualize the structure of software systems via graphs. Vimex offers functionality that is complementary to REVisio's capabilities, because metrics help users to measure and assess the properties of software systems. The integration of Vimex and REVisio, therefore, will support a broader range of reverse engineering activities in a tightly integrated manner. It supports users in performing typical reverse engineering activities for software system abstraction and understanding—as depicted in Figure 6.1—with four steps: (1) Both REVisio and Vimex can import source data that has been extracted from the target system's source code by Rigi or other reverse

engineering parsers. (2) Based on the data, REVisio can generate graph views of the target system. (3) REVisio provides an interactive graph editor for analysis activities; Vimex enhances the information that the editor provides with metrics data, which enables the reverse engineer to analyze the system more effectively. (4) To document results, Vimex can export metrics charts and REVisio can output system decomposition as hierarchical graphs.

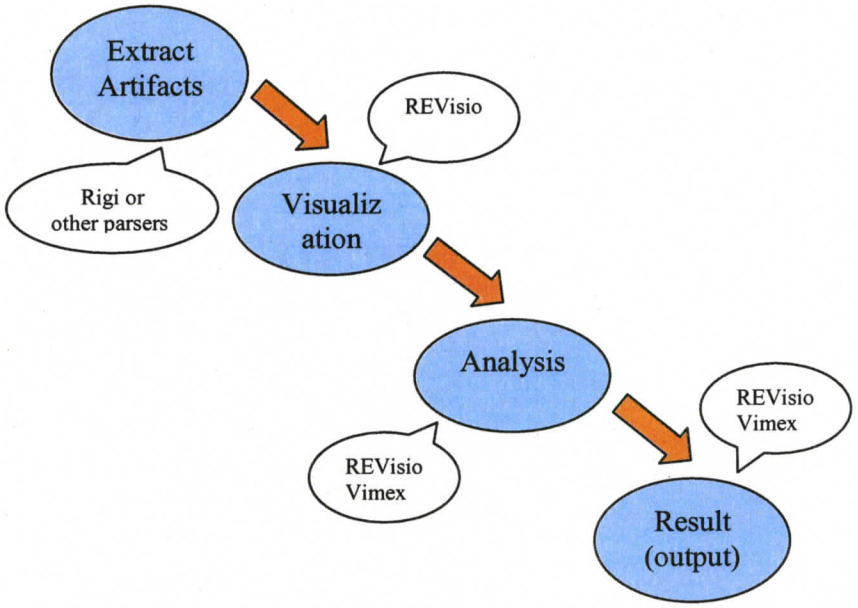


Figure 6.1: Reverse Engineering Activities Supported by REVisio and Vimex

A typical usage scenario of REVisio and Vimex that illustrates the benefits of integrating Vimex and REVisio is as follows: (1) a user first generates graphical views of a software system or subsystem using REVisio; (2) the user then navigates to the part of the system in which s/he is interested for further analysis (cf. Figure 6.2); (3) the user can now calculate metrics based on the data set available in the current view (cf. Figure 6.3/4) to gather important information about the subsystem, and to make decisions for the next step (e.g., to improve the cohesion of the subsystem).

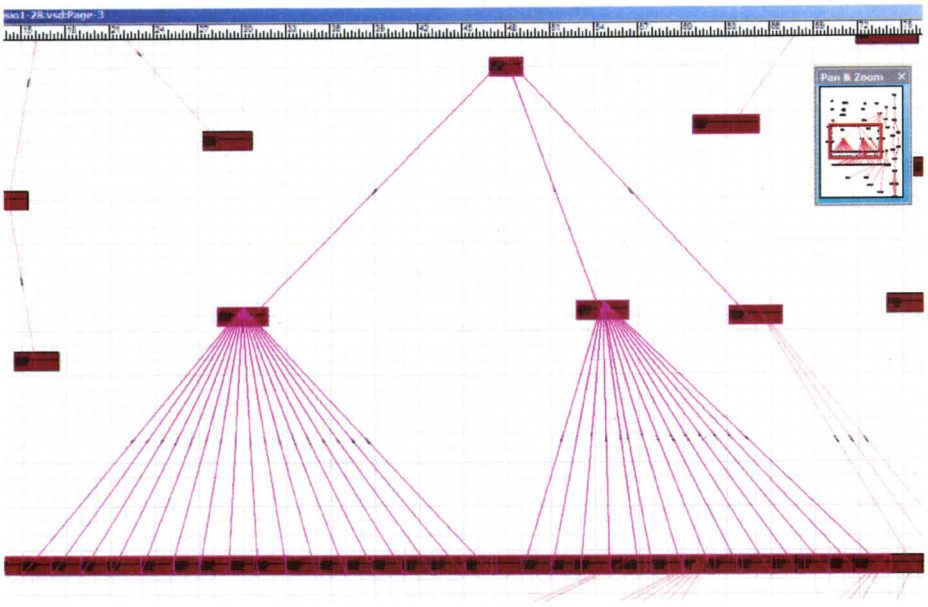


Figure 6.2: A system graph with partial data highlighted (in purple)

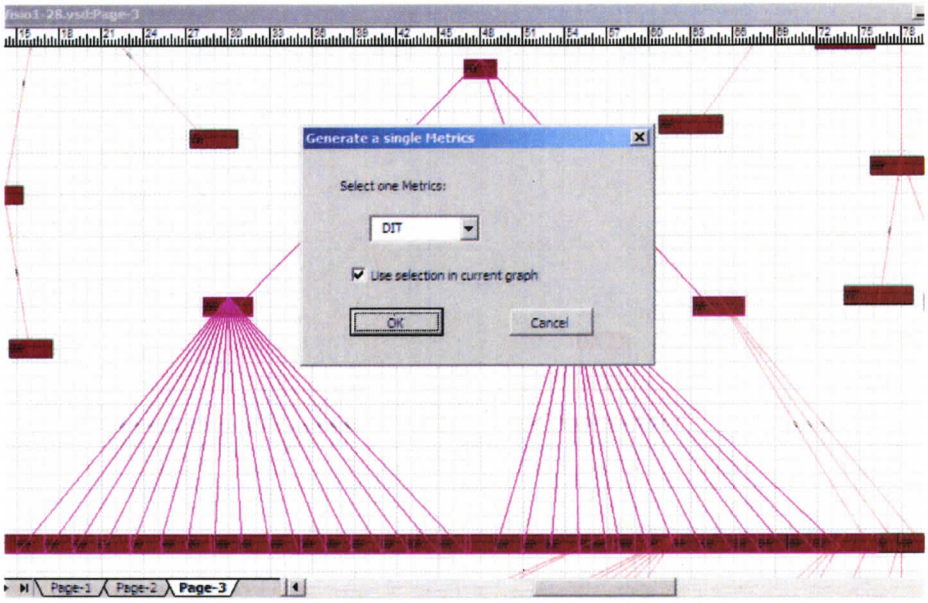


Figure 6.3: Calculating a specific metrics for the current selection

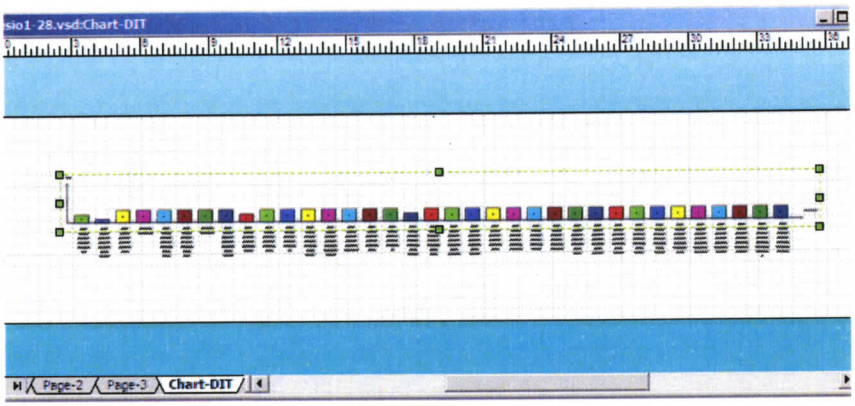


Figure 6.4: Generating a bar chart on a separate page

Since Vimex and REVisio are built using the same end-user programming environment and scripting language (i.e., VBA), it is easy for them to share internal data and interoperate on both the control level and data level. The integration is straightforward: by combining their function blocks into one Visio file, basic integration is already achieved. To integrate them seamlessly into one tool, a unified user interface was developed to overcome their differences.

6.3 Benefits of Integration

The integration of Vimex with REVisio creates a comprehensive tool for reverse engineering tasks. The benefits are as follows:

- The software metrics provided by Vimex complement the reverse engineering functionality of REVisio.
- After the integration, REVisio becomes a data provider for Vimex's metrics calculations. Therefore users can work with REVisio and instantaneously calculate metrics data.
- Vimex and REVisio help users throughout the entire reverse engineering process. Based on the metrics provided by Vimex, users can get more

information about the target system's software artifacts to guide their reverse engineering activities in REVisio.

6.4 Summary

In this chapter, we presented the integration of Vimex and REVisio. Both tools are built on top of Microsoft Visio. The integration of Vimex and REVisio combines reverse engineering functionality with metrics calculations, resulting in a more comprehensive tool-set compared to each individual tool. In the next chapter, an evaluation is conducted between Vimex and two other metrics tools.

Chapter 7 Evaluation

We proposed the AAA characteristics for building ideal metrics tools for research (cf. Chapter 1): adoptability, attractiveness, and appropriateness. To satisfy the AAA characteristics, we followed the adoption-centric methodology, which is based on the hypothesis that by improving cognitive support and interoperability, a software engineering tool is picked up more easily by industrial collaborators. We also employed visualization technologies, which are widely used to represent information in graphical formats to make numerical data more intuitive and attractive to end-users. We also identified and addressed the needs of different user groups, which lead to the support for different interaction levels in Vimex.

In this chapter, we evaluate Vimex by comparing it to two other metrics tools—CodeCrawler and Imagix 4D—with respect to the AAA characteristics.

7.1 Comparison with CodeCrawler

CodeCrawler is a software visualization tool that also combines a metrics tool and a software visualization tool [Lanz99, DDL99, SDML01, MLSD01]. It represents entities of software systems as graphs via nodes (e.g., class, method and attribute entities) and arcs (e.g., method calls, access to variables, and containment).

For visualization, metrics data are attached to a node (e.g., a rectangle) via its graphical properties (i.e., the width, height and color of a rectangle). Because the number of properties that a node can possess is limited, it is impossible to attach all the data from all of the available metrics to it. Therefore, choosing the proper metrics for a given node is important. Based on the node types (i.e., system, class, method, and attribute), CodeCrawler categorizes metrics into four corresponding scopes: system, class, method, and attribute scope. Only certain types of metrics should be attached to certain types of nodes. For example, it is more suitable to associate the "depth of inheritance" metrics with class nodes than attribute nodes.

The main features of CodeCrawler include:

- CodeCrawler provides various types of graphical layouts, including tree, correlation, histogram, and checkers (cf. Figure 7.1).
- CodeCrawler integrates metrics data into software artifact views. Usually metrics tools only provide metrics views, and software artifact visualization tools only generate software artifact views. CodeCrawler combines these two different types of views into a single one. Figure 7.1 (c) depicts an inheritance tree with metrics data attached to every node; the size and color of a node represent attributes provided by metrics data.

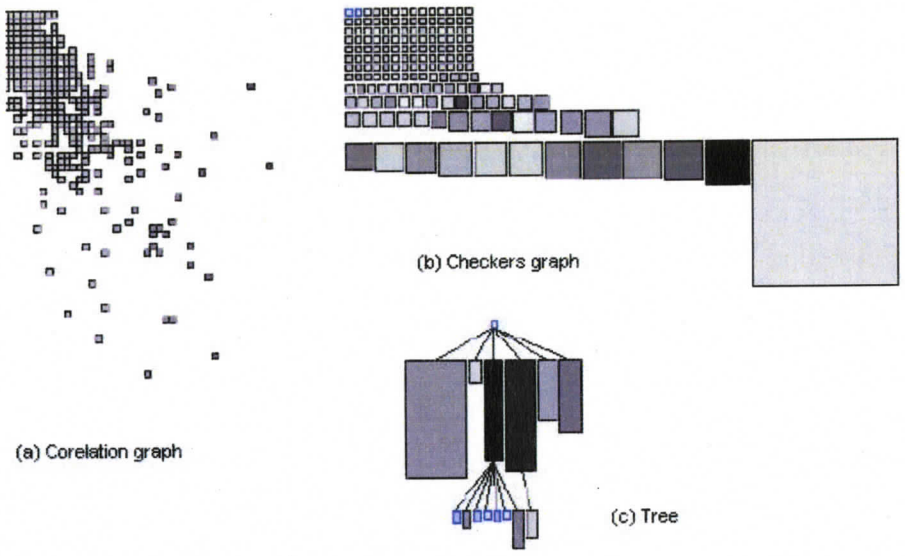


Figure 7.1: CodeCrawler's graphs [DDL99]

We compared Vimex and CodeCrawler with respect to our AAA characteristics and observed the following aspects:

- **Adoptability**. Vimex is constructed in the Microsoft Visio host environment with the VBA scripting language, which has a larger user base and a popular and mature user interface. CodeCrawler is written in Smalltalk and built based on Moose, which provides a reengineering environment and meta-model for CodeCrawler's reverse engineering functionality and metrics calculations. Based on the comparison of these two environments and the end-user programming languages, Vimex may have a better chance of being adopted because of the popularity of Microsoft Visio and VBA.
- **Attractiveness**. Both tools emphasize the integration of metrics visualization with artifact visualization to increase the utility of the tool; yet they have different approaches. CodeCrawler shows its strength by attaching metrics to a variety of graphs (cf. Figure 7.1) to help users

understand the system better. Vimex focuses on presenting metrics data in separate views. It uses a variety of charts to represent metrics data and statistics results effectively. The main visualization technique of CodeCrawler is graph visualization and thus metrics are also visualized using graphs as depicted in Figure 7.1. Being a diagramming tool, Visio is much more versatile with respect to its variety of visualization techniques and metrics can be visualized with the most appropriate technique.

- **Appropriateness**. CodeCrawler tries to deliver appropriate metrics data to users. It classifies metrics data into four scopes according to the types of software objects. Therefore, metrics data can be attached to related types of objects. Vimex is designed to represent metrics data in different interaction levels according to different user preferences. Users can choose to view detailed or highly-abstracted metrics data in the form of system properties. Thus, both tools pursue different strategies which have advantages and disadvantages in different application contexts.

7.2 Comparison with Imagix 4D

Imagix 4D is a reverse engineering tool that helps software developers to understand software written in C and C++. As an analysis and comprehension tool, Imagix 4D also generates a series of metrics to help users to identify problems in the development or maintenance stages.

We compared Vimex with Imagix 4D with respect to our AAA characteristics and arrived at the following conclusions:

- **Adoptability**. Imagix 4D has two parts: (a) a software reverse engineering component, and (b) a software metrics component. The metrics component

complements its reverse engineering component, so that potential users of the metrics functions are those who are already familiar with the tool's reverse engineering functionality. Launching metrics functions in Imagix 4D is as simple as one button click, but loading the source data requires more operations such as setting up the environment (an optional step), creating a new project, loading source code and header files, etc. Vimex is less complex on data import operation compared to Imagix 4D.

- **Attractiveness**. Imagix 4D does not provide any visualization for metrics data. Instead, it displays metrics data upon the selection of objects in a software view. As shown in Figure 7.2, the right side of the screen provides a structured view; the left side of the screen lists the metrics data of an object from the view (here "main"): A user can highlight different objects from the graph and view their metrics data. Imagix 4D can also attach color schemes to software views (i.e., use different colors to indicate values). Compared to Imagix 4D, Vimex identifies four types of visualizations and supporting them with different charts. Users can enjoy more options and flexibility with Vimex in gathering useful information.

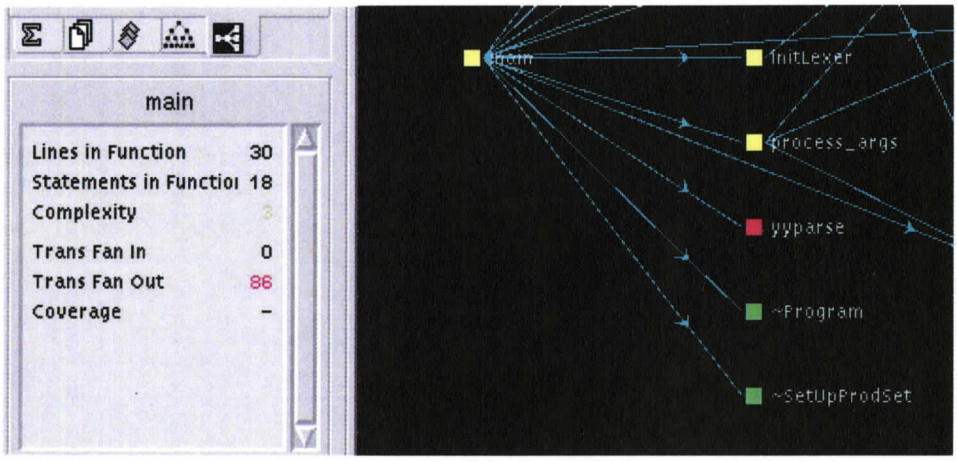


Figure 7.2: Metrics Data Besides a Software Graph

- **Appropriateness**. The target users of Imagix 4D are those software engineers who want to understand or maintain legacy software systems. Based on information gathered from metrics data, Imagix 4D provides a function named "Source checks", which looks for software problems in source codes and generates quality reports with suggestions for trouble-shooting. Vimex targets a variety of users, especially for those non-programming persons such as project managers. Chart is intuitive and attractive output format for them to get to understand software products. Imagix 4D, with its long development history, is a very polished tool targeting a specific and narrow domain. In contrast to Imagix 4D, Vimex does not produce dedicated quality reports for reengineering tasks. Over the years, Imagix 4D has developed a sizable user base and its users are intimately familiar with its user interface. Thus, Imagix 4D is essentially a COTS product albeit not as popular as Visio, and its limited customization options and extensibility would be a drawback in comparison with Visio.

7.3 Development Experience

Vimex is built on top of the Microsoft Visio COTS product. We demonstrated that with the ACSE approach a viable and useful metrics tool can be developed in relatively short time. While we have not conducted any formal experiments, we assess informally that the cognitive support afforded by the Vimex tool is probably better than by CodeCrawler (except for long-term users and the developers themselves), but probably not as good as Imagix 4D with its long development history. This could be a potential drawback because of existing idiosyncratic interactions that cannot be changed anymore because there is the risk of alienating the existing user base.

A mature and polished user interface is usually critical to a tool's success. Vimex "inherits" Visio's user interface, which has the same style as other Microsoft Office products, which are already popular among users. Thus, it is unnecessary for us to spend up-front research time and effort on designing and polishing user interface metaphors and interaction paradigms. Considering that a tool's baseline functionality may consume up to 90% of the resources and only 10% is left for research—this can be a huge saving.

Visio also provides small gadgets like the Pan and Zoom window, which can naturally be leveraged when exploring Vimex's visual outputs. The Pan and Zoom window contains the snapshot of an entire diagram. By dragging the pointer in the window, a user can zoom into a particular part of the diagram effectively.

Visio provides object models and a library of reusable components for developers to simplify coding of the operations of complicated widgets, for example the connectors in Visio diagrams. It is difficult for developers to maintain the links or connections between nodes if they are separate, disconnected shapes in the object model. In contrast, a connector automatically maintains the connections between two nodes when one or both of the nodes are moved. In addition, Visio may adjust the layout of the connectors so that the diagram becomes more readable (i.e., lines are not clustered together and have fewer crossing points).

While we were able to take advantage of Visio's diagramming support when implementing Vimex, we had to work around several limitations. A typical example of such a limitation is Visio's standard bar chart shape, which only supports a maximum of twelve bars. Our metrics data, however, needs potentially hundreds of bars, so it was necessary for us to create our own bar chart model.

Another drawback that we encountered was that we did not have full control over Vimex's maintenance and evolution (which obviously heavily depends on the evolution of its host tool, Visio and Microsoft Office). This drawback is not just a Visio-specific problem; all tools that are built on top of host tools face the same kind of problems. When a new version of the host tool is released, it may publish an updated object model. In the worst case it may happen that the changes of the host tool may affect a significant design choice of the add-on tool, so that developers have to re-design and re-implement (part of) the tool to accommodate the host tool's changes. However, the more popular a COTS product already is, the less likely this will be the case.

Especially in the beginning of Vimex's development we felt uncomfortable with the limited and lacking quantity of the available documentation provided by Microsoft Office. It proved difficult to find relevant information—especially programming APIs that we needed—in the MSDN library [MSDN06]. While we wanted to show that it is feasible to extend a COTS product without interaction with the developers of the host COTS product, it is certainly advisable to utilize appropriate mailing lists and newsgroups.

7.4 Summary

In this chapter, Vimex was evaluated against two other metrics tools: CodeCrawler and Imagix 4D. Compared to CodeCrawler (a tool that shows metrics data within system views), Vimex delivers metrics information separately through a variety of suitable visualization techniques, and offers different levels of metrics granularity that cater to different users' proficiency and background.

Imagix 4D provides a metrics tool inside its reverse engineering environment. The available metrics are fine-grained and exclusively target software developers. Compared to Imagix 4D, Vimex supports more sophisticated metrics visualizations and offers a broader range of metrics that go beyond the needs of software developers.

Finally, we summarized our development experiences with selected aspects and features of the host environment, which will hopefully prove useful for other researchers that want to follow the ACSE tool-building methodology.

Chapter 8 Conclusions

8.1 Summary

Software metrics are an important aspect of software engineering and will continue to play a central role in measuring and assessing software and process attributes.

Metrics tools are key to help users to collect, summarize, and abstract information of the subject software system they are trying to understand. To represent metrics data more efficiently, visualization techniques are applied to these tools. However, this is insufficient, because many other factors or non-functional requirement, such as cognitive support or interoperability, may also influence whether a tool is successful or not.

In this thesis, a new approach was followed for building a metrics tool, and a prototype of the tool, Vimex, was implemented. Our tool-building approach follows the ACSE methodology, which proposes to build tool functionality on top of a host component. We chose Microsoft Visio, a widely used diagramming tool, as the host, in order to take advantage of the cognitive support, interoperability, and powerful diagramming functionalities that Visio provides. Visio is based on the VBA Microsoft Office development environment and is able to interoperate with other VBA-

supported applications. Vimex provides metrics functionality with two user preference levels to meet the needs of different kinds of users. To evaluate Vimex, it is compared against two other metrics tools: CodeCrawler and Imagix 4D. This evaluation showed that Vimex is able to compete with the capabilities of these tools. We believe that the lessons learned from building Vimex on top of Visio—with its resulting strengths and weaknesses—will be beneficial for other researchers that plan to build a metrics or reverse engineering tool.

Building a successful metrics tool is a non-trivial endeavor. Vimex shows that the ACSE tool-building methodology is feasible in the sense that reverse engineering tools with sophisticated functionality can be realized with it, which are both useful and usable. In particular, Vimex is a sophisticated tool in the sense that it assists users in calculating and visualizing complex software metrics, helping them to perform reverse engineering and maintenance tasks. Metrics data are visualized via different charts, which may help a user to obtain metrics results accurately and effectively. Furthermore, it implements a new design idea of delivering different detail levels of metrics data to different user types.

When Vimex inherits much goodness from Microsoft Visio, we should also be aware of the trade-off of this approach. It is noteworthy that Vimex is restricted or limited by the host environment—Visio. As the designer of Vimex, we may not be able to think out of the box of the Visio supported features.

The contribution of this thesis is not only limited to implementing mere metrics functionality, but also the investigation of a general approach for building of (metrics) research tools.

8.2 Contributions

In this thesis, we investigated and discussed background information and related work regarding software metrics in an effort to provide a general and efficient way to build metrics tools. Vimex, a prototype tool, was built to illustrate our ideas.

The main contributions of this thesis are as follows:

- We proposed a framework of the AAA characteristics (i.e., adoptability, attractiveness, and appropriateness) for assessing metrics tools;
- We conducted background research and identified related work;
- We investigated the benefits and drawbacks of Microsoft Visio as a host environment for building metrics tools;
- We outlined a general solution to lower the barrier between users and metrics tools by introducing two granularity levels for visualizing metrics data with different details;
- We designed and implemented the Vimex prototype tool; and
- We described our experiences and lessons learned for further investigations in this domain.

8.3 Future Work

This section discusses our proposed future work Vimex is a metrics tool prototype with sophisticated capabilities. Still, we feel that further investigations are necessary on how to reveal software attributes with software metrics more effectively. We also would like to investigate more high-level, integrated metrics solutions to aid in the comprehension of a software system as a whole; for example, providing more views for the comparison or summarization of metrics data may reveal information about the development history and evolution of a software system.

It would also be useful to further improve on the usability of Vimex. In the presentation layer, wrapping more metrics details from non-experienced users might be a good starting point. In addition, investigating a flexible way for users to switch between metrics would be valuable to improve the tool's navigability through metrics data.

In the process of developing Vimex we designed some Visio metrics masters. Letting users develop their own metrics masters might be another worthwhile avenue to pursue. To further generalize the tool we can envision a mechanism that allows users to create their own graphical shapes and attach metrics data to them. With such a new master shape, Vimex would allow users to link data from several metrics to the new master shape to generate a new customized view. This feature should be useful for experienced metrics users who prefer to create their own specialized metrics views.

To evaluate Vimex convincingly, we plan to conduct a long-term empirical research based on the AAA characteristics and cognitive support. The following criteria will be applied in the empirical research:

- Evaluation will be conducted in a continuous way over different time periods.
- Random users will be chosen for use case study, including: Visio users, non-Visio users, project leader/architecture/managers, programmers and general users.
- Proper case study questions will be designed to evaluate the major factors of AAA characteristics and cognitive support, including how users feel about Vimex's interface (cognitive support); how difficult it will be for first time users to start using Vimex (cognitive support); what characteristic the users like most in Vimex and what is their most wanted

characteristic (Attractiveness); if the users like the metrics output in charts, if not, what is their preferred format (Attractiveness); when will users use Vimex (i.e., users' usage scenarios) (Appropriateness); if the users feel helpful with the metrics granularity levels corresponding to their tasks (Appropriateness); if the users are willing to use Vimex again or not and the reasons (Adoptability); etc.

Vimex is considered to be "good" if positive feedbacks are over a certain percentage (i.e., 50%) and the percentage shows a trend of increasing over certain time periods. The summary and feedback of the case study will be recorded for further development of Vimex.

Bibliography

[Arif93] A. Arifoglu. "A Methodology for Software Cost Estimation," *Software Engineering Notes*, 18(2):96-105, April 1993.

[BLMS04] R. Balzer, M. Litoiu, H. A. Müller, D.B. Smith, M.-A. Storey, S. Tilley and K. Wong (eds). *Proceedings 4th International Workshop on Adopton-Centric Software Engineering (ACSE 2004)*, Edinburgh, Scotland, UK, IEE Press, ISBN 0-86341-421-4, 92 pages, May 2004.

[BMP95] M. J. Bassman, F. McGarry and R. Pajerski. "Software Measurement Guidebook," Software Engineering Laboratory, June 1995. http://sel.gsfc.nasa.gov/website/documents/online-doc/94-102.pdf

[Brad03] S. Bradshaw. "Software Test Metrics—A Practical Approach," Questcon, Inc., March 2003. http://www.questcon.com/home.nsf/0/92273A9D4DF6A06785256D8000430632/$File/TFBMetricsWhitePaper.pdf?OpenElement

[BVT03] R. K. Bandi, V. K. Vaishnavi and D. E. Turk. "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics," *IEEE Transactions on Software Engineering*, 29(1):77-87, January 2003.

[CFYS02] C. Fillies and Y. Sure. "On Visualizing the Semantic Web in MS Office," *6th International Conference on Information Visualization*, pp. 441-446, London, England, 2002.

[DDL99] S. Demeyer, S. Ducasse and M. Lanza. "A Hybrid Reverse Engineering Platform Combining Metrics and Program Visualization," *6th IEEE Working Conference on Reverse Engineering (WCRE '99)*, IEEE, October 1999.

[DMRT01] Design Metrics Research Team. "An Introduction to Design Metrics Research," Ball State University, 2001. http://www.cs.bsu.edu/homepages/metrics/introduction/

[GKM05] G. Gui, H. M. Kienle and H. A. Müller. "REGoLive: Building a Web Site Comprehension Tool by Extending GoLive," *7th IEEE International Symposium on Web Site Evolution (WSE 2005)*, September 2005.

[HWS00] R. C. Holt, A. Winter and A. Schürr. "GXL: Towards a Standard Exchange Format," *7th IEEE Working Conference on Reverse Engineering (WCRE 2000)*, pp. 162-171, USA, November 2000.

[Imag03] Imagix 4D, Imagix Corporation, 2003. http://www.imagix.com/products/products.html

[Jack02] D. Jackson (editor). *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C, February 2002. http://www.w3.org/TR/2002/WD-SVG11-20020215/

[Kelv1894] W. T. Kelvin. *Popular Lectures and Addresses*, London; New York; Macmillan and Co.,1891-1894.

[Kien01] H. M. Kienle. "Exchange Format Bibliography," *Software Engineering Notes*, 26(1):56-60, January 2001.

[Kien06] H. M. Kienle. *Building Reverse Engineering Tools with Software Components*, Ph.D. Dissertation, Department of Computer Science, 350 pages, University of Victoria, 2006.

[KKPK73] K. W. Kolence and P. J. Kiviat "Software Unit Profiles and Kiviat Figures," *Performance Evaluation Review*, 2(3):2-12, September 1973.

[Lanz01] M. Lanza. "The evolution matrix: recovering software evolution using software visualization techniques," *4th ACM International Workshop on Principles of Software Evolution (IWPSE 2001)*, pp. 37-42, Vienna, Austria, 2001.

[Lanz03a] M. Lanza. "CodeCrawler – A Lightweight Software Visualization Tool", *2nd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2003)*, pp. 51-52, 2003.

[Lanz03b] M. Lanza. "CodeCrawler – Lessons Learned in Building a Software Visualization Tool," *7th IEEE European Conference on Software Maintenance and Reengineering (CSMR 2003)*, pp. 409-418, March 2003.

[Lanz99] M. Lanza. *Combining Metrics and Graphs for Object Oriented Reverse Engineering*, Diploma thesis, University of Bern, October 1999.

[Ma04] J. Ma. *Building Reverse Engineering Tools using Lotus Notes to Leverage Cognitive Support and Improve Interoperability*, Master's Thesis, University of Victoria, 2004.

[Meye98] B. Meyer. "The Role of Object-Oriented Metrics," *IEEE Computer*, 31(11):123-125, November 1998.

[MHKK88] H.A. Müller and K. Klashinsky. "Rigi—A System for Programming-in-the-large," *10th IEEE International Conference on Software Engineering (ICSE '88)*, pp. 80-86, Raffles City, Singapore, 1988.

[Micro03] Microsoft Visio 2003 Software Development Kit (SDK), 2003. http://office.microsoft.com/en-us/FX010857981033.aspx

[MKKW03] J. Ma, H. M. Kienle, P. Kaminski, A. Weber, and M. Litoiu. "Customizing Lotus Notes to Build Software Engineering Tools," *2003 IBM Centre for Advanced Studies Conference (CASCON 2003)*, pp. 211-222, October 2003.

[MLSD01] M. Lanza and S. Ducasse. "A categorization of Classes based on the visualization of their internal structure: the class blueprint," *16th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications (OOPSLA 2001)*, pp. 300-311, 2001.

[Morr89] K. Morris. *Metrics for Object-Oriented Software Development Environment,* M.Sc. Thesis, MIT Sloan School of Management, 1989.

[MSDN06] MSDN Library, Microsoft Corporation, 2006. http://msdn.microsoft.com/library/default.asp

[MSW03] H. A. Müller, M. -A. Storey, and K. Wong. "Adoption-Centric Software Engineering (ACSE) Project Synopsis," *ACSE 2003 Workshop,* Portland, USA, Collocated with ICSE 2003. http://www.acse.cs.uvic.ca/index.html, 2003.

[MWW03] H. A. Müller, A. Weber, and K. Wong. "Leveraging Cognitive Support and Modern Platforms for Adoption-Centric Reverse Engineering (ACRE)," *Proceedings 4th International Workshop on Adoption-Centric Software Engineering (ACSE 2003)*; pp. 30-35, ICSE 2003 Workshop; Portland, Oregon, USA.

[NFMN00] N. Fenton and M. Neil. "Software Metrics: Roadmap," *Conference on The Future of Software Engineering*, pp. 359-370, June 2000.

[PDPB04] P. Donzelli and P. Bresciani. "Improving Requirements Engineering by Quality Modeling—A Quality-Based Requirements Engineering Framework," *Journal of Research and Practice in Information Technology*, 36(4):277-294, 2004.

[Robe79] F. S. Roberts. "Measurement Theory with Applications to Decision-making, Utility, and the Social Sciences," *Encyclopedia of Mathematics and its Applications*, Addison Wesley Publishing Company, 1979.

[SCCK94] S. R. Chidamber and C. F. Kemerer. "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, 20(6):476-493, June 1994.

[Schn01] N. F. Schneidewind. "Investigation of the Risk to Software Reliability and Maintainability of Requirements Changes," *IEEE International Conference on Software Maintenance (ICSM 2001)*, p. 127, November 2001.

[SDML01] S. Ducasse and M. Lanza. "Towards a Methodology for the Understanding of Object-oriented Systems," Technique et science informatiques, 20(4):539-566, 2001.

[Sem01] SemTalk, Semtation GmbH, 2001. http://www.semtalk.com/

[SQHE01] A. S. M. Sajeev (Project Leader), A. Quigley, M. Hannaford, P. Eades, W. Wang and P. Kuwanoda. "Software Metrics and Visualization," The University of Newcastle, Australia, 2001. http://www.cs.newcastle.edu.au/Research/SERG/projects.html

[STSH02] S. Tilley and S. Huang. "Documenting Software systems with Views III: Toward a Task-oriented Classification of Program Visualization Techniques," *20th Annual International Conference on Computer Documentation (SIGDOC)*, pp. 226-233, Toronto, Canada, 2002.

[SWM97] M. -A. Storey, K. Wong and H. A. Müller. "Rigi—A Visualization Environment or Reverse Engineering (Research Demonstration Summary)," *IEEE 19th International Conference on Software Engineering (ICSE-97)*, pp. 606-607, Boston, USA, May 1997.

[THP03] S. Tilley, S. Huang and T. Payne. "On the Challenges of Adopting (ROTS) Software," *3rd International Workshop on Adoption-Centric Software Engineering (ACSE 2003)*, pp. 3-6, May 2003.

[Till95] S. Tilley. *Domain-Retargetable Reverse Engineering*, Ph.D. Thesis, University of Victoria, 1995.

[TKVM00] T. G. Kirner and V. F. Martins. "Development of an information visualization tool using virtual reality," *2000 ACM Symposium on Applied Computing*, 2, pp. 604-606. Como, Italy, 2000.

[WKM02] A. Weber, H. M. Kienle and H. A. Müller. "Live Documents with Contextual, Data-Driven Information Components," SIGDOC 2002, pp. 236-247, Toronto, Canada, 2002.

[Wong98] K. Wong. *Rigi User's Manual*, June 1998. Version 5.4.4, 168 pages.

[Wong99] K. Wong. *Reverse Engineering Notebook*, Ph.D. Thesis, University of Victoria, October 1999.

[ZCKW03] Q. Zhu, Y. Chen, P. Kaminski, A. Weber, H. M. Kienle and H. A. Müller. "Leveraging Visio for Adoption-Centric Reverse Engineering Tools," *10th IEEE Working Conference on Reverse Engineering (WCRE 2003)*, pp. 270-275, Victoria, British Columbia, Canada, November 2003.

Appendix A Microsoft Visio Object Model