Live Documents with Contextual, Data-Driven Information Components

Anke Weber
Department of Computer Science
University of Victoria, Canada
Victoria, BC V8W 3P6
+1 250-721-7294
anke@csr.uvic.ca

Holger M. Kienle
Department of Computer Science
University of Victoria, Canada
Victoria, BC V8W 3P6
+1 250-721-7294
kienle@csr.uvic.ca

Hausi A. Müller
Department of Computer Science
University of Victoria, Canada
Victoria, BC V8W 3P6
+1 250-721-7630
hausi@csr.uvic.ca

ABSTRACT
We introduce the notion of a live document and we describe our concept of live documents with contextual, data driven information components. The dynamic and interactive features of live documents provide a consistent data source for multimedia presentations targeted to various audiences and multiple platforms. Therefore, they contribute to the solution of key challenges in single sourcing and repurposing. We motivate the use of live documents with sample scenarios from the field of systems documentation. We further discuss how live documents can benefit from an interdisciplinary research approach across the fields of technical communications, systems documentation, and software engineering. Finally, we describe our experiences with prototype implementations of live documents based on Scalable Vector Graphics (SVG) and Microsoft Office Automation, respectively.

General Terms
Documentation.

Keywords
Live documents, systems documentation, single sourcing, repurposing, software engineering, reverse engineering, context-aware systems, software development environments, Scalable Vector Graphics, Microsoft Office

INTRODUCTION
The Internet moves increasingly closer to the user's desktop. For example, icons on our desktops are no longer shortcuts to local documents and programs only, but links to remote tools and applications that might reside on an anonymous server on the Internet. As a result, our desktop is morphing into "just another kind of Web browser", reducing the relevance of the user's operating system in the process. This trend has been one of the core issues in the so-called "browser-war", superficially centered on the market dominance of either MS Internet Explorer or Netscape Communicator [6].

The lines between Web browsers and office applications become blurred as office tools, such as Microsoft Office XP, Adobe FrameMaker, and Adobe Acrobat provide more support for Web interaction with every new release. For example, the user can copy tables from Web pages into Microsoft Excel, Microsoft Word document outlines are structured with hyperlinks, Web pages are started automatically from the office editor in the default Web browser, and, with the so-called "Web Capture" feature in Adobe Acrobat 5.0, Web pages are loaded and can be subsequently browsed in the PDF viewer.

In summary, desktop applications and documents become increasingly data-driven and "*live*": the data presented to the user is automatically kept current and in sync with the latest released version (e.g., via auto-deployment).

The traditional notion of documents as being mostly static entities is therefore changing towards the notion of constantly evolving information artifacts, so-called *live documents*. With emerging technologies, we can enrich office documents with multimedia components that provide for seamless data access, user interaction, customization, and group collaboration.

Mockus, Hibino, and Audris describe live documents, which they term *LiveDocs*, "as a framework for integrating and controlling information visualization (infoVis) components within Web pages to create interactive 'live' documents". They also discuss the notion of a live document as a "type of interactive document with embedded, contextual information visualization components" [23].

In our work, we generalize the notion of information visualization components to the concept of *contextual, data-driven information components (CDICs)*. The interactivity and data-driven features of live documents provide a consistent data source for multimedia document presentations targeted to various audiences (e.g., print and online user manuals that address both novice and expert users of a software system). Live documents are therefore a promising solution for consistency related problems in systems documentation, especially for single sourcing, for

repurposing, and for keeping the documentation consistent with the evolving source code.[1]

To address the problems above, our research on live documents draws from the fields of technical communications, systems documentation, and software engineering, in particular reverse engineering. Reverse engineering is the process of understanding legacy software system for future maintenance and evolution purposes [5]. It includes the re-documentation of the system under observation at different levels of abstraction. Re-documentation for reverse engineering faces the typical problems of documenting any software product; one of the key issues is to keep the re-documentation and analysis results in sync with the evolving system. Another important problem is the task of organizing and archiving of the obtained reverse engineering results, so that they are available for future system evolution. As a case study, we are using live documents to enhance the documentation capabilities of the Rigi reverse engineering tool [24].

Rigi is an interactive, visual tool designed for program understanding and software re-documentation. The core of Rigi is a graph editor enhanced with additional functionality for reverse engineering tasks. Rigi uses a graph model and abstraction mechanisms to structure and represent system artifacts. Nodes of the graph represent artifacts of the system and directed arcs represent artifact relationships. A simple example is a call-graph for a program. Nodes in the call-graph represent procedures of the program and arcs represent calls between procedures. Typically, different kind of nodes and arcs are visualized with different colors and/or shapes. Since legacy systems tend to be huge (several million lines of code), the graphs can contain thousands of nodes and arcs with different kinds of node and arc types.

The reminder of this paper is structured as follows. In Section 2, we motivate the use of live documents with sample scenarios from the field of systems documentation. Section 3 specifies the notion of a live document. In Section 4, we describe the concept of live documents with CDICs. In Sections 5 and Section 6, we describe our experience with prototype implementations of CDICs based on Scalable Vector Graphics (SVG), a new XML technology for encoding text and two dimensional graphics, and MS Office Automation, respectively. Section 7 reports on related work. Section 8 summarizes our work and outlines our future research agenda.

1. Scenarios of Usage for Live Documents

In this section, we briefly discuss our perception of systems documentation processes, their role in software development projects, and the role of technical communicators. On this basis, we describe two scenarios of live documents usage in the field of systems documentation.

Software development projects create, as well as depend on, a variety of technical documents such as requirement specifications, performance goals, functional specifications, design decisions, and maintenance logs. Software reverse engineering processes produce additional documents, diagrams, and system graphs (e.g., Rigi graphs). Technical documentation departments compose manuals and tutorials for the user of the software product. Systems documentation even influences project management decisions [33].

Consequently, systems documentation processes reveal company-wide collaborations between technical writers, software developers, application domain experts, and the marketing team. Technical writers need to acquire a fundamental knowledge of the system and they have to conduct thorough research (e.g., interviews of software developers and customers) to include the latest systems information available. They receive annotations from editors and proofreaders and they rewrite their documents many times accordingly.

The role of systems documentation has begun to change from *"a daunting task that is not done at all"* [26] to building an information architecture in parallel to the software development process [14] [2]. Despite the new perspectives, documentation remains low in priority in the software development cycle and often at the end of the software development process. Therefore, technical writers continue to work under hard deadlines [20].

Obsolescent documentation practices and shortening development cycles call for new documentation processes and methods that deal with the newly emerging requirements, understandings, and identities in the field of technical communications [10].

In the following subsections, we motivate the use of live documents as a contributing technology to these challenges.

1.1 Documents that synchronize with their changing context automatically

Detailed step-by-step descriptions of user interface screenshots are a common method to explain the system interface and functionality to the user. However, screenshots are one of the most time-consuming production and maintenance tasks in the documentation process [27]. If we replace the systematic instructions with a live document that embeds the application interface itself or a mimicking, rudimentary clone with extensive user guidance, we eliminate the need of keeping screenshots in line with an evolving user interface. At the same time, it allows the user to explore the system in place, without leaving the context of the document.

For example, consider the following scenario. A reverse engineer has worked with the SVG implementation of the Rigi tool in MS Internet Explorer and developed a call-graph for a ray tracer program that shows the main modules for dead code in the program. The reverse engineer needs to capture the view to integrate it in the online documentation as well as in a MS PowerPoint presentation for an upcoming group meeting of the software developers group. While the additions to the online manual are due immediately, the PowerPoint presentation is not due for another couple of days during which the reverse engineer is still working on the architecture and is expecting some important changes. Instead of repeatedly generating time-consuming screenshots, the reverse engineer embeds the SVG component in the online documentation (in fact, the reverse engineer could work directly in the documentation pages) as well as in the MS PowerPoint presentation. In this way, the online documentation as well as the MS PowerPoint presentation are up-to-date and synchronized. People attending the presentation will be able to visit

[1] Single sourcing is the production of multiple output versions and formats from a single source document, while repurposing refers to the reuse of information artifacts in different contexts [4].

the online documentation later, look at the graph and even explore new views of the system by themselves. Furthermore, during the presentation, the reverse engineer can interactively change the graph to demonstrate evolution processes or reveal more detailed insights of the software architecture. As the SVG components in the MS PowerPoint presentation and in the online documentation are both referencing the same data sources, any changes made during the presentation will be automatically reflected in the online documentation.

Figure 1 and 2 illustrate the above described scenario. In Figure 1, the SVG component that implements the Rigi tool is embedded in a MS PowerPoint slide and can be annotated using the MS PowerPoint editing tools. In Figure 2, the SVG component is embedded in MS Internet Explorer. The SVG component provides the same look&feel and interactive behavior in the MS PowerPoint slide show and in MS Internet Explorer: the user can drag nodes, select nodes with the mouse, and choose filtering, customization, and layout functions from the menus on the right. In Figure 2, the user has selected filtering according to node types and can now choose a node type from a sub menu.

In another scenario, the reverse engineer has worked with the implementation of the Rigi tool that is implemented on top of MS PowerPoint and developed a graph view of a ray tracer program that shows the main modules for dead code in the program. However, the next steps will be performed by a colleague, who is more experienced with the further steps of exploring the architecture and who will also do the presentation during the upcoming software developer's meeting. Consequently, the reverse engineer annotates the obtained view in MS PowerPoint and checks it into the company-wide version management tool.

The colleague checks out the MS PowerPoint presentation, changes the graph and adds annotations at the same time. After the presentation the MS PowerPoint file is checked back to the version management tool. A couple of weeks after the presentation, the technical communicators check out the MS PowerPoint presentation and turn it into a tutorial by adding some slides and applying the layout according to the corporate standards. As the documentation and the reverse engineering application reside in the same tool, documentation and source code have never been separated and they are both in sync. Please see Figure 6 for a MS PowerPoint implementation of the Rigi tool.

Another typical and hard to trace example of a last minute change is the renaming of variables in the source code [20] [31]. A data-driven document with live links to the source code can track these changes and update itself continuously.

The above scenarios illustrate how data-driven live documents can synchronize with their changing context automatically.

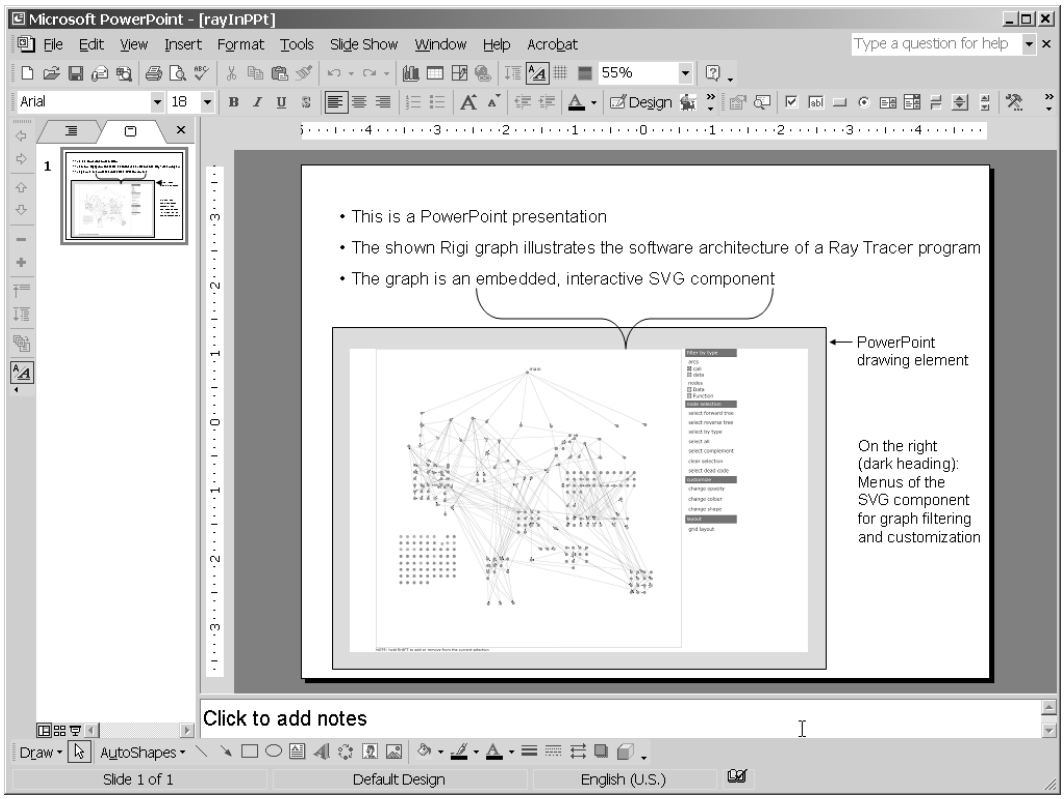

Figure 1. The SVG component, which implements the Rigi tool, is embedded in a MS PowerPoint slide and can be annotated using the MS PowerPoint editing tools.

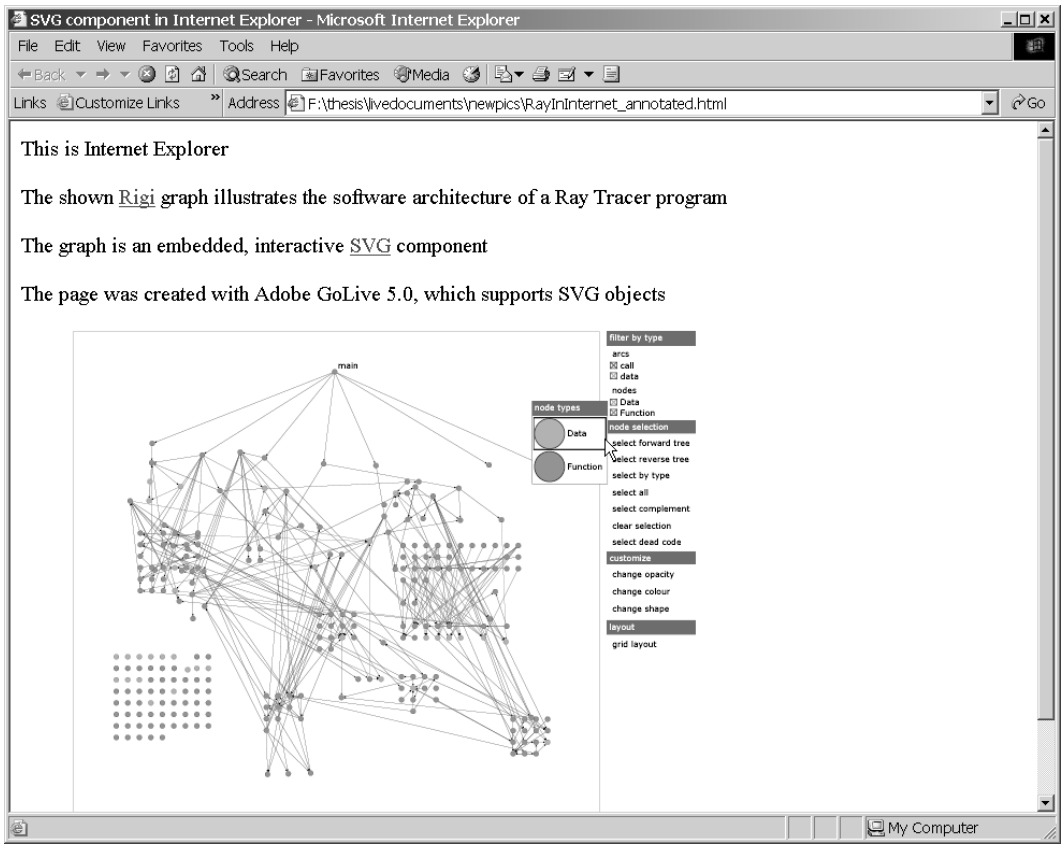

Figure 2. The SVG component, which implements the Rigi tool, is embedded in MS Internet Explorer.

1.2 Documents that rewrite themselves

The company's documentation pool is a collection of multimedia files in different formats such as simple text, PDF, HTML, and word processing formats such as Microsoft Word, Adobe FrameMaker and Adobe PageMaker. These documents and files commonly exist in multiple versions. For example, versions that are suitable for the Web, suitable for print and suitable for audio. Versions might be only slightly different. For example, an HTML document published on the company's Intranet might not contain proprietary data when published on the Internet while all other parts remain the same.

The production of multiple output versions and formats from a single source is a key issue in systems documentation. Hakkarainen describes the challenges of a single source operation as the management of constraints like editorial differences, including the choice of fonts and graphical elements, the targeting to different audiences, and the expected difference in writing style for online and print. Single sourcing for different formats appears to be one of the most costly tasks in producing systems documentation [12].

Another problem related to single sourcing, repurposing, and to the writing process itself, is the reuse of information components such as text blocks. Copy and paste operations are common for writing a document that is similar to a previous document (e.g., when the software version of the system only slightly changed). The writing process is a continued rewriting process, during which the writer accumulates multiple versions of information blocks that reflect a growing knowledge of the system, new facts,

A live document with CDICs can sense its context (e.g., the current editor tool) and act as a control instance for format, layout, and audience. It can adapt its contents to the recognized environment and, in the best case, rewrite itself automatically.

2. The Notion of a Live Document

We approached live documents on a broad basis with room for innovative ideas to identify their characteristics, possible application domains, and promising implementation tools and technologies. As part of our research, we organized several brainstorming meetings of the software engineering group in the Computer Science Department of the University of Victoria.

In summary, the participants described live documents with characteristics such as interactive, dynamic, flexible, mobile, and adaptable. They also raised concerns about the numerous perceptions of live documents, missing definitions, and appropriate classifications. They stressed the necessity to differentiate clearly between live documents and documents including graphics, animations, and simulations.

We were satisfied with the meetings as they raised key ideas for future research directions and started ongoing discussions in our group. We further concentrated on specifying requirements on live documents that result from these meetings, their role in the system documentation process (see Section 2), and from related work, in particular the LiveDocs framework [23]. The following requirements list describes the results of our data gathering processes. The list is subject to future revisions.

(R1) Requirement 1: Has a state
The core characteristic of a live document is its state. For example, to maintain editing changes of a traditional document, the user has to administrate different file versions of this document. In contrast, a live document that has a state can contain these changes as modifications of its state.

(R2) Requirement 2: Manage state automatically
A live document can change its state frequently during its life cycle. Live documents should manage their states independently and automatically. For example, the scenario of usage in Section 2.1 illustrates that it is one of the key challenges of systems documentation to keep the documentation in line with the evolving source code. This task requires frequent automatic updates of the live document's state. Explicit or default states of live documents allow the reader to read and manage live documents as if they are traditional static documents. Furthermore, the managing of live document states calls for reversible states and an integrated configuration and version management.

(R3) Requirement 3: Support for reuse
The importance of reuse results from the reusable nature of the writing process itself and in repurposing for different audiences and for versioning. For example, in Section 2.2 we describe how a document that can rewrite itself benefits the documentation process. Efficient rewriting however can only be achieved on reusable processes and components (e.g., text blocks).

(R4) Requirement 4: Adapt intelligently to the presentation
Intelligent automatic adaptation to the presentation media addresses key issues in single sourcing. For example, the transformation of print to online documents and vice versa is very difficult to achieve due to heterogeneous constraints on layout, space, and human reading behavior.

(R5) Requirement 5: Adapt intelligently to the reader
Intelligent adaptation to the reader supports targeting different audiences as required in repurposing. It also addresses personalization and customization issues. At the same time, live documents should support common reading behavior and document management (e.g., printing and saving) to facilitate adoption and to free the user of learning tasks outside the core system features.

(R6) Requirement 6: Support for advanced visualization
Visualization facilitates content comprehension and is an important factor in systems documentation. For example, in Section 2.1 we describes how a live document can replace the tedious process of describing detailed screenshot of the user interface by replacing them with a rudimentary, mimicking interface of the system itself.

(R7) Requirement 7: Support for contextual search and navigation
In contrast to traditional documents, live documents are no longer restricted to their document contents, but they can access external resources such as the system source code. Therefore, search mechanisms can extend the document content as well and include these external resources. For example, a search for a variable name can include the document contents as well as the source code and provide for internal and external navigation of the search results.

(R8) Requirement 8: Support for scripting
Scripts provide users of live documents with quick access to other states of live documents and allow them to perform repetitious tasks efficiently through automation. In particular, mechanisms for recording scripts can support personalization and customization. Pre-defined scripts free authors from having to explain how to accomplish various user interactions, and they enable authors to provide a series of user interactions in a single script [23].

3. Live Documents with Contextual, Data-Driven Information Components (CDICs)

Figure 3 illustrates the idea of a live document as a document that embeds a **contextual, data-driven information component (CDIC)** with the following characteristics:

- it is data-driven
- it is in context
- it is interactive
- it is adoption-centric

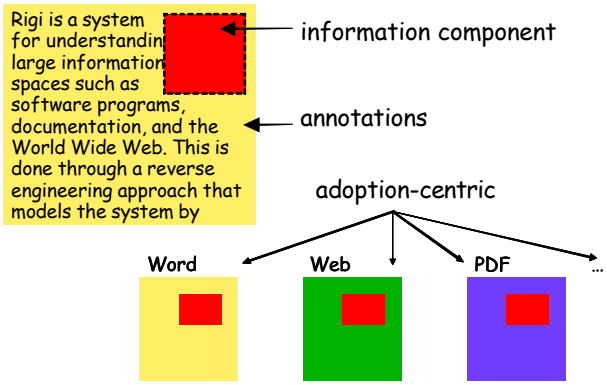

Figure 3. Live documents with CDICs

A **data-driven** information component reads and writes local and remote data. It is therefore well suited to incorporate dynamic and changing information with so far static documents and to synchronize and update the host document to a dynamically changing environment (see Requirement R1 and R2).

We distinguish different levels of **interactivity** for CDICs. **User interaction** allows the reader to customize the CDIC with personal data and to explore and analyze the data independently from the author (see R4). The *LiveDocs* framework shows that the so obtained results are more relevant for the reader's needs. It also illustrates how linked views with shared data add substantial analytic power [23]. We further distinguish between **local interoperability** between the CDICs of a document and **external interoperability**, which is necessary for the interaction with remote applications and data repositories.

The notion of being **"in context"** refers to CDICs that interact with their environment intelligently, e.g. through communication and adaptation. This characteristic addresses Requirements R4, R5, and R7. Greenberg describes context-aware computing as "*an important evolutionary step in computer use. Understanding context is vital if we are to build effective ubiquitous computing systems, information appliances, reactive environments, and computer devices that fit within architectural settings*" [11].

Context-aware systems are also useful for knowledge acquisition in systems with specialized terminology [19]. In systems documentation, this feature can support the user in understanding domain-related terminology.

The user, the host document, the presentation media, and the set of embedded CDICs with their internal and external relationships establish the context of a live document with CDICs. Consequently, its state depends on the incorporated data from the CDICs external data sources, on the performed user interactions, on the interactions between the CDIC and the host document, and on the interaction with possible other embedded CDICs.

Adoption-centric CDICs aim at supporting the user's every day work in the best possible way. Experience shows that research environments are not easily adopted by industry as they often lack the consistency and quality required in a commercial environment [15] [16]. We aim at overcoming these adoption problems by implementing CDICs on top of standard office environments. Thus, the user will only need to learn a small new extension of a familiar tool and can still exploit the accumulated cognitive processes in working with this tool [37]. We can therefore exploit the traditional document management processes as specified in Requirement 2.

Our component-based approach naturally supports reuse oriented research as specified in Requirement 8. There is also no limit as to the visualization features of the CDICs (see R6). In reference to requirement R8, it is valuable for our work that many standard office platforms offer support for recording and replaying scripts (e.g., based on macros in MS Office and on JavaScript in SVG).

We have implemented prototypes of CDICs with SVG plug-ins and COM components. The main challenge in these implementations is to balance the requirements outlined in Section 3. For example, if a CDIC is developed for use with multiple platforms, its data-driven features provide for a consistent use of dynamically changing data. However, in order to overcome adoption-related problems, the prominent challenge is to apply the CDICs context-aware features in a way that blends the CDIC homogenously with its environment. For example, if we embed the CDIC in a HTML document for display in Internet Explorer, it should allow for a Web based navigation style. If we embed it in a MS PowerPoint slide, it should allow for a menu and tool-based navigation style. In general, the CDICs should adapt to the look & feel of the host platform. The CDIC should also allow for interoperability with its host document. For example, in reference to Requirements 5 and 7, the tools search feature should be applicable to the host document as well as to the CDIC.

We have shown above with references to the specified requirements that embedding a CDIC contributes to the transformation of a traditional document into a live document.

In the following sections, we report on our prototype implementation experiences of CDICs with SVG plug-ins and MS COM components, respectively.

4. Prototype Implementation of CDICs with SVG Plug-Ins

As mentioned in Section 1, SVG is a new XML grammar for encoding text and two dimensional vector graphics. It is a World Wide Web Consortium (W3C) recommendation and an open Web standard [35]. The integration of SVG components with Web browsers and standard office platforms is supported by many commercial vendors, including IBM, Sun, Microsoft, Corel, and Adobe, which are among the members of the W3C SVG committee [1].

So far, many of the commercial vendors support the integration of SVG components with Web browsers and standard office platforms in a plug-in oriented style. For example, SVG comes with high-quality visualization engines such as the Adobe SVG viewer and Batik. These viewers provide functionality for scrolling, zooming, loading, saving, and searching. The Adobe SVG viewer is part of various host tools such as MS Office. SVG is therefore a promising technology for a component-based implementation of live documents on top of standard office tools.

SVG files can be used as standalone document types and they can be embedded inside HTML or XML, which adds to the transparency of live documents based on SVG.

SVG drawings can be interactive and dynamic. Animations are defined or triggered either declaratively or via scripting. Sophisticated applications of SVG are possible with a supplemental scripting language (e.g., JavaScript). A rich set of event handlers (e.g., *onmouseover* and *onclick)* can be assigned to any SVG graphical object.

The following list gives an overview of the basic SVG features:

- **Plain text format**
  SVG files can be read and modified by a range of tools and by end-users.
- **Open standard**
  SVG is an open W3C recommendation developed by a cross-industry consortium. Unlike other graphics formats, SVG is not proprietary.
- **Efficiency**
  SVG files are usually smaller and more compressible than comparable JPEG or GIF images.
- **Scalable and Print Resolution**
  Unlike bitmapped GIF and JPEG formats, SVG is a vector format. Thus, SVG images can be printed with high quality at any resolution. SVG graphics are immediately available for print and online use. There is no further need to maintain two different versions due to different image file formats, e.g. JPG and GIF for fast downloading of Web pages versus high resolution images in TIF, postscript, or bitmap format for printing.
- **Zoomable**
  You can zoom in on any portion of a SVG graphics without seeing any degradation in quality.
- **Searchable and selectable text**
  Unlike in bitmapped images, text in SVG text is selectable and searchable.
- **Scripting and animation**
  SVG enables dynamic and interactive graphics and is JavaScript ready.

- **Works with Java technology**
  SVG works with the SUN Java technologies' graphics engine [30].

- **XML grammar**
  As an XML grammar, SVG offers all the advantages of XML such as interoperability, internationalization (e.g., Unicode support), wide tool support, manipulation through standard APIs (e.g., the Document Object Model (DOM) API), and transformation through XML Stylesheet Language Transformation (XSLT).

4.1 Rigi Graphs in SVG

As part of our research, we re-implemented the visualization of Rigi graphs in SVG [17]. Figure 4 shows a complex Rigi graph in SVG with interactive features for selecting and filtering nodes and arcs. The graph visualizes parts of IBM's SQL/DS systems, which consist of over one million lines of code. The graph contains 928 nodes and 4203 arcs.

The use of SVG allows us to display Rigi graphs consistently on different platforms, in particular in MS Internet Explorer and in MS PowerPoint. The Rigi graph in SVG provides the same interactive features on both platforms. It is therefore a good example of a live document, in which a full-fledged user interface replaces the traditional screenshots. This type of live document can synchronize with its changing context automatically (see Section 2.1).

We have not yet implemented a thorough contextual and adoption-centric approach. For example, the SVG components do not adapt to the specific look&feel of either tool. However, if users embed SVG documents into MS PowerPoint, they can use MS PowerPoint's functionality to organize and annotate the SVG documents.

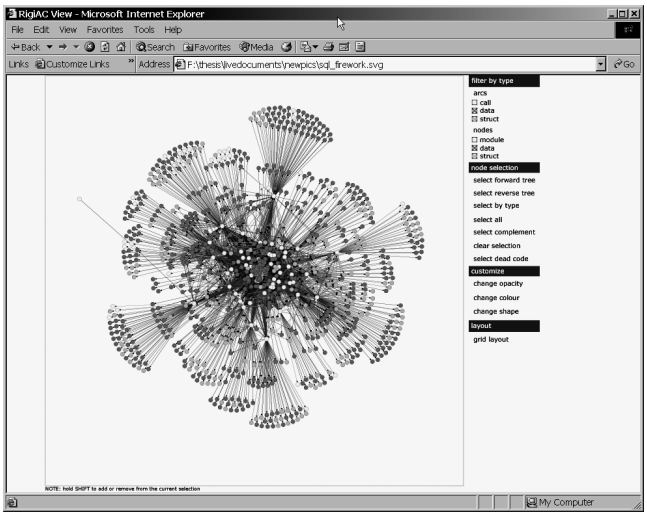

Figure 4. Rigi graph in SVG, displayed in Internet Explorer

4.2 SVG Implementation Experiences

We experimented with Internet Explorer, with Microsoft Word XP, and with Microsoft PowerPoint XP for embedding SVG components, and with Adobe Illustrator 9.0 and 10.0 for creating them. So far, we made the following observations:

- The combination of a small file size with a high-resolution makes SVG equally suited for efficient Web access and high-quality printing. For example, the SVG files size of Figure 4 is 224 KB compared to a high-resolution screenshot of the same graphic of more than 9 MB, which we produced earlier for a poster.

- The implementation of complex interactions (e.g., graph filtering) was fast in comparison with the same task done in Tcl/Tk for the Rigi user interface.

- We generated Rigi graphs both manually and semi-automatically based on intermediate textual Rigi views of the graph. As SVG is not a high-level language, manual programming is tedious and repetitious. Consequently, automatic generation of SVG files is the appropriate approach for SVG file creation.

- Loading of SVG components does not scale very well; all used tools freeze for a recognizable time while loading the graph of Figure 4. Once loaded, the graphics perform efficiently in Microsoft PowerPoint XP and Internet Explorer but they slow down document behavior, e.g. opening a slide with a SVG component requires the same amount of time for initially loading it. Similarly, scaling SVG components (e.g., the one in Figure 4) freezes the tool for a recognizable time, which is different to common graph formats such as JPEG or TIF.

- The integration in the MS Office tools is only rudimentary. For example, in MS PowerPoint, inserts the SVG component as an object and sets the path explicitly in the preference option of the SVG component instead of inserting it from the "Insert" menu via mouse click. We developed a simple MS PowerPoint macro that works similar to the common process of opening a supported file format.

- We performed authoring of SVG files with Adobe Illustrator 9.0, which so far only supports the creation of SVG files. Adobe Illustrator 10 adds support for reading and writing SVG files and for data-driven graphics. We have not yet fully tested SVG file authoring with Adobe Illustrator 10.

5. Prototype Implementation of CDICs with COM components

Office application development is the process of customizing an MS Office application to perform some function or service. MS Office automation can range from writing a simple Microsoft Visual Basic for Applications (VBA) procedure to creating a financial analysis and reporting application. Office automation allows developers to use Microsoft VBA code to create and control COM objects exposed by any application, dynamic-link library, or Microsoft ActiveX control that supports the appropriate programmatic interfaces. VBA and automation make it possible to program individual Office applications, as well as to run other applications from within a host application. For example, you can run an instance of MS Excel from within MS Access to perform mathematical and analytical operations on your MS Access data.

Office XP offers new features that fit well with our approach on live documents and that are especially well-suited for advanced documentation purposes, e.g. Smart Tags, XML support, Web based searching, group collaboration, and dashboards. For example, Smart Tags allow the author to scan and mark the text and present the user with a choice of action, e.g. to retrieve data from the Web or a database.

In Subsection 5.1, we present a prototype implementation of live documents for the Rigi tool and in Subsection 5.2, we summarize our experiences with this implementation.

5.1 Rigi Documentation with Office Automation

In reverse engineering, the term documentation becomes a synonym for the analysis results of reverse engineering tools (e.g., graph visualizations of the system artifacts). The term documenting-in-the-large describes these high-level approaches to architecture and design documentation. In contrast, documentation-in-the-small refers to source code annotations and detailed design documents [34].

So far, reverse engineering seems to concentrate on reverse engineering tools for documenting-in-the-large without providing the support to document-in-the small. There is also little tool support for explaining and describing the obtained analysis results and for subsequently organizing and archiving them, so that they are easily accessible and available for system evolution.

For example, Rigi provides graph analysis reports for documentation purposes [24]. However, these reports are more byproducts than documentation tools in their own right.

Therefore, our implementation aims at enhancing the documentation features of the Rigi tool with a basic statistical analysis, a slide show of the analysis results, and consecutive "shots" of Rigi graph views for their automatic inclusion in a MS PowerPoint slideshow.

For the statistical analysis, we apply built-in functionality of standard office tools to Rigi graphs. For example, we programmed MS Excel XP to import a Rigi graph and to perform a statistical analysis, which measures the complexity of the graph (e.g., the number of nodes and graphs with respect to their different types such as data and function types). We visualized the statistical results with MS Excel charts. MS Excel automatically updates the analysis results and charts when the graph changes (see Figure 5).

We extended MS Excel and MS PowerPoint with the following VBA functions, which are accessible from a "*Rigi*" menu in the MS Excel toolbar.

- **ImportRSF**
  This function imports a file with the graph model in the text based Rigi Standard Format (RSF) [38].

  Statistics
  This function performs a basic graph analysis on a previously imported RSF file, e.g. it calculates the number of nodes. The calculation exploits build-in MS Excel functions only. The calculation writes the analysis results in a new MS Excel sheets and adds a graphical chart to this sheet, which illustrates the results.

- **Copy to Slide**
  This function starts MS PowerPoint and copies the graph chart on the active sheet to a new PowerPoint slide.

- **PowerPoint Slide Show**
  This function opens a file dialog window to select a Rigi view file. It starts MS PowerPoint and draws the Rigi graph on a new slide.

- **Rigi View**
  This function automatically starts the Rigi tool.

- **Rigi Demo**
  This function includes the subentries "List Demo", "Ray Demo", and "SQL Demo". For example, the "Ray Demo" function starts the Rigi tool and starts the build-in Rigi Ray Tracer demo.

- **Reset**
  This function deletes the active sheets from the active Excel workbook and removes the Rigi menu from the Excel toolbar.

Figure 5 shows the MS Excel menu and the statistical analysis results that describe the complexity of the imported graphs in terms of the number of nodes and types classified according to their types. The MS Excel chart visualizes these results.

Although this example might appear simple, the implementation in a programming language like C or Tcl/Tk would require complex graph algorithms and visualization techniques without the flexibility and transparency of the MS Excel mechanisms. The statistics can be part of live reports that replace the traditional Rigi reports mentioned above.

Furthermore, we will integrate dynamic Rigi graphs with these live reports and with the Rigi user manual. We added a button "Slide" to the Rigi tool. This button starts MS PowerPoint automatically and captures the current Rigi graph view. This graph view is drawn in a new MS PowerPoint slide. The nodes and arcs of these graph views are MS PowerPoint objects. This functionality allows the reverse engineer to capture consecutive views of the graph to document continuous insights in a professional slideshow environment. The author can further enhance the slideshow of the graph views with annotations, additional graphics, and even animations by using the original MS PowerPoint functionality.

Figure 6 illustrates the workflow described above. On the top, you see the Rigi toolbar. The window on the left shows an original Rigi graph view. The small window that overlaps the Rigi toolbar in the top right corner shows the button "Slide". Pressing this button starts MS PowerPoint and reproduces the Rigi view with MS PowerPoint drawing objects that are displayed in a new MS PowerPoint slide.

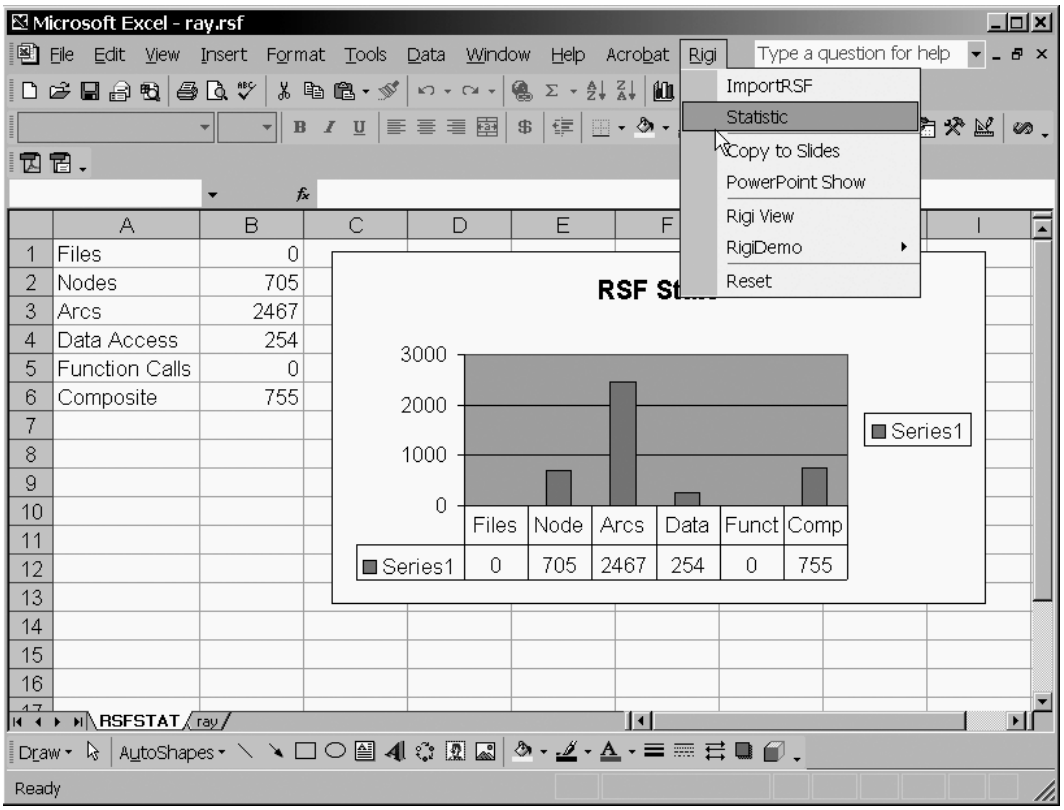

Figure 5. Analysis report for a Rigi graph in MS Excel that updates automatically when the data source (the graph information) changes

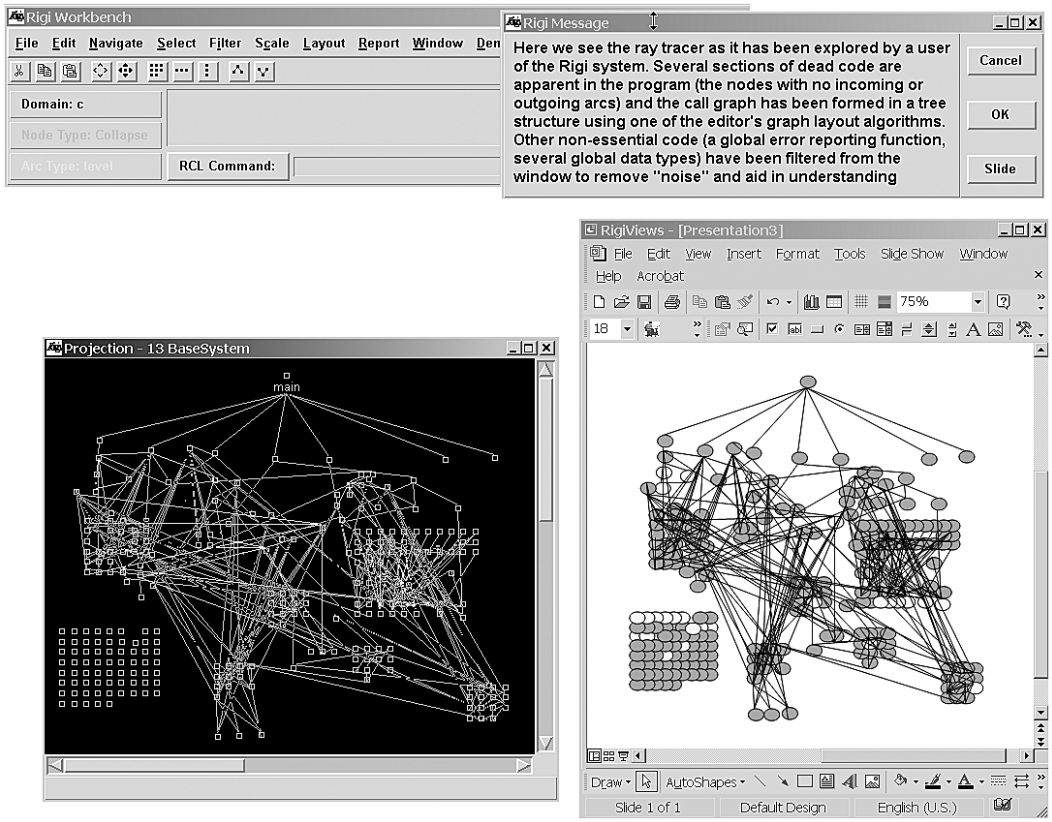

Figure 6. Rigi graph in the original tool (left) and as a MS PowerPoint drawing (right bottom window)

5.2 MS Office Automation Implementation Experiences

One of the keys for effective MS Office automation is to understand the MS Office object models and the often-inconsistent Microsoft terms and technologies (e.g., DCOM, COM, ActiveX) as well as the installation procedures and the scope of packages like Office XP developer. This is even more difficult as the documentation and the relevant Microsoft Web pages are complex and hard to understand.

We also found that we need to set up a costly MS Windows server environment for using advanced Office XP features in the future (e.g., Dashboards, group collaboration features, and SQL repositories).

So far, the visualization of Rigi graphs with MS PowerPoint drawing objects for the nodes and arcs scales very poorly. For example, loading the Rigi graph in Figure 4 as a MS PowerPoint drawing needs about three times longer than loading the corresponding SVG plug-in. Once the MS PowerPoint graph is drawn on the slide, changing slides performs normal and efficiently fast in contrast to the behavior of the SVG plug-in (see Section 5.2).

6. Related Work

The information visualization components in the LiveDocs framework are Java applets placed in a standard HTML web page [23] [22] [9] [3] [8]. The applets add interactivity to statistical tables, which the reader can explore and customize with mouse clicks. With the support for HTML and JavaScript, authors can include any of the standard control widgets available in HTML (e.g., check box button, radio button, input box, and drop-down choice menu), and attach JavaScript function calls to an HTML link. The standard applet parameters allow authors to specify links between the applets, and to initialize and customize the set of controls. Views are linked if a user's selection in one view automatically propagates the corresponding selection to all other linked views. Authors can initialize parameters to display key results when the applet is first loaded, e.g. by selecting and highlighting a subset of values. In this regard, the applets are used to create static documents (e.g., to present summaries of key results). The LiveDocs framework has been used to facilitate collaboration among researchers on a project and to disseminate results to researchers inside and outside of the project. Summary information of key results provided managers with feedback on how the software of their department had evolved over time as well as hints to how they might restructure their code or their organizations to improve the software engineering process.

In comparison with the LiveDocs framework, live documents with CDICs do not primarily target information visualization. They are an abstract view of embedded components that add interactivity and data synchronization to their host document. We also emphasize the contextual dimension of live documents to explore their ability to adapt to their environment automatically. Besides Web documents, we focus on documents that are created with standard office platforms.

The Reverse Engineering Notebook is an environment for reverse engineering analysis [39]. The main idea is to document the continuous understanding of the system during the software evolution process in a similar way to maintaining a scientific journal lab. This is achieved with recordable scripts that generate the documentation dynamically and therefore adapt to changing data sources and contexts automatically. The Reverse Engineering Notebooks has been partially implemented on top of the Rigi tool and we will include results of this work in supporting the scripting features of our approach.

Javadoc generates Web-based API documentation for the Java programming language from documentation comments in the source code [29]. However, the generated Web pages are not linked to the actual source code. When updates in the source code happen, the documentation must be regenerated.

The Briefing Associate is a tool that produces semantic markups as authors compose briefings [32]. The Briefing Associate is a research project that builds on the MS PowerPoint platform. It is therefore a valuable resource for our implementation related issues of CDICs based on COM components.

7. Summary and Future Work

This paper introduces live documents as a contributing technology to address key consistency related problems in systems documentation, especially in single sourcing, in repurposing, and in keeping the documentation consistent with the evolving source code. We specify live documents as documents that embed contextual, data-driven information components (CDICs). This implementation independent specification provides us with the basis for our future work that draws from an interdisciplinary research approach across the fields of technical communications, systems documentation, and software engineering, in particular reverse engineering.

As live documents are a means to mediate complex technical information, we further require an **authoring framework** that frees the author from any implementation tasks. Live documents should therefore be written and composed in much the same way as traditional static documents. Among others, we can achieve this with using standard office platforms that are familiar to the author, and with templates, population mechanisms, and mechanism for data sharing.

In contrast to proprietary file formats (e.g., Macromedia Flash [18]), text-based technologies like SVG and office automation add the feature of **"**openness**"** to live documents: the user can read and manipulate the implementation of the CDICs. In the worst case, the user can even destroy the implementation of the live document. On the other hand, adaptation issues can be delegated to the author's and even to the user's side. The user can easily incorporate small changes without requesting or waiting for version changes from the author. The framework should therefore provide **extendibility and documentation** for users who plan to adapt the framework to their needs (e.g., APIs, libraries, and scripts).

The importance of **reuse strategies and technologies** has long been recognized in software development, where research in software reuse is ongoing. Delisle describes that one of the primary obstacles of reuse in software development is the lack of reuse intentions of the software developers [7]. Although software reuse can improve both the quality and the productivity of software development, software developers might not anticipate that a certain component exists, or they do not want to search for it. Delsile suggest the integration of active information delivery in

reuse retrieval systems to overcome the adoption problems of reuse engineering. In contrast, reuse problems are a well-recognized problem in the technical communications community. Consequently, our planned comparison of cognitive models and reuse tools and technologies can benefit both fields.

A feasible approach for reuse related research as well as for establishing context for live documents is the integration of structured writing approaches in our work, in particular the **Information Mapping** method [13]. The Information Mapping method provides a broad definition of information types for documentation texts [28]. These information types can be used and extended to classify chunks of text for establishing context, e.g. for customization, storage, retrieval, and search issues.

Advanced context related research is of primary importance for the development of **contextual search mechanisms**. For example, Priestley describes how context-sensitive search interfaces for dynamically assembled documentation can replace the conventional table of contents [21].

For a **formal specification** of describing and establishing context for CDICs, we are considering intensional logic [36], and behavior models [25].

We will continue to explore and compare standard office platforms for alternative CDICs implementations (e.g., LotusNotes and StarOffice). We will use live documents to enhance the documentation features of the Rigi tool. Our ultimate goal is to integrate the Rigi tool so tightly with its documentation that the lines between the tool and its documentation vanish.

8. Acknowledgements

We would like to thank Fang Yang for the CDIC prototype implementation and documentation with COM components. We would also like to thank Jon Pipitone for many productive discussions and his work on the implementation of the SVG components. Many thanks go to Jeff Michaud and Jens Jahnke for proof reading this document. The comments of the reviewers were greatly appreciated.

9. References

[1] Adobe SVG Zone. http://www.adobe.com/svg/

[2] Anklam, P. 1999. Technical communications as knowledge management: Evolution of a profession. In Proceedings of the ACM 17th International Conference on Systems Documentation (SIGDOC '99), pp. 36-44

[3] Ball, T.; Eick, S.G., and Mockus, A. 1997.Web-based analysis of large-scale software systems, In ICSE '97 Workshop on Software Engineering (on) the World Wide Webhttp://www.bell-labs.com/user/audris/papers/websoft/

[4] Brierley, S. 2002. Beyond the buzzword: single sourcing. *intercom*, Vol. 49, No. 1, pp. 15-17

[5] Chikofsky, E. and Cross, J. 1990. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, Vol. 7, No. 1, pp. 13-17

[6] CNN Article 1997. Microsoft rivals say browser war is really about Windows. http://www.cnn.com/TECH/9710/20/microsoft.impact/

[7] Delisle, N.M. and Schwartz, M.D. 1987. Contexts - A Partitioning Concept for Hypertext. *ACM Transactions on Office Information Systems*, Vol. 5, No. 2, pp. 168-186

[8] Eick, S.G. et al. 1997. Web-based text visualization. In *SOFTSTAT '97 Advances in Statistical Software 6*, pp. 3-10, Lucius & Lucius

[9] Eick, S.G.; Graves, T.L.; Karr, A.F., and Mockus, A. 1998. A Web Laboratory for Software Data Analysis. *World Wide Web*, Vol. 1, No. 2, pp. 55-60

[10] Green, R. 1999. Component-Based Software Development: Implications for Documentation. In Proceedings of the ACM 17th International Conference on Systems Documentation (SIGDOC '99), pp. 159-164

[11] Greenberg, S. 2001. Context as a Dynamic Construct. *Huma-Computer Interaction*, Vol. 16, No. (in press)

[12] Hakkarainen, K.A. 1999. Unifying Documentation Teams. In Proceedings of the ACM 17th International Conference on Systems Documentation, pp. 176-184

[13] Horn, R.E. 1998. Structured Writing as a Paradigma. In *Instructional Development: State of the Art,*. Englewood Cliffs, N. J.:, Educational Technology Publications

[14] Hunter Utt, M. and Mathews, R. 1999. Developing a User Information Architecture for Rational's ClearCase Product Family Documentation Set. In Proceedings of the 17th International Conference on Computer Documentation (SIGDOC '99), pp. 86-92

[15] Iivari, J. 1996. Why are CASE Tools not used? *Communications of the ACM*, Vol. 39, No. 10, pp. 94-103

[16] Kemerer, C.F. 1992. How the Learning Curve Affects CASE Tool Adoption. *IEEE Software*, Vol. 9, No. 3, pp. 23-28

[17] Kienle, H. et al. 2002. Leveraging SVG in the Rigi Reverse Engineering Tool. In Proceedings of the SVG Open Developers Conference (on CD Rom and Web), Zurich, Switzerland, July 15-17, 2002

[18] Macromedia Flash. http://www.openswf.org/

[19] Maynard, D. and Ananiadou, S. 2000. Terminological Acquaintance: the importance of contextual information in terminology. In Proceedings of the NLP2000 Workshop on Computational Terminology for Medical and Biological Applications, Patras, Greece

[20] McAllister, G. 1999. The Developer-Documenter Relationship in Java Software Development. In Proceedings of the ACM 17th International Conference on Systems Documentation (SIGDOC '99), pp. 144-146

[21] Priestley, M. 1999. Dynamically Assembled Documentation. In Proceedings of the ACM 17th International Conference on Systems Documentation (SIGDOC '99), pp. 53-57

[22] Flexible information visualization components for authoring WWW Live Documents. http://citeseer.nj.nec.com/mockus99flexible.html

[23] Mockus, A. et al. 2000. A Web-based approach to interactive visualization in context. In Proceedings of the Working Conference on Advanced Visual Interfaces, pp. 181-188

[24] Müller, H.A. and Klashinsky, K. 1988. Rigi-A System for Programming-in-the-large. In Proceedings of the Proceedings of the 10th International Conference on Software Engineering, pp. 80-86

[25] Phelps, T.A. and Wilensky, R. 2000. Multivalent documents. *Communications of the ACM*, Vol. 43, No. 6, pp. 82-90

[26] Powell, A.L. et al. 1996. A systematic approach to creating and maintaining software documentation. In Proceedings of the Symposium on Applied Computing, Philadelphia, Pennsylvania, United States, pp. 201-208

[27] Sobiesiak, R. 1995. Towards a new generation of authoring tools. In Proceedings of the Annual ACM Conference on Systems Documentation (SIGDOC '95), pp. 157-161

[28] Stieren, C. 1997. Add One Egg, A Cup of Milk, and Stir: Single Source Documentation for Today. In Proceedings of the ACM 15th International Conference on Systems Documentation (SIGDOC '97), pp. 255-262

[29] SUN Javadoc. http://java.sun.com/j2se/javadoc/

[30] SUN and SVG. http://www.sun.com/software/xml/developers/svg/

[31] Susan Korgen 1996. Object-Oriented, Single-Source, On-Line Documents that Update Themselves. In Proceedings of the ACM 14th International Conference on Systems Documentation, pp. 229-237

[32] Tallis, M.; Goldmann, N.M., and Balzer, R.M. 2002. The Briefing Associate: Easing Authors into the Sematic Web. *IEEE Intelligent Systems*, Vol. 17, No. 1, pp. 26-32

[33] Tilley, S.R. 1992. Management decision support through reverse engineering technology. In Proceedings of the CASCON '92, November 1992, pp. 319-328

[34] Tilley, S.R. 1993. Documenting-in-the-large vs. Documenting-in-the-small. In Proceedings of the CASCON '93, Toronto, Ontario, Canada, October 25-28, 1993, pp. 1083-1090

[35] W3C Scalable Vector Graphics (SVG). http://www.w3.org/Graphics/SVG/Overview.htm8

[36] Wadge, W.W., Schraefel, M. C. 2001. A Complementary Approach for Adaptive and Adaptable Hypermedia: Intensional Hypertext. In Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia, Third Workshop on Adaptive Hypertext and Hypermedia

[37] Walenstein, A. 2002. Theory-based Cognitive Support Analysis of Software Comprehension Tools. In Proceedings of the 10th International Workshop on Program Comprehension, La Sorbonne, Paris, France, June 26-29, 2002

[38] Wong, K. 1998.The Rigi Manual, http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/rigi-5.4.4-manual.pdf

[39] Wong, K. 1999. The Reverse Engineering Notebook. PhD Thesis, Computer Science Department, University of Victoria, Canada.