Dashboard Software Metric Visualization for SQR

April 2003

Prepared By:John James Willson

Table of Contents

List of Figures

List of tables

EXECUTIVE SUMMARY

This report and related research activity owes its inception to many people. The idea for software metrics in general was sparked by the first Westmost course delivered by Professor Paul Sorenson [Sorenson1993] [Florac1999]. Professor Carl Gutwin gave the idea of using paper prototypes and end user interface testing in his Westmost course [Gutwin1999]. Professor Kal Toth in his course provided scoping and software design [Toth1996]. Dan Thornhill and Darrin Miller provided the industrial gist for the research mill and the prompting for visualization [Brio2002] [Miller2002]. Finally Professor Ken Wong showed immense patience and guidance with the author as he was performing and documenting this research [Wong1996] [Wong1999].

This document covers research into using software metric visualization of SQR source code artifacts: reports, includes and/or programs. SQR is a widely used commercial procedural programming language.

A survey of the current state of source code visualizers was conducted mainly using the Internet. The survey reviewed many current visualizers, and involved downloading and applying them to SQR programs. The survey results are summarized in subsequent sections; however, there were not any direct SQR visualizers. Indeed the time for source code visualization of procedural languages as opposed to object-oriented seems to have passed. No visualizers that dealt with either SQR and/or RPG could be located. It was as though developers of visualizers were not interested in commercial source code other than COBOL or C. It was decided to build a visualizer as opposed to purchasing a visualizer off the shelf. As a result, some preliminary visualization activities were conducted as well as establishing concrete tasks for visualization users. These tasks resulted in high-level requirements for building a visualizer.

SMV (SQR software metric visualizer) was built in prototype form. The first prototype was very primitive, resulting in a limited view of key performance indicators for SQR developers. However, the prototype showed promise and was redeveloped two more times until a suitable working version was obtained. The first prototype fleshed out the high-level requirements and prioritized them into a collection of most useful visualizations.

By applying SMV to a large commercial report suite, insights into SQR coding were obtained. These insights also required further refinement of SMV by adding features such as a legend for colour coding and SQR specific language colour coding. Developer styles were clearly visible, as were software change events that were not immediately obvious from source code inspection. Indeed, an evaluation of

SMV was performed with an initial group of SQR developers whereas software development managers performed evaluation of the SMV visualizations. These evaluations formed the basis for the research conclusions and recommendations.

The main conclusions and recommendations of this research concern the SMV software tool and its products. Software metric visualization should be applied to many commercial language based applications. There is a need for visualization in the management of intermediate to large commercial software development and maintenance activities. SMV should likely be an add-in to development environments and source code management tools. If the activity cannot be measured then the activity cannot be managed. Ease of presentation of software measures is the core function of SMV.

The application of visualization in general, is a large research area. A universally agreed classification of this research is needed. There is a proliferation of approaches and visualizers in this area. The use of a user panel and/or initial investigation group seemed not to be a widely used approach to building visualizers.

1. INTRODUCTION

This document is about software source line visualizers and software metric visualizations of SQR programming. SQR is a widely used report writer language that is database and operating system independent with strong SQL capabilities. The document covers a survey of visualizers and also covers research concerning one developed visualizer for SQR. Source line visualizers are popular where measurements of thousands of lines of source code (KLOCs) are used as key productivity indicators (KPIs) for software developers and management. Although KLOCs have been shown to not be a highly reliable indicator of software productivity, the measurement still remains one of the easiest to obtain for managing software development and maintenance. Problems in KLOC concerning issues of ‘comment’ instructions and general language constructs can be mitigated using a visualizer.

This document is also provided in partial fulfillment of a Masters in Science in software technology from the University of Alberta. As a result the document covers: some objectives of doing research in this area; the systematic methodology followed; what data was gathered; and the development of a visualizer for SQR.

This section continues with the objectives of this research, the scope of the research, and the approach taken. Finally the section concludes with some background on SQR.

1.1 OBJECTIVES

The objectives of this research were as follows:

- to assist large SQR development projects in the construction and maintenance of SQR artifacts;
- to help software managers in assessing productivity of SQR developers and overall project progress;
- to provide software engineers with an estimation tool for SQR artifact upgrades; and,
- to provide software managers with a rapid snap shot of code status over a period of time.

These objectives essentially guided the research that was conducted.

1.2 SCOPE

It became apparent early in the research that visualizers of source code, have a long history in computing and are embedded in much of character based computing. Appendix F lists some of the search terms used to browse the Internet and their related number of hits or citations. Visualizers are now being applied to object-oriented source code. However, object-oriented source code visualization is concerned more with characteristics of class structures than with the make up of an individual line of source code. Line visualizers of source code are explored in detail in this study, since they tend to bridge between character visualization and the more recent object-oriented visualizers. Visualizers that were hyperbolic or used mapping techniques such as 3D, fisheye views or virtual reality were not explored in detail. These visualizers did not lend themselves immediately to a line oriented procedural language such as SQR. Further, although SQR readily produces output in HTML and XML, web-based visualizers were not explored in detail. Finally not explored in detail were visualizers that captured class information and/or object-oriented information. SQR is not an object-oriented language although it can have a limited amount of class functionally. What was explored in detail was the application of line based visualization techniques to SQR source code, and in particular to a suite of SQR programs running as a defined application. Many additional SQR based applications were also researched. A visualizer software prototype was built to perform multiple views that included information about author and language constructs. There did not previously exist a visualizer that directly applied to the language constructs of SQR.

1.3 APPROACH

The approach to conducting the research consisted of the following:

- a current survey of SQR and of visualizers was completed;
- a review of software quality metrics as it is applied to SQR development was completed;
- the building a SQR software metrics visualization tool was started;
- the building of a prototype to measure the applicability of visualizers to SQR software products was done;
- the refinement of the prototype and its application to a known SQR problem domain was performed three times with end user evaluation; and,
- conclusions and recommendations to the further use of the visualization approaches were stated.

In particular, a prototype was built and applied in a limited fashion to a collection of SQRs. This prototype was then shown to an initial group of SQR developers and SQR software development managers. The prototype was then redesigned and applied to a large suite of SQR programs. The prototype was finally redesigned and applied to a 'working' report suite of newly developed SQR programs. An independent initial group viewing the results of the final prototype performed an end-user evaluation of the prototype.

1.4 BACKGROUND

To assist in understanding the need for an SQR visualizer, this section discusses SQR program and language structure. Then a discussion of software source line visualizers in general is given. A brief comment on revision control systems and their relationship to visualizers is made. SQR visualizers and SQR metrics are then discussed.

SQR is widely used in data warehouse applications. The language is also widely used in Peoplesoft, SAP, and Oracle and other enterprise reporting packages (ERPs). Further, the language is also used for report publishing to the Internet. The language is used to complete GAP s (requirements not being met by features of a software system) in ERP implementations and to fit multiple applications together. Typically, development in the language consists of “hierarchical input processing and output” waterfall methodologies. [Willson2001]. SQR has structured programming language capabilities with lines of code being a strong software metric. Code developed with SQR can be maintained and supported using revision control software and standard defect reporting. The source code is usually collected into flat ASCII files on any given operating system. The SQR compiler and run time version are command line driven on all platforms. SQR is also used to call other languages such as C, C++, Visual Basic, Delphi, and others (or be called itself). SQR gets embedded or hidden in many applications because of its strong reporting capability that provides reports in most formats including Post Script, PDF, HTML, XML, line printer, and HP. Although there is a graphical user interface to recent Brio versions of SQR [Brio2002],

SQR can be command line executed.

1.3.1 SQL to SQR

```
Begin-Program
Begin-Select
PRODUCT_LINE,
PRODUCT_NAME,
STORE_NAME,
AMOUNT_SALES,
                    Print  &product_line    (+1,1)
                    Print  &product_name    (0,40)
                    Print  &store_name      (0,+2,30)
                    Print  &amount_sales    (0,+5)        edit $$$,$$$,$$$.99
From        STORES
            SALES_FACT
            PRODUCTS
Where       STORES.STORE_ID = SALES_FACT.STORE_ID
And         SALES_FACT.PRODUCT_ID = PRODUCTS.PRODUCT_ID
Order By    PRODUCT_LINE, PRODUCT_NAME, STORE_NAME
End-Select
End-Program
```

Figure 1 SQL to SQR Conversion or What's SQL and What's SQR

Figure 1 is from Nick Moscaritolo a Senior Systems Consultant of Brio Software and his presentation at [Brio2002]. By inspection of the SQR program above it is very easy to see SQR's roots. Indeed, in most cases SQR passes SQL directly through to the underlying database without modification. So a very simple SQR program can have very few changes to that of SQL, but with left-hand column alignment being significant to the compiler. This holds true for whatever underlying database the SQR and/or SQL is acting upon. In figure 1, note the use of 'Begin' before the words 'Program' and 'Select'. And notice the use of the word 'End' before 'Select' and 'Program'. These words bracket and modularize SQR. Additional 'Begin/End' combinations for 'Procedure' and other constructs are available. In the following figure, figure 2, the SQR language flow is briefly described.

1.3.2 SQR Program Structure

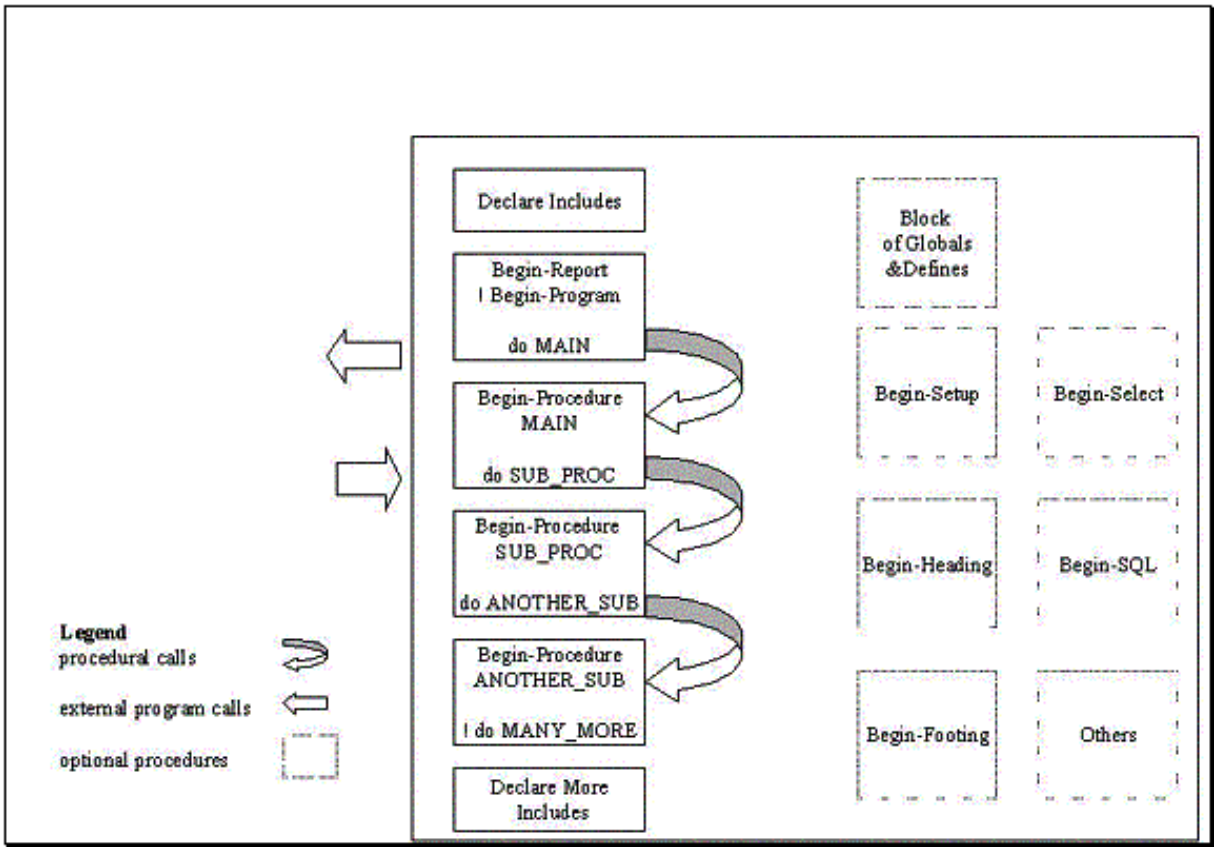

Figure 2 SQR Language Program Flow

In Figure 2, the flow of an SQR program is presented. It is important to note that sub routines are called procedures and that a parameterized procedure creates local variables to that procedure. All other variables in SQR are global. Compiler instructions include 'includes' and 'defines'. SQR also has reserved words and directives for using a global reserved word within a local procedure. A common problem of using 'includes' is creating variables that are unlikely to be named by a developer in order to prevent variable collision. Modularization is completed using key word Begin/End pairing.

1.3.3 SQR Language Structure

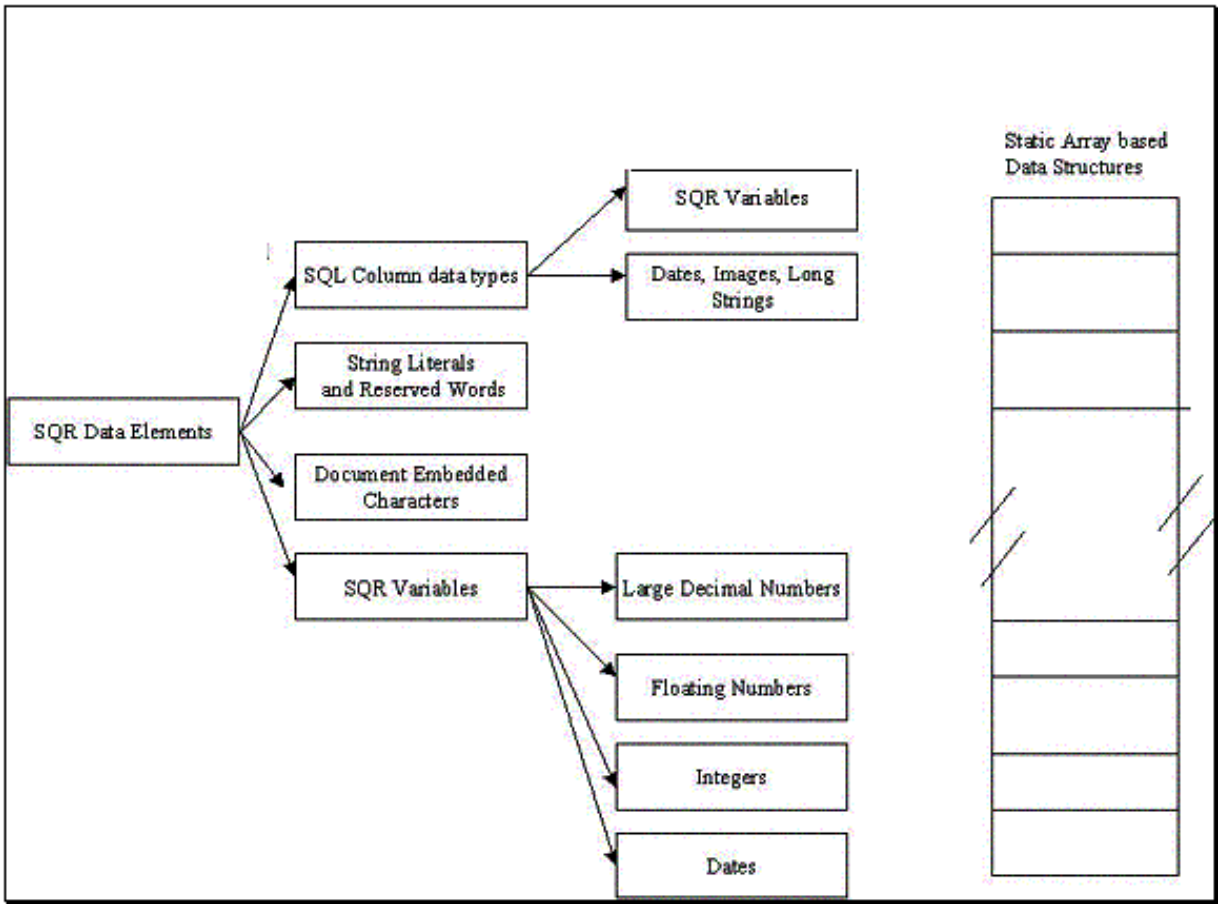

Figure 3 SQR Language Data Structures

Figure 3 shows the data structures of earlier versions of SQR. An important missing data structure is the class. However, pseudo class structures may be obtained through using 'includes' as the example program below shows. In figure 4 following, an example of an abstract data type of a stack is shown. This software and other software such as standard deviation are constructs that extend SQR as a programming language and can be obtained from http://www.dssltd.com/. Extension of the language is possible through user-defined functions and calls. The language itself is written in C; however, direct access to the C pointers is not available to the SQR developer.

1.3.4 Object Based SQR Program

SQR PROGRAM

```
#include 'stack.inc'

begin-setup
  declare-variable
    integer #number #remainder
  end-declare
end-setup

begin-report
  do get_number_base10
  do create_stack
  do print_base2
end-report

begin-procedure get_number_base10
  show  'Base 10 to base 2 Number Converter'
  input $numbase10 'Enter base 10 number
 (max 4 digits) ' type=integer maxlen=4
end-procedure

begin-procedure print_base2
  show ' Number in Base 10 = ' $numbase10
  move $numbase10 to #number
  while #number <> 0
    let #remainder = mod(#number, 2)
    do stack_push(#remainder,#depth)
    let #number = trunc(#number/2,0)
  end-while
  display  ' Number in Base  2 = ' noline
  do stack_empty(#depth, #stack_empty)
  while not #stack_empty = 1
    do stack_pop(#remainder, #depth)
    display #remainder noline
    do stack_empty(#depth, #stack_empty)
  end-while
end-procedure
```

STACK INCLUDE

```
begin-procedure  create_stack
#define MAX_ROWS 1000
create-array name=stack size=(MAX_ROWS)
          field=j:number
! #depth - Stack depth
  let #depth = 1
end-procedure

begin-procedure  stack_empty(#depth, :#stack_empty)
  if #depth = 0
    let #stack_empty = 1
  else
    let #stack_empty = 0
  end-if
end-procedure

begin-procedure  stack_push (#push, :#depth)
    let stack.j(#depth) = #push
    let #depth = #depth + 1
end-procedure

begin-procedure  stack_pop (:#pop, :#depth)
     let #pop = stack.j(#depth)
     let #depth = #depth - 1

end-procedure
```

Figure 4 SQR with object based modularity

In figure 4 above a limited approach to object oriented programming is shown in a program that converts base 10 integers to base 2 integers. Note that in the 'include' on the right-hand side, there is a 'define' of maximum rows of 1000. In early versions of SQR memory was reserved through block assignment. So even though SQR was under the wraps C language code, memory reservation for array sizes were fixed and not dynamic. (No pointer accesses in SQR). Again note the blocking structure in the above SQR. In the following figure, figure 5, classic measurements of developer productivity in coding or maintaining SQR programs is presented.

1.3.5 Classic SQR Metrics

For a given SQR program	
	Number of runs
	Number of abends
	Number of defects

	Number of support calls
For a collection of SQR programs functioning as an application suite	
	Number of lines of code
	Number of lines of comments in the lines of code
	Number of individual SQR programs
	Number of authors of the code
	Number of revisions
	Number of 'includes' used
	Number of dependencies or external program calls
	Maximum depth of dependency call tree
For a collection of application suites functioning as a system	
	Number of applications
	Number of revision lines
	Number of lines of comments

Figure 5 Classic SQR Metrics

Classic SQR metrics include counts of the number of lines of code (KLOCs), SQR abends, and defects. Standard software engineering measurements and related statistical analysis may be performed on these measurements.

After selecting an application in figure 5, the following classic measurements are provided for that application:

- the total lines of code in all SQR artifacts in the application including the lines of code that are contained in 'includes' and those lines that may be comments;
- the number of SQR programs in the application and the number of includes;
- the number of authors with code in the application and the total number of revisions of lines of code; and,
- the number of dependencies within the application consisting of the calls of SQR programs to includes and includes of includes and the maximum depth of the longest path. Recursion is not permitted in SQR and is usually trapped at the compiler level.

When metrics are gathered by connection to a production data base, additional statistics are provided such as number of abends of an SQR, the number of times the SQR was run, the number of reported defects of the SQR, and total number of support calls concerning a given SQR

In figure 5 additional classic measurements are shown including system statistics concerning multiple applications. Finally an ad hoc SQL report writer is provided to capture other counts that may be of interest.

Some not-so-classic metrics missing from figure 5 include:

- Measurements of programming style of a given developer including language construct (say use of the *move* command rather than the *let* command);
- Measurements of productivity including lines of code in production produced per time interval and/or number of other developers using an individual developer includes; and,
- Counts of component slices of a given SQR for possible re-engineering of a SQR application suite.

Work breakdown structures on a project that is mainly SQR development have task assignments. These task assignments usually are 'include' based and/or individual SQR based. There is some application or vertical silo based tasking but this may be unusual.

2.0 CURRENT STATE SURVEY

In this section, a discussion of the methods applied to surveying the state of SQR and visualizers is presented. The results of the survey are presented in summary form. Finally the recommendation to go ahead with an SQR visualizer is made.

2.1 METHOD

The methodology followed included the following steps.

- An environmental web scan using Google, Surf Saver and Bulls Eye was performed to look for visualizers.
- A review of web based user groups and key contacts including SQR-users and PeopleSoft fans (rm-users@sqrug.org and/or sqr-users@sqrug.org) was completed.
- Attendance at the international Brio Software 2002 conference including the SQR developer forum and meeting of several Brios' personal was done.
- A review of selected publications of ACM and IEEE was completed.

- Contacting of selected vendors was completed.

Not all of the above steps were completed in performing the current state survey.

2.1.1 Data Gathering

Data for the research was gathered in many ways. This included personal interviews, questionnaires, inspection of other visualizers and metric capturing software, and regular communications.

- In particular, Darrin Miller [Miller2002] provided his SQR software metrics used for managing SQR report suite development. Darrin Miller also provided SQR defect tracking software. This software was written in SQR itself.
- Selected software metric visualizers such as Brio Metric 7 were acquired and applied to SQR visualization.
- Finally the relationship to revision control systems and software control systems was explored. None of the standard revision control systems were using visualization at this time (PVCS, Rational, SourceSafe, MKS, …)

2.1.2 Web Scan

In performing the environmental survey scan, the web was used extensively. Standard search engines were used along with some specialized search tools for finding visualizers.

- For a list of standard search engines used, see the Appendix.
- Additional search engine approaches included:
  - SurfSave which allows the saving of a complete web site;
  - BullsEye which posts to multiple search engines concurrently and ranks the retrieved information; and
  - Internet Cartographer which clusters web sites around statistics of interest such as key words.

Search terms included one or more combinations of the following terms:

- visualize
- SQR
- SQL
- software metrics

- software key performance indicators
- lexical parsers
- lexical analyzers
- visualizers

Although this is not an exhaustive list of search terms, it resulted in a clustering of concepts concerning SQR visualizers. This clustering defined some of the concepts used in the prototypes.

2.2 SURVEY RESULTS SUMMARY

Through the data gathering and web scanning considerable information was gathered concerning software metric visualization and SQR. This information and some speculation are provided in the following paragraphs. These comments are meant to show the range of visualization possible and the myriad of approaches that could be used in the development of a prototype.

2.2.1 Software Source Line Visualizers

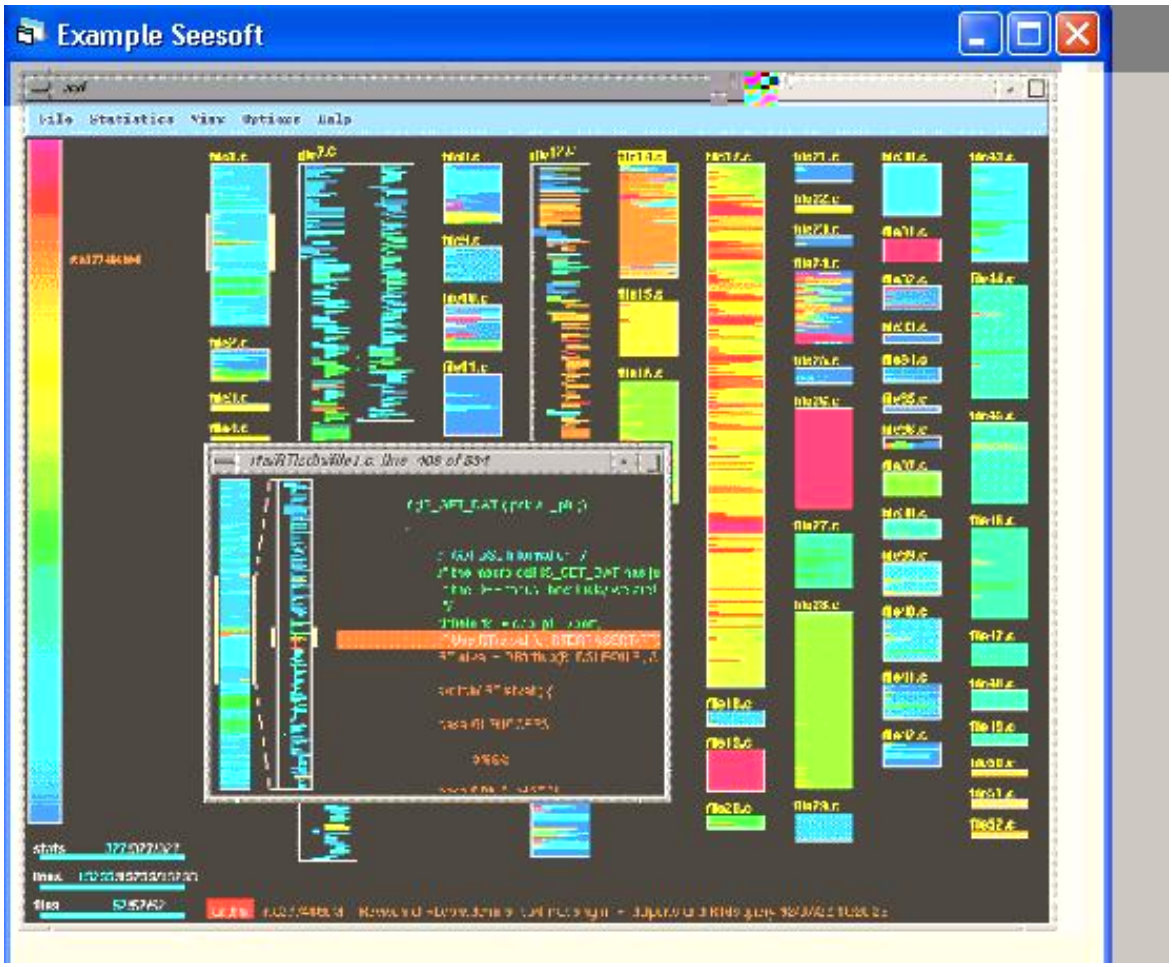

Figure 6 Seesoft [Eick 1992]

Software source line visualizers have a decade long history of exploration starting mainly with [Eick1992]. In Seesoft one sees a visualizer applied to C source code for a large development and maintenance project. The capturing of source lines of code is at the pixel level with one character of a source line’s attributes being mapped to one pixel on a screen’s attributes. This results in the retention of some characteristics of source code mapping directly to a view of the code and the removal of other characteristics. As a result a granularity of information about source lines of code is obtained. To span multiple levels of granularity Seesoft provides both a visual view and a drill down using a popup window that shows the original source code. [Knight2000] in her *System and Software Visualizations* provides many examples of visualizers that have been developed since Seesoft. [Eick1996] provides examples of the application of the visualization technique to other kinds of system development deliverables including data bases, operating systems, and data structures. Current visualization approaches such as [Tao1999], [Qsm2002], and [Swanson2001] provide visualization of object-oriented languages such as Java and C++. C++ has a history of being studied in detail including [Kuh1994]. Many visualizers now exist in both the ‘shareware’ and commercial domains. Current visualizers are now concentrating on the visualization of language structures of object-oriented languages. In [Emden2002] *jCosmo – Java Code Smell Browser Tool* is presented that looks not only for language structures but also at developer approaches to using those constructs.

2.2.2 Revision Control Systems

One of the curious non-events in commercial revision control systems is the lack of adoption of visualizers and visualization techniques for code control and management. Even more curious is that software quality and metrics seem to go together and revision control systems track many of the metrics necessary to increase software quality. [Booch2002] discusses software quality and the Unified Modeling Language but the Rational Software Corporation has not offered a visualizer for metrics as of November 2002.

2.2.3 SQR Visualizers

Although there are many visualizers that cover most of the standard and legacy languages including: C, C++, Java, Lisp, FORTRAN, COBOL, etc, some languages have not had visualizers applied to them. In particular, there are no visualizers for many of the widely used commercial languages such as RPG and SQR. These two legacy languages are still in wide use and still growing in the number of advocates and uses. It is much like the growth of radio after the widespread use of television. Radio continued to proliferate; however, at a reduced rate compared to the growth of television. So many legacy languages continue to grow but not at the rate of the object-oriented or class languages. SQR is just such a language. It continues to grow from its initial codification in [Burton1994] through [Mellen1998] and [Landres1999] to [Miller2002].

There are a limited number of software metrics that get applied to SQR development. These metrics are not visual and usually software managers reviewing development activities rely on table driven data. In large SQR development efforts there is little overview effort assessment other than the number of modules completed or the number of reports completed. See [Willson2001] for SQR development approaches and see [Dask1992] for software metrics in a major client of this document’s research. Visualizing team member effort across modules or programs is not possible. Visualizing quality and amount of code developed is not presently available. In effect the measure of SQR software processes has been restricted to table driven data and that data’s interpretation.

2.2.4 SQR Metrics

Software metrics for SQR deal with the measurement of the software products produced and with the process by which these products were developed. The visualizations of these measurements would deal with the representation of these metrics in a graphical form. [Dumke1999] provides an overview of metric tools. [Wong1996] provides insight into how language constructs in program understanding

could be used as a technique for visualization. Much of the initial research in SQR was acquired while doing computer consulting [Willson1998].

2.2.5 History of SQL Analyzers, Lexical Analyzers, Parsers, and Visualizers

Although there are many citations of SQL analyzers, lexical analyzers, and parsers, as shown in Appendix F, there are not a large number of visualizers that complement the analyzers and parsers. Considerable efforts have been made at a meta-level of visualization. In [Sgi2002] SGI provides Mindset that performs visualization for data mining. This visualization is very broad and complete, and runs on a strong graphics display. Competing products include [Genvis2002], [Dbminer2002], and Ghost Miner. Mention should also be made of KLOCWORK at http://www.klocwork.com; and, the product suite here covers many different kinds of insight into general software metrics. All these visualizers work at the SQL data base level and could have applicability to SQR visualization. However, these visualizers would have to be tuned for SQR language visualization as they are currently tuned to treating pure data. They would also have to be tuned to individual author issues.

2.2.6 SQR IDEs

There are many SQR integrated development environments that could provide SQR visualization. However at this time the following IDEs are concerned only with the character representation of SQR. Hence they leave the visualization to revision control systems and as we have seen current revision control systems do not perform visualizations. Also at this time, ERPs are not performing visualizations either.

See in particular:

- SQR/ Brio Report Builder http://www.brio.com/
- SQR Runner http://www.sqr-runner.de/
- SQR Plus http://www.sqrtools.com/
- SQR Workbench http://www.nau.edu/

2.2.7 Selected Visualizers

The current closest visualizer for SQR directly is SQR Tree (http://www.kagi.com/) written by Wayne Ivory. However, SQR Tree only provides a call or dependency tree of procedures. It is a utility that shows a tree structure of the procedural and 'include' files of a given SQR program.

2.3 SMV REQUIREMENTS

As a result of the current state survey, it is apparent that there is not an SQR metric visualizer that is readily available. It is also apparent that visualization could be useful to this programming language as it has been to other languages. So what kind of visualizer is required? Before specifying an SQR visualizer, consider the level of granularity that a visualizer should work at. Programmers tend to work at the code level of granularity. So how about using visualization on a single SQR program?

2.3.1 Character Views versus Visualization

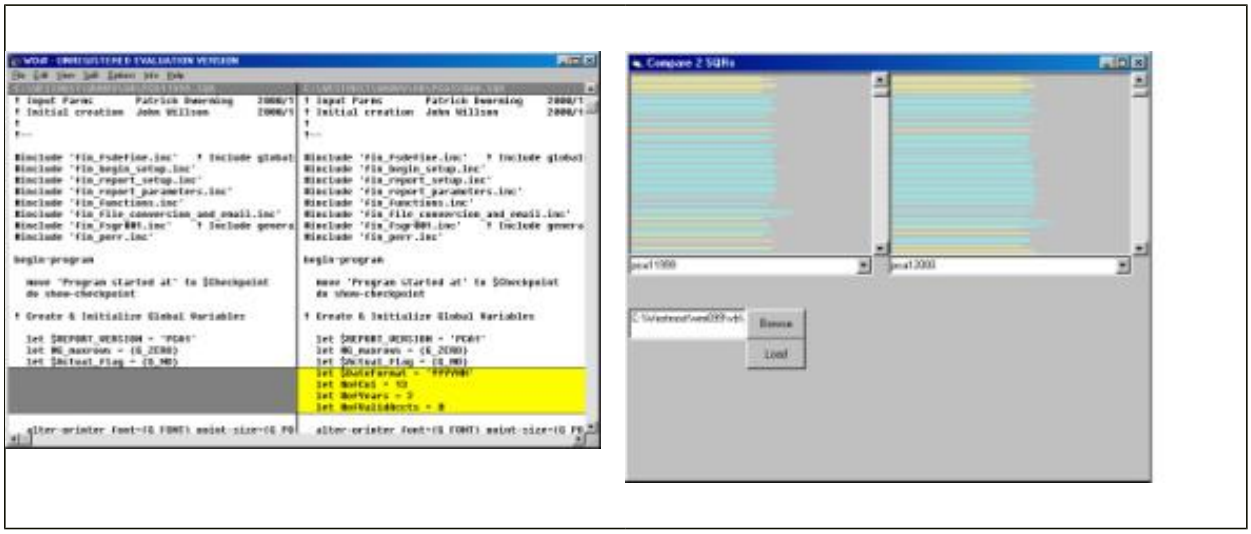

Figure 7 Character views versus Pixel views

In Figure 7, a character view and a visualization view of the same two SQR programs is presented. This is an example of one kind of single visualizer. From inspection, the character view is the better heuristic when looking for change events between the two SQR programs. Programmers typically look for change events from one program, one system, one version or package to another. Comparing two SQR programs in some form of graphical approach is better done with character display comparisons.

The use of SQR visualization then becomes a matter of what level of granularity is required. For an SQR developer, character based visualization may be all that is required. For a team leader and/or software development manager a higher level of visualization may be required. As a result the next section deals with SQR software metric visualization research. However, before describing the research, the following possible tasks of an SQR visualizer and their prioritized requirements are listed from discussions with SQR community.

2.3.2 Concrete Task Examples of Visualizer Users

The following task examples are gathered from interviews concerning usage of an SQR visualizer. For a copy of the interview questionnaire please see the appendix.

2.3.2.1 Task Example 1:

An SQR developer has written 100 KLOCs in 12 SQRs for a large hospital over 2 years. He has written these SQRs using 'includes' that he personally developed and/or cloned from experience with multiple ERPs. He has not worked on the hospital system for over sixteen months when he is called to come back and make modifications to his production SQRs.

2.3.2.2 Task Example 2:

A project lead for a large computer vendor maintains SQRs remotely throughout North America. His background includes being an Oracle DBA and a DB2 DBA for over 10 years. He typically has from 3 to 10 SQR developers doing maintenance activities for him. He does not necessarily know the state of the SQRs when his organization sells the maintenance service. One annual routine activity of his SQR crew is to upgrade his maintenance customers to the latest version of their database, their ERP, and SQR.

2.3.2.3 Task Example 3:

A delivery manager for a SQR software boutique converts customers to SQR that have reports written in Crystal, Envision, SQL Plus, PL/SQL, and other database query languages. Typically the delivery manager gets called in to do the conversions when the other report writers do not scale to enterprise level reports and/or do not scale to the Web. Sometimes the scalability is not just a performance issue of end-user response times but one of multi-language requirements or cultural needs.

2.3.2.4 Task Example 4:

A freelance SQR consultant gets called in to make an SQR product suite more efficient. The report suite delivers its production reports but the organization has grown rapidly and the reports are becoming less timely. Newly trained SQR developers who did not take advantage of the many capabilities of SQR wrote the original reports. In particular, the use of in memory arrays, 'load-lookup tables', as opposed to database retrieval is not done.

2.3.2.5 Task Example 5:

A delivery manager for Brio professional services gets tasked with rapidly creating financial reports for a data warehouse that is being fed data from a newly installed ERP system with limited web reporting capability. The timeframe for implementation is 4 months and the number of SQR developers is 44 in 3 global locations. A report suite approach is required using a systematic methodology; however, enforcing of SQR development standards would elongate the 4 months so individual developers adopt their own programming style to complete their individual SQR assignments.

2.3.3 Tentative List of Requirements

The previous task examples resulted in the creation of the following table. The table is prioritized. The ranking scale is the author's. (Note that the column entitled *scale* has the following legend: a - must, b - should, c - could, d - exclude. E.g. the column is then read as 'the requirement must be met by the visualizer'.) As a result of discussions with the initial end user group, the requirements were separated into a prioritized list. This list became the objectives for the SQR software metric visualizer.

Table 1 Visualizer Requirements

#	Requirement	Scale	Rationale
a1	Capture 'raw' statistics of code productivity	a	Core functionality of a visualizer. Presentation of these statistics is not necessary if they can be viewed appropriately.
a2	Drill down on visualized abnormalities	a	Provides a way of viewing the source code to inspect whether an abnormality is real or imagined
a3	Selectable targets of visualization including author, variables, language constructs, words	a	Provides a way of selectively visualizing.

a4	View change events consisting of change of author, code modification, other	a	Core operation of a visualizer. It major change events can not be viewed then you do not have a visualizer.
b1	Capture significant views for presentation and discussion	b	Provides a way of using visualization for human resource issues and/or maintenance issues.
b2	Easily retrieve SQRs for visualization	b	Provides a way of importing SQRs for visualization
b3	Highlight abnormal change events	b	Provides an automated way of viewing change events
b4	Interface to version control software such as PVCS, MKS, Rational, other	b	Provides an easy way of capturing change events of interest.
b5	Provide ad hoc reporting of statistics not just pre canned statistics	b	Provides a way of retrieving statistics of interest by passing out-of-the-box visualizations. Allows for more custom visualizations.
b6	Provide feedback on the visualizer operation	b	Assists a visualization user to make appropriate use of the visualizer.
b7	Provide rapid learning of visualizations including legend and titling	b	Assists in moving up the learning curve faster.
b8	Provide snap shot capability for before and after timed views	b	Provides a way of measuring changes in terms of percentages at a macroscopic level.
b9	Set colour codes for author, for language construct, for variable	b	Provides a way of distinguishing statistics of interest in a visual form.
b10	View a whole SQR application not just a program	b	Provides a way of managing application report suites
b11	View programmer language style	b	Provides a way of viewing particular developers way of using the language.
c1	Export visualizations to other office productivity tools such as Visio, Word, other	c	Provides a way of enhancing the visualization.
c2	Interface to developer tools such as TextPad, Brio Report Builder, other	c	Provides an easy way of making changes after viewing a change event.
c3	Provide thin client for visualization to the web	c	Provides a way of managing SQR development in multiple geographic locations.
c4	Set level of view such as an elevation ascend/descend	c	Provides for scalability of view. Results in settable levels of granularity.

c5	View consistency of programmer style	c	Provides a way of viewing language style independent of kind of language construct.
c6	View history of visualizations including archiving of significant visualizations	c	Provides a way of making routine a series of visualizations say from year to year.
d1	Convert visualizations into statistics or statistics into visualizations	d	Provides another media for understanding underlining change events.

The above table of requirements was high level and was refined through out the development of the prototypes of the visualizer. However, the table did provide a starting point for development and a measure as to progress of the development.

3.0 SQR SOFTWARE METRIC VISUALIZATION RESEARCH

This section deals with SQR software metric visualization research. In particular the methods that were followed is discussed. Also discussed is how SQR development and maintenance could be improved using visualization. This section also covers the first prototype of an SQR visualizer and describes it as a software tool. Finally the section discusses recommendations for SQR visualizer use and the first redesign of the visualizer.

3.1 METHOD

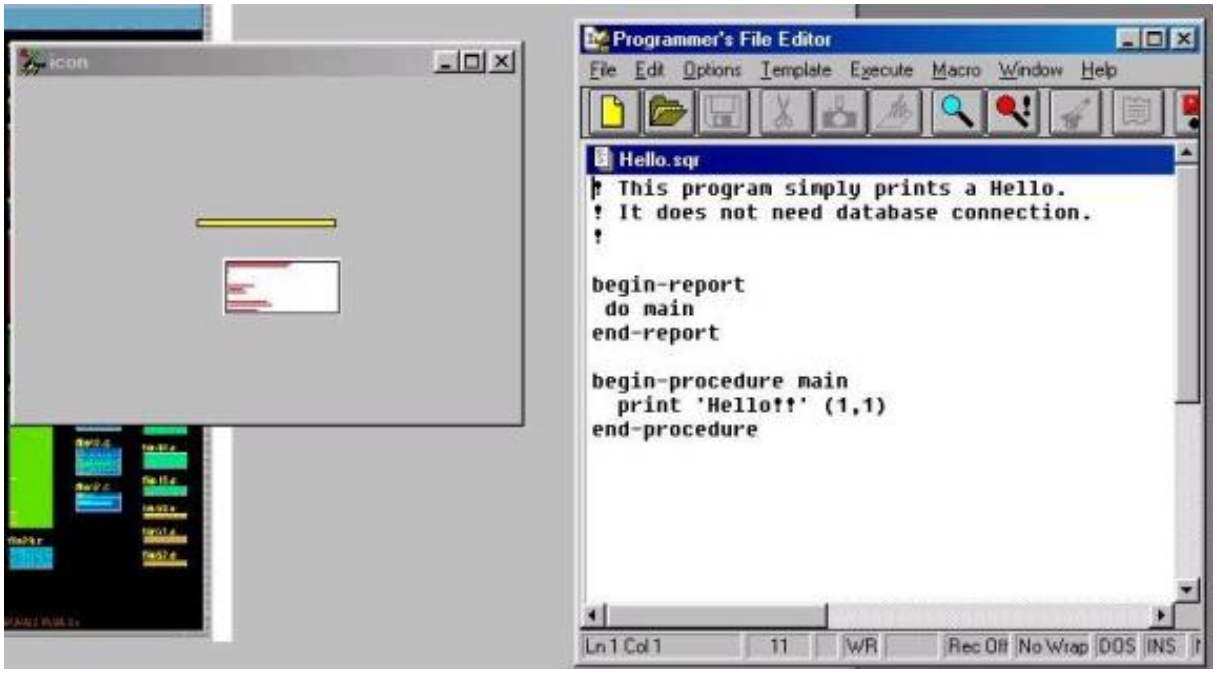

Figure 8 SQR 'Hello World' Character and Pixels

Figure 8 above shows a character example of an SQR program entitled 'Hello World' and a visualization of 'Hello World' in pixel translation. The theory behind an SQR visualizer is shown in this figure. Take a given line of SQR code and convert the line into pixels. The characteristics of the line that are of interest are mapped to the characteristics of a screen's pixel. This enables up to 50 KLOCs to be visualized on a single screen. SQR programs range in line count from very small such as 'Hello World' to 10 KLOC as per a very large program. The colour attribute of the pixel is assigned depending upon the statistic of interest such as latest row changes, age of a row, author of a row, type of code, dependency on other rows, language construct, etc. So the same information graphically presented allows compression of information on the screen. This compression can be in many forms. The compression shown is one pixel for one character of information. Both are presented on single line. However, twips in the Windows world would allow up to 20 characters per pixel. So many different variations on the character/pixel relationship are possible. In research a variety of these variations were tried with varying success. The line representation seemed the most intuitive; however, other representations were possible and tried. In particular, the hyperbolic and fish eye representations were coded to the first prototype stage; but they did not seem to have the insights that a line representation held as the transition from the code to the view seemed to require more 'thinking' on the part of the end user.

3.2 PROCESS IMPROVEMENT

As a result of viewing lines of pixels in place of lines of code certain patterns are discernable. In particular, where SQR program size is large or when many programs have to be viewed concurrently in order to track say a variable across an application suite, visualization is a good approach. It is not better than say using 'grep' in the Unix world or 'find' in the Windows world; however, it is better when viewing a certain author' s code contribution or latest changes. So the process improvement is possible in many ways. These ways include development across multiple developers of large program suites and/or the maintenance of SQR in large program suites.

3.3 FIRST PROTOTYPE

Figure 8 above is an example from the first prototype of the SQR software metric visualizer (SMV). In figure 9 below one of its first applications is shown.

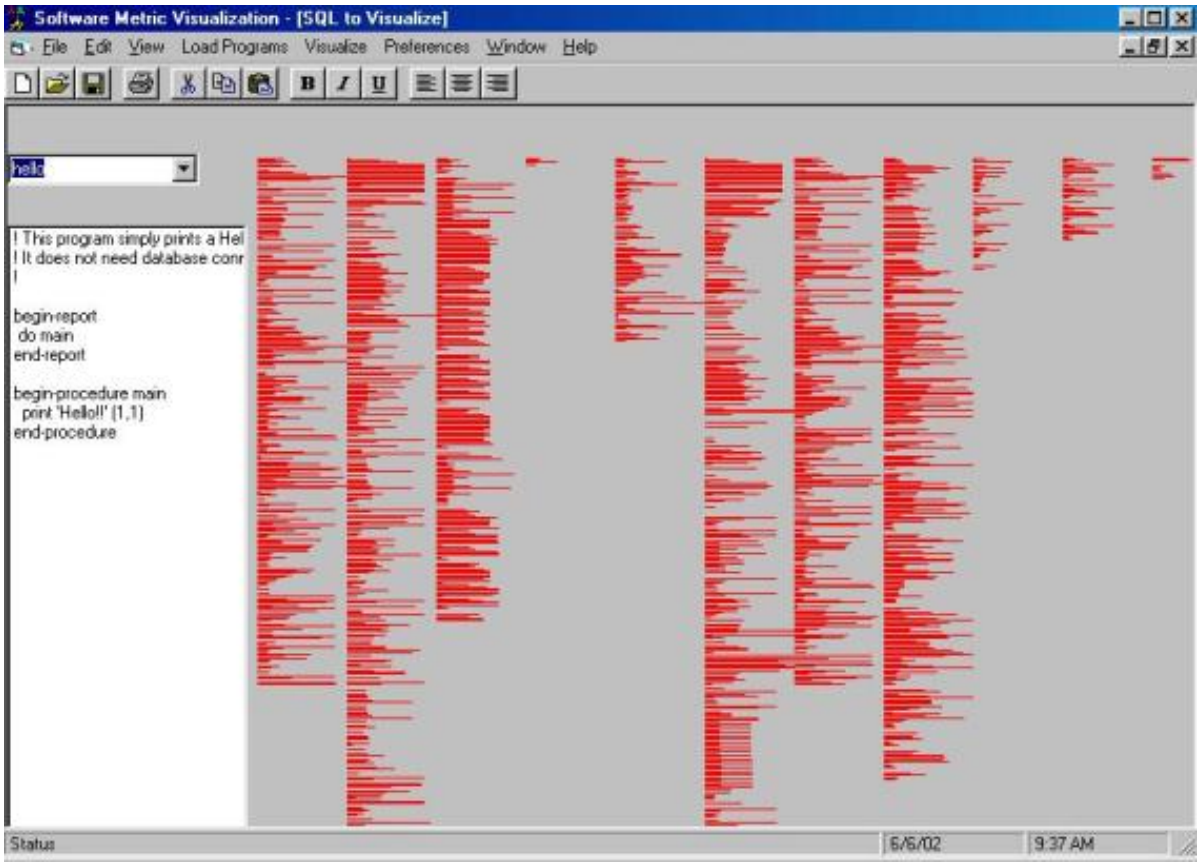

Figure 9 Multiple SQRs visualized using First Prototype. 'Hello World' in list box.

Visualizing without colour coding for either author origination and/or language line construct makes for lack of clarity. Note however, that some language constructs remain visible in the above figure and that there is indentation like the original code. So one starts looking for patterns of styles. It became very clear in the first prototype that a pop-up or list box of code being visualized would be a necessity. As a result SMV allows clicking on a pixel in order to show the line under inspection in context of the program listed in the list box. The first prototype had several shortcomings as is shown in the following figure 10.

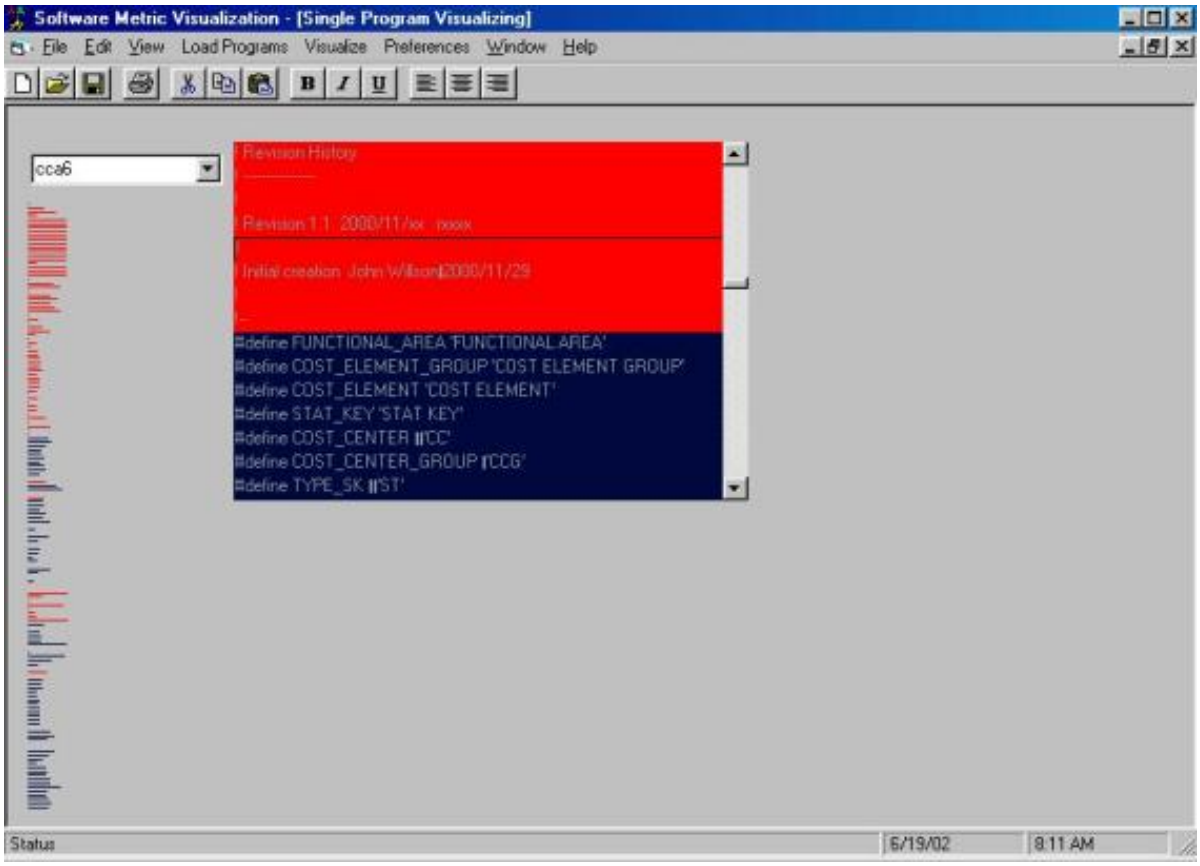

Figure 10 Single SQR view with language line colouring and emphasis on full source code display.

In figure 10 a single SQR is visualized with language line colouring that matches the colour of the source code shown again in a list box. This language colouring enables some code inspection. For example, lines that are 'comments' in SQR could be ignored or filtered out depending upon their colour. Indeed, in the following figure, figure 11, a collection of filters and options are shown. These options increased as the prototypes were refined and became an interesting part of the research.

3.4 SOFTWARE TOOLS

3.4.1 Display options

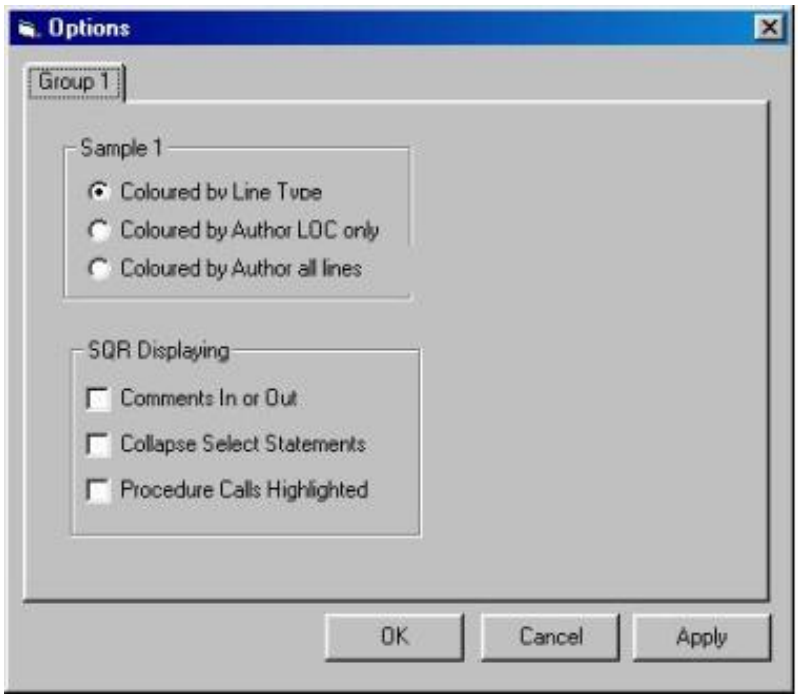

Figure 11 Initial SQR visualizing options.

Figure 11 shows many of the initial options of the SMV. Line colouring by type of SQR line is settable. Also settable are individual author's colours. This allows for the tracking of language constructs across an application suite and the contribution of an individual author to an application suite. Variations of these colours occurred in later prototypes including using the first 5 pixels of a line for author colouring independent of the colouring for the line's type. In addition filters were allowed. These filters included the removal of 'comments' for both visualization and classic statistical counts. The 'select' statement which is a major component of most SQR programs was also allowed to be filtered in or out. The 'select' statement was still identified; however, only the 'begin/end' were shown in pixel form allowing the visualization of language constructs independent of the underlying SQL used. Finally 'procedural' calls and 'includes' could be highlighted and shown as useful.

3.4.2 Traditional Software Metrics Screen

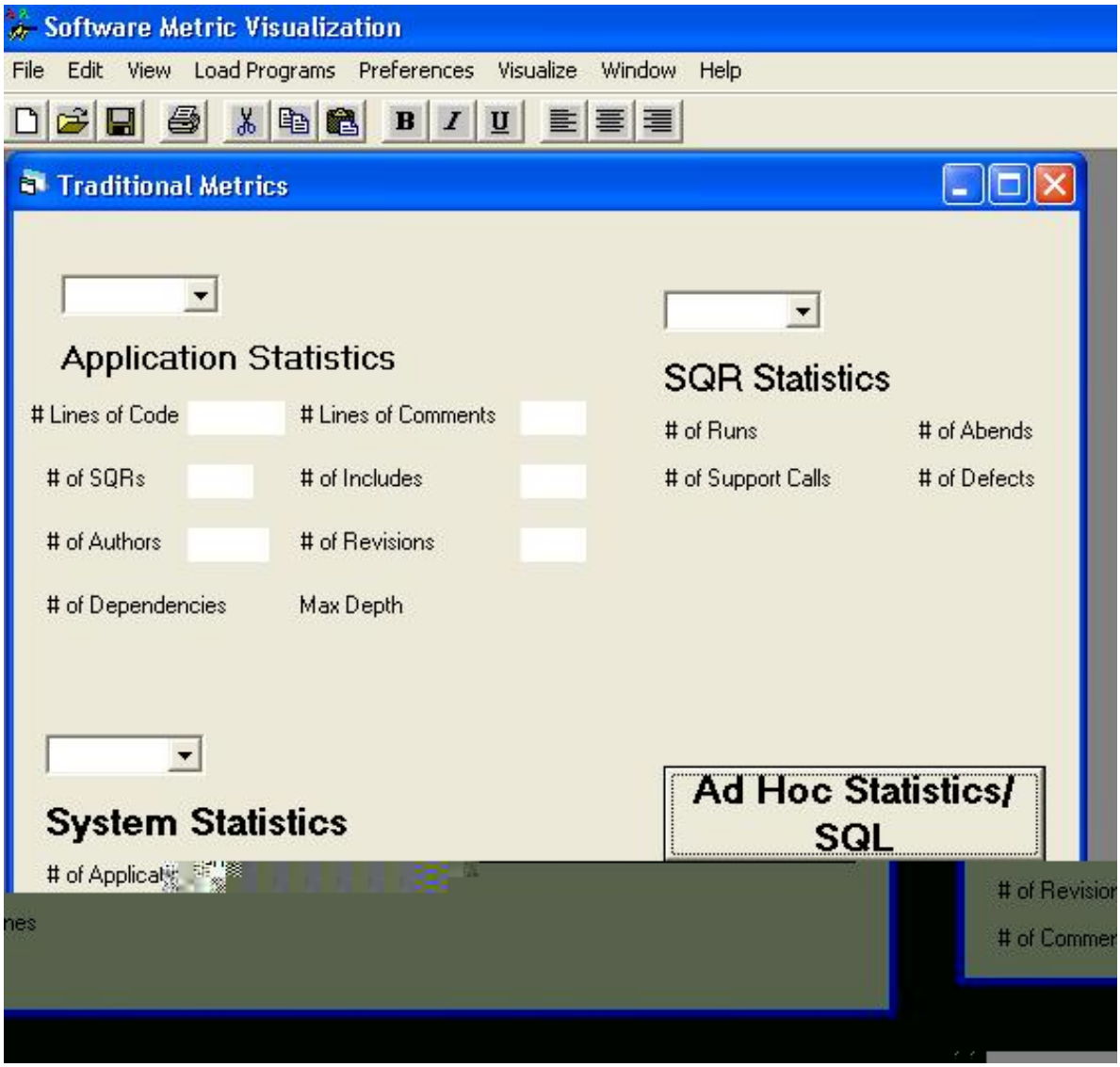

Figure 12 Traditional Software Metrics Screen

Figure 12 above shows the translation of traditional software metrics into the first prototypes statistical screen. Note that an experienced SQR user would know SQL quite well and as a result would be able to perform their own ad hoc statistical queries on the underlying SQR code.

3.5 RECOMMENDATIONS FOR APPLICATION

The first prototype showed promise for visualizing SQR software metrics. However, the prototype required much refinement and redesign.

3.5.1 First Redesign

The first redesign resulted in the specification of the kinds of visualizations possible. Although the use of character/pixel conversion with appropriate colouring resulted in a useful visualizer, it was very painful to remember the number of settings to achieve a particular visualization. These settings required constant tinkering and the understanding of the code of an SQR application suite was being replaced with the remembering of the tinkering of the visualizer. As a result, a selection of visualizations was fixed and the backend of the visualizer went from reading flat files to putting the flat files into a relational database. The use of a relational database was a major design decision; however, considering that SQR usually performs against such a database it makes sense to use the database available. (Note 'star' database structures are also becoming common with the magnitude of the amount of data retrieved in large systems becoming substantial. Indeed in the case of UDB - IBM's enhanced DB2, the recommended report writer is SQR)

3.5.2 Selected Fixed Visualizations

The selected fixed visualizations were:

- Classic measurements. These measurements became SQL counts on the lines of code in a given system, in a given application suite, and in a given program. In addition, ad hoc SQL access was provided for those who wanted additional unspecified counts. Thus meeting requirements a1 and b5.
- Multi-program visualizations. These visualizations became the basis for comparing a complete application suite and represented the silos (or column like listing) of programs that were shown in the Seesoft image. The multi-program visualizations allowed the tracking of common language constructs, authors, and statistics of interest viewed across the application suite. Thus meeting requirements a3, a4, and b1.
- Single program drill-downs. This visualization allowed drilling down into a given program fundamentals. This was the lowest level of granularity of the SMV but still provided interesting insights into the large application suite subsequently viewed. Thus meeting requirements a2 and b2.
- Application sizing inspection. Although this visualization was the simplest in terms of view since it could have been done with any manner of office productivity suite software such as Excel or Lotus 123, this view provided an overview for subsequent drill down or visualization and became very popular with the user testers. Thus meeting requirements b10 and b3.
- Application dependencies. This view was of the procedural calls of the application suite and in particular the calls to other 'includes'. This was a recursive view of the application with many 'includes' including other 'includes'. Thus meeting b1 and b8 requirements.

- Language dependencies. This became the classic view of the first prototype and all subsequent prototypes used this view as a basis. It is one of the reasons why SMV had to be built as opposed to buy. Many commercial off-the-shelf visualizers could handle developer languages but had not developed filters for commercial languages such as SQR and/or RPG. The language filtering became the first visualization that new SMV users looked for. Thus meeting requirements a3, a4, b7, b8, b9, and b11.
- Author dependencies. Since software metrics was one of the driving forces for doing the research, author dependency views were critical to filter out the 'ectoplasm' white noise of individual author contribution to a large program suite. Thus meeting a4 and b9 requirements.

There were other issues in the redesign of SMV; however, these visualizations became the driving force for the next prototypes.

4.0 APPLICATION OF PROCESS AND TOOLS

This section describes the application of SMV to a large SQR application suite. The SQRs used in the examples in this section are all in production and are now in maintenance mode. They total over 100 KLOCs developed by over 22 programmers over 6 months. These SQRs are based upon the report suite 'includes' of Darrin Miller [Miller2002] and the author. These SQRs were built from September 2000 to January 2001 and are a subset of *Task Example 5*.

4.1 METHOD

All prototypes were built in Visual Basic with ODBC to a SQL database for source line retrieval. SMV will likely be rewritten in Java with JDBC or rewritten in Visual Basic .Net. The rewriting will allow the SMV to become a plug-in to other tools. However, this rewrite has some problems associated with it. Namely, the display of information in the Windows environment is at the twips level of granularity (20 twips to the pixel) whereas most Java only addresses to the pixel level. As a result the rewrite will likely have to be in vector line graphics bypassing the character to pixel conversion and going directly to the character to line conversion.

The second and third prototype outputs are presented in the following figures. The third prototype was given to a initial group of SQR developers and their managers for validation of the human computer interface.

4.2 DOMAIN DESCRIPTION

The application report suite consisted of cost centre accounting and profit centre accounting reports for a large telecommunications manufacturer. The management problem consisted of managing the rapid development of the report suite and then subsequently maintaining the report suite. The manufacturer hired Brio professional services to manage the overall development. SMV was applied to the report

suite in two stages. The first consisted of visualizations on the SQRs as created and placed into

4.4 SOFTWARE TOOL RESULTS

4.4.1 Large application suite view.

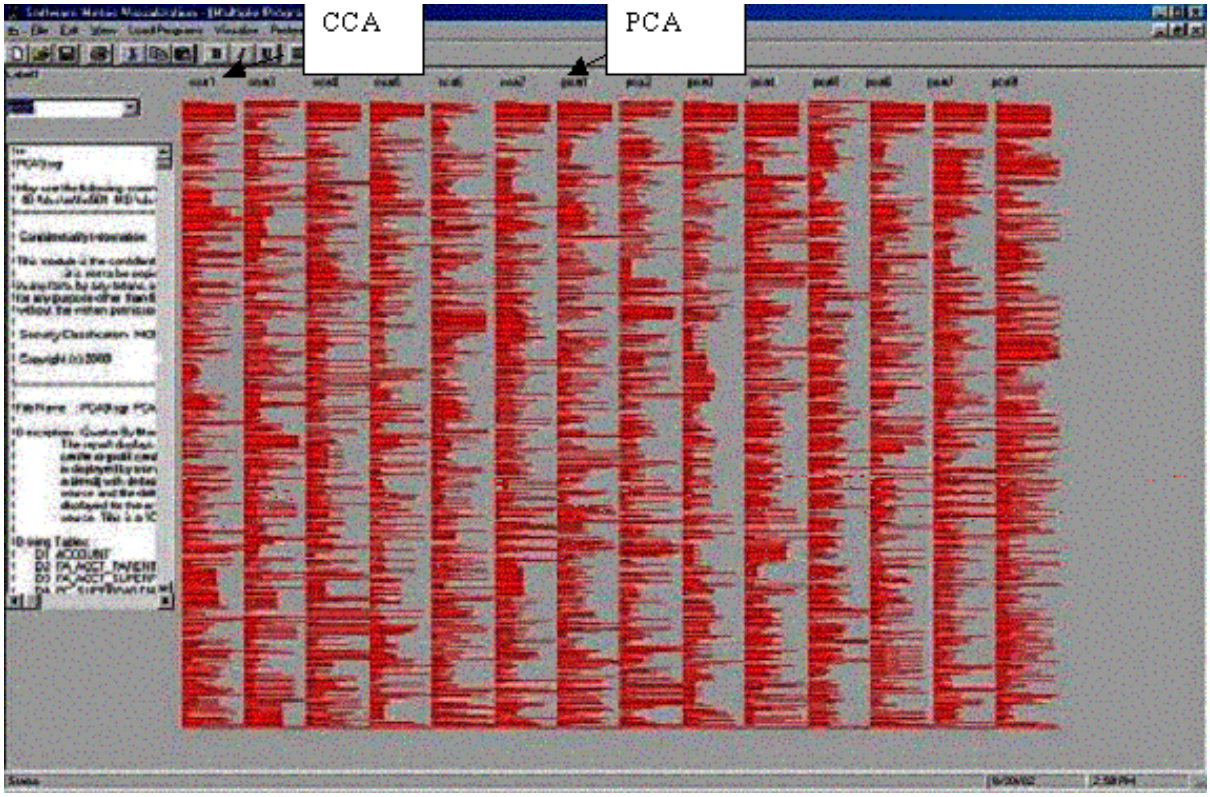

Figure 13 Client application domain with 14 SQRs and 5000KLOCs

In figure 13 the client application consisting of both cost centre accounts (CCA) and profit centre accounts (PCA) is shown. This view was provided by the first prototype and with out colour coding for either lines of code type or for author origination, the view lacks clarity. However, some structure of the code is visible including indentation like the original source code. Figure 14 following provides the second prototype view of CCA including only line type colouring.

4.4.2 An example abnormality

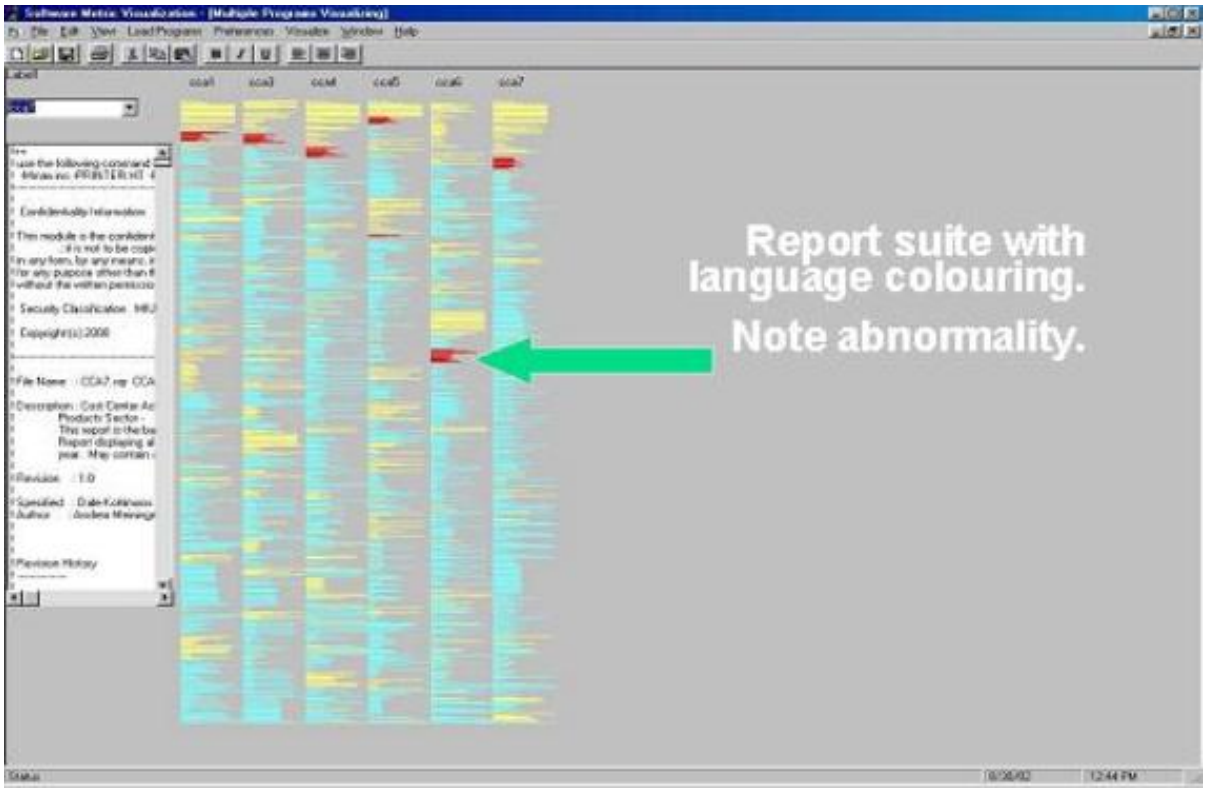

Figure 14 Client subset report suite looking for outliners

Figure 14 above shows the advantages of line type colouring. In this visualization the dependency colour shows that there are 'includes' in one SQR, CCA6, at a program location that none of the other SQRs in the report suite have. So why has the developer for CCA6 got his includes different than the others? In order to answer that question, consider figure 15 drilling down on the SQR CCA6 below.

4.4.3 An outliner in context.

Figure 15 SQR with outliner full screen view

By viewing all of SQR CCA6 the SMV user can see where the 'includes' are used in this SQR. By viewing the list box, jumping to the code before the abnormality and noting the number of defines before the 'includes' the user can see what is different for this developer. This developer’s style is to use 'defines' for local/global variable control rather than take the defaults provided by SQR or the 'includes'. In actuality this developer programmed some of the 'includes' used by the other developers and his variables were more likely to collide with those of the 'includes' during debugging of the 'includes'.

However, by visualizing this program code, there also emerges an interesting pattern. This pattern is of similar language constructs. Indeed, this program uses thirteen monthly 'select' constructs rather than one construct called thirteen times.

This may require a meeting with the developer. Should the developer use thirteen 'Begin-Selects' or one that gets called with variable parameters.

4.4.4 Another kind of application suite overview.

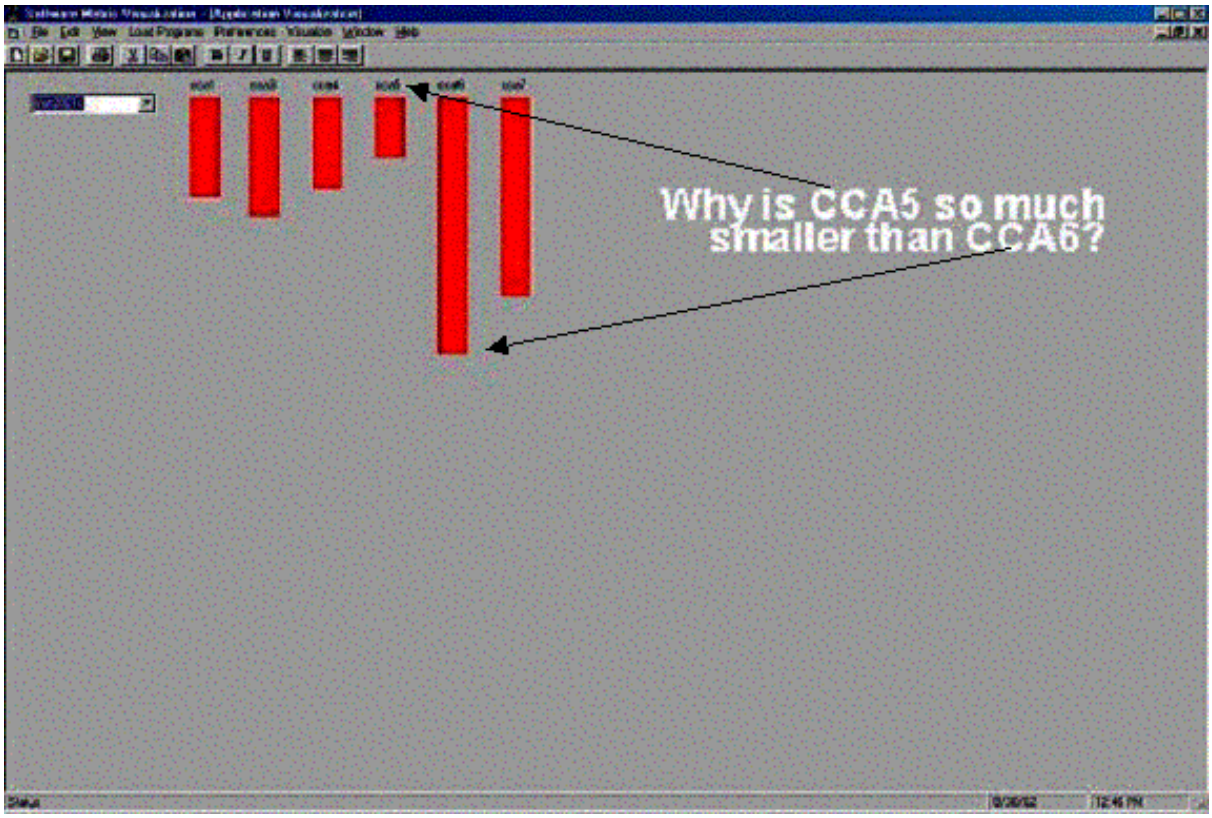

Figure 16 View of application subset by size

CCA5 is way smaller than CCA6. This is because when CCA5 is reviewed there is one 'Begin-Select' called multiple times as opposed to thirteen in line 'Begin-Selects' of CCA6. Hence the differences in program size. It becomes a testable hypothesis which SQR program approach is better. It turns out that CCA6 is slower since the database is called and dropped thirteen times where as CCA5 has only one database connection that is called many times but dropped once. CCA6 is easier to maintain when program or underlining data bugs are found as a given month will be out whereas CCA5 might require advanced debugging to find out why a particular month is out. Similar analysis is possible with respect to the underlying data used in each SQR.

4.4.5 The application suite call tree view.

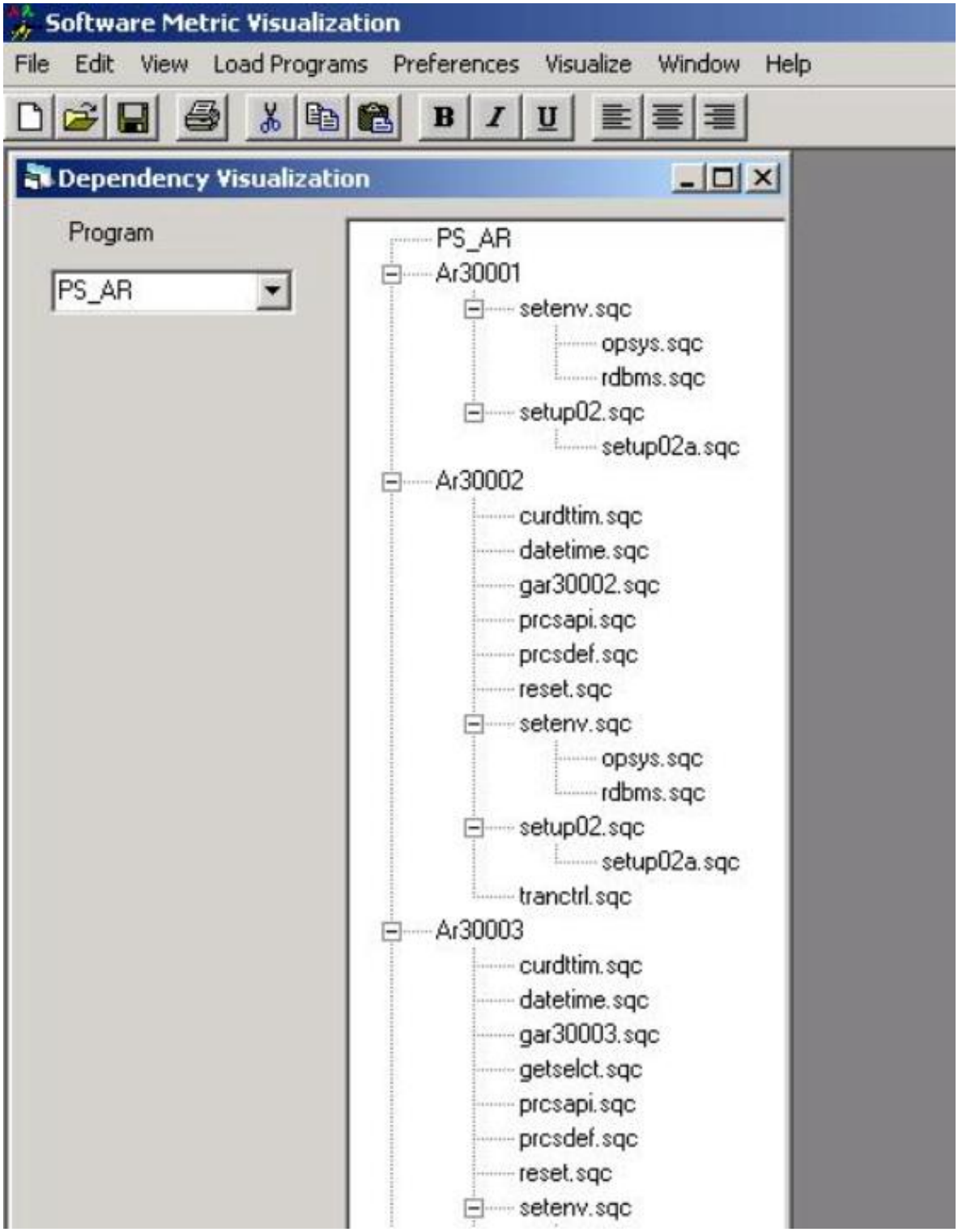

Figure 17 Call tree dependency view of an application

In figure 17 above, the report suite has been changed to Peoplesoft's accounts receivable because the depth of the call dependency tree of CCA was only one. However, the depth in this case shown is 3 and in some Peoplesoft report suites has been measured at seven! Hence changing one SQR may involve reviewing that change across many includes and/or other SQRs within the call tree.. The call tree provides a rapid view of this change. From this dependency view it is apparent how important 'includes' can be to view change events. Although this view is not a line-oriented view like the previous visualizations, it is an important view. It is the first of a slice type approach to visualization. (Slice type

views are visualizations that cross the application suite. Another slice type view might be security if it was implemented in the application suite. Many non-functional requirements can become slices.) In this view some of the components of SQR are visualized as they interrelate to other components. Many additional type slices are possible but this first slice view was chosen because the dependency view provides immediate information that is usually required. In the following figure, 'includes' themselves can be treated as other SQRs.

4.4.6 Includes can be visualized as SQRs.

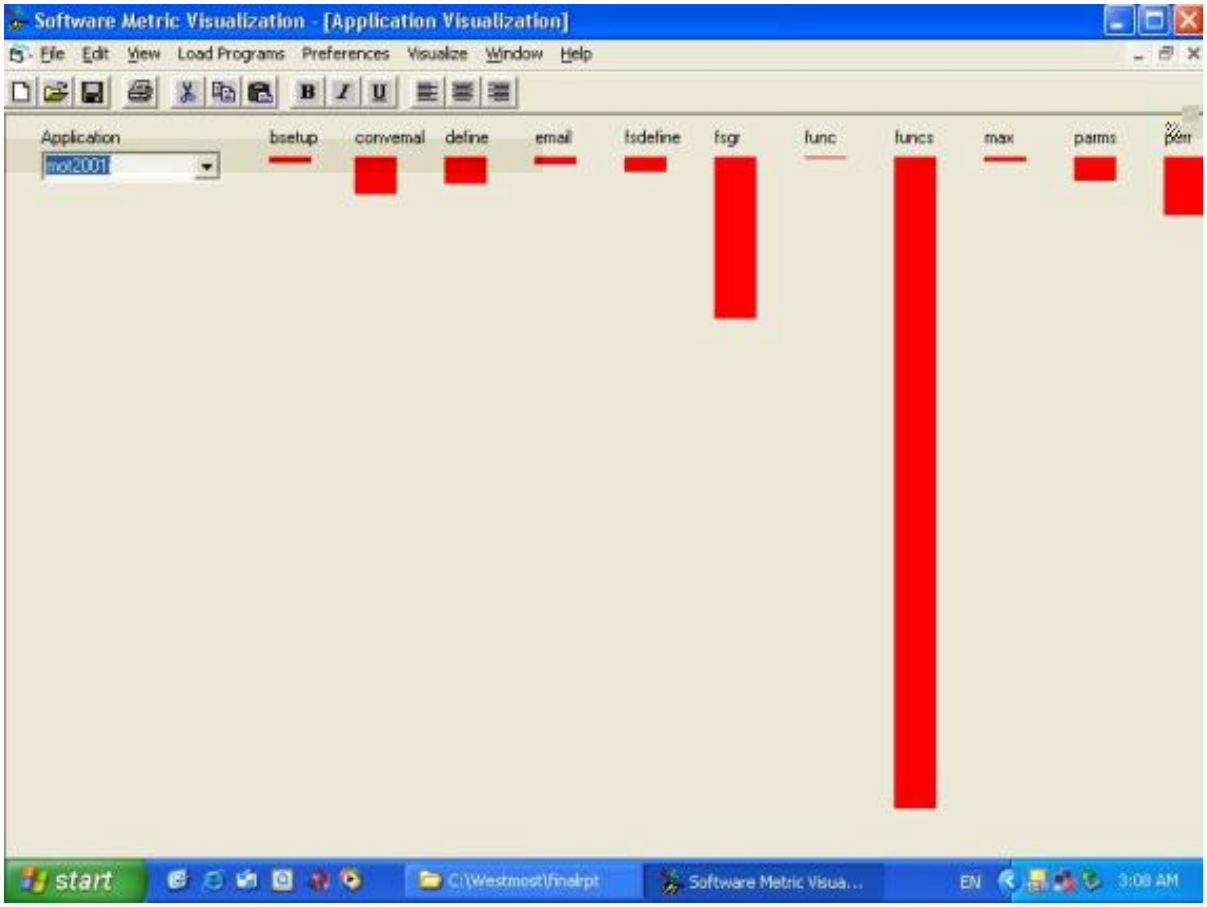

Figure 18 Includes treated like SQRs

In figure 18, 'includes' are visualized as though they are SQRs. This is natural when viewing the call tree of dependencies. Indeed, software developers create their own standard for the identification of 'includes'. The extension for SQR is SQR; however, the extension for includes can be any of the following: 'inc', 'sqc', or 'sqr'. Since 'includes' are really compiler directives they may be have any extension including 'no extension'. It is only by convention that one extension is used over another. In the figure above, note that the source code of the 'include' does not have to be proceduralized. Although from inspection it is obvious that the 'function include' has numerous functions that would be in the form of procedures. However, the 'fsdefine include' does not have any procedures. It is an 'include' of nothing but financial system definitions and 'defines'. A call tree of 'includes' could be built that has as

its starting trunk a collection of 'includes' instead of a collection of SQRs. Considerable feature functionality in an SQR report suite can be developed using 'includes'.

4.4.7 Developers' individual lines are viewed.

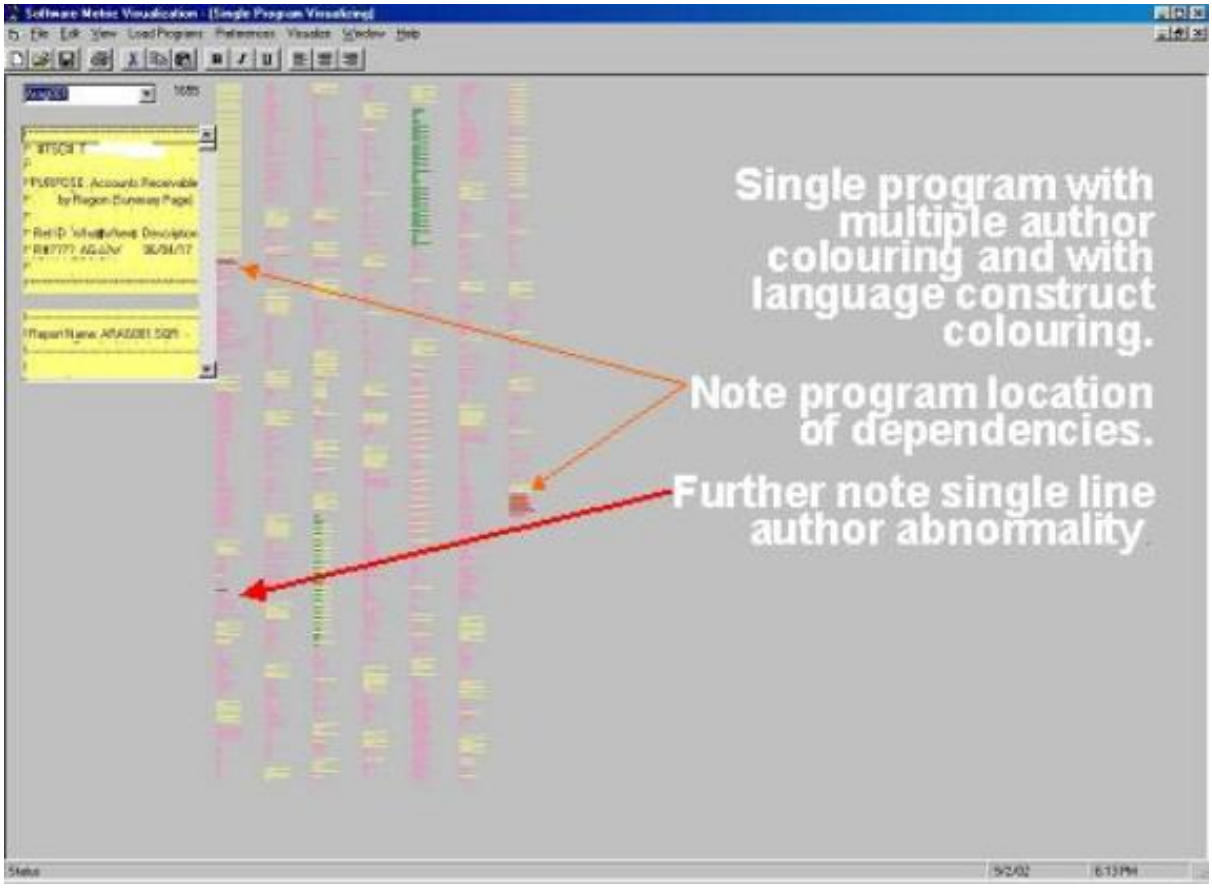

Figure 19 View using language line colouring and other line colouring

This is an accounts receivable SQR that has been modified by another developer. First note where the 'includes' are in this SQR because of the colour coding of the language constructs. Second note that there is author colouring. The colouring of original author in the first five pixels has been added as a result feedback. (In a later prototype, this visualization contains a legend of authors with their colours and language constructs with their colours.) Note the single line of code changed. It is coloured different and different colour filters could be applied. Going to this line of code in the code Pop up window shows that this line is setting the title of the new SQR and is not a big concern. However, it might never have been noticed if a textual view of 4000 lines of code were required to be read.

4.4.8 The development manager's dashboard.

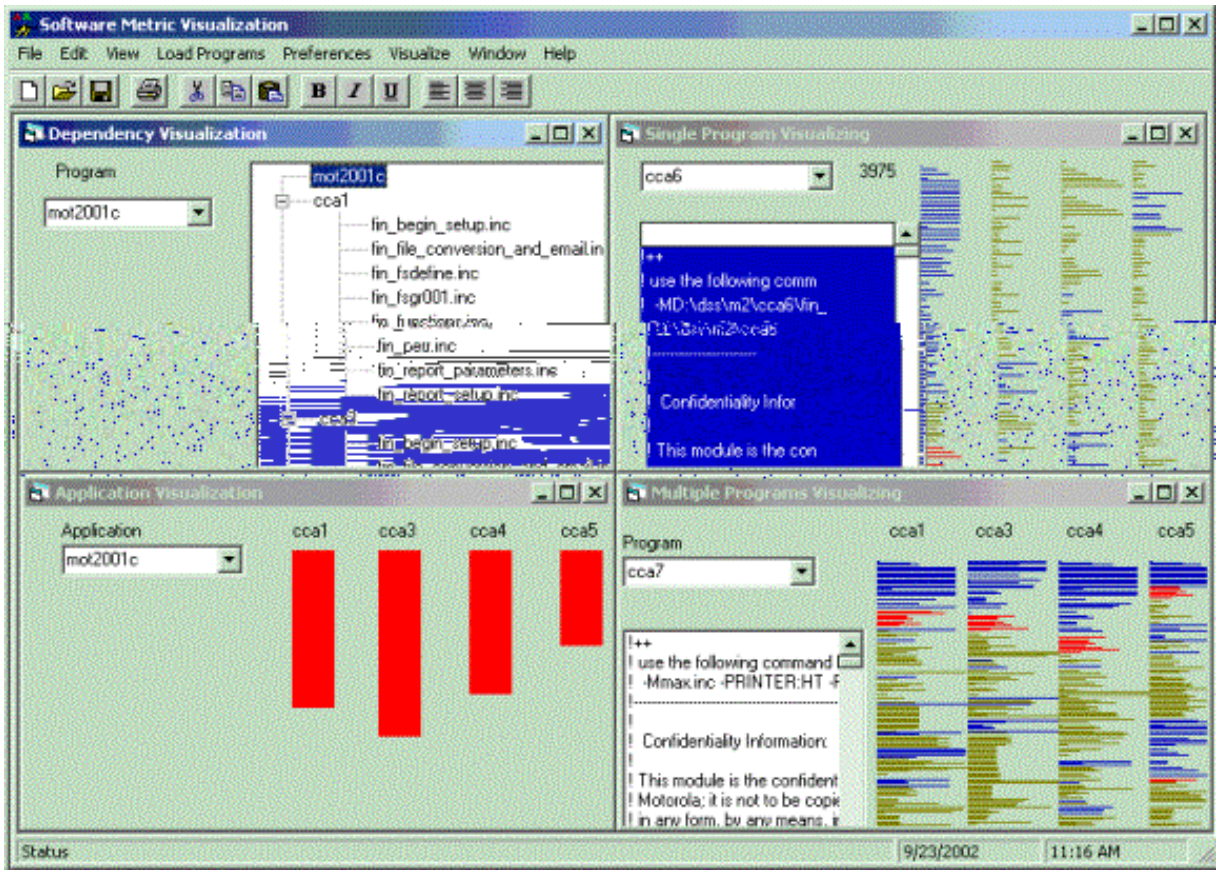

Figure 20 Dashboard like effect using MDI forms and multiple views

Figure 19 is dashboard like because it shows the state of the software with 'dashboard' like features. Although real time updates of the dashboard are possible since the source code is in a SQL database server with triggers on the server change events, usually this kind of real time update is not needed for small to intermediate development projects. However, it would be interesting to experiment when managing say 10 or more developers working concurrently on a large scale development to view overall progress and project contribution through using a dashboard. The dashboard could provide for inspection of source code change events with drill down and with KPIs of individual developer performance.

Dashboards for a software development manager of SQR projects could consist of multiple concurrent views of developer activities. The views would be drillable to the level of manager interest and would have proposed indicators for a manager's review. The red, yellow, green signal lights usually highlight many indicators and provide a way of intuitively knowing underlying software performance. End users would want dashboard features to reduce and/or abstract the amount of information coming into to them concerning developer and development activities.

4.5 END-USER EVALUATION

4.5.1 Sample Human Computer Interfacing Dashboard

The following figure shows the most popular visualization of SMV was the dashboard like view.

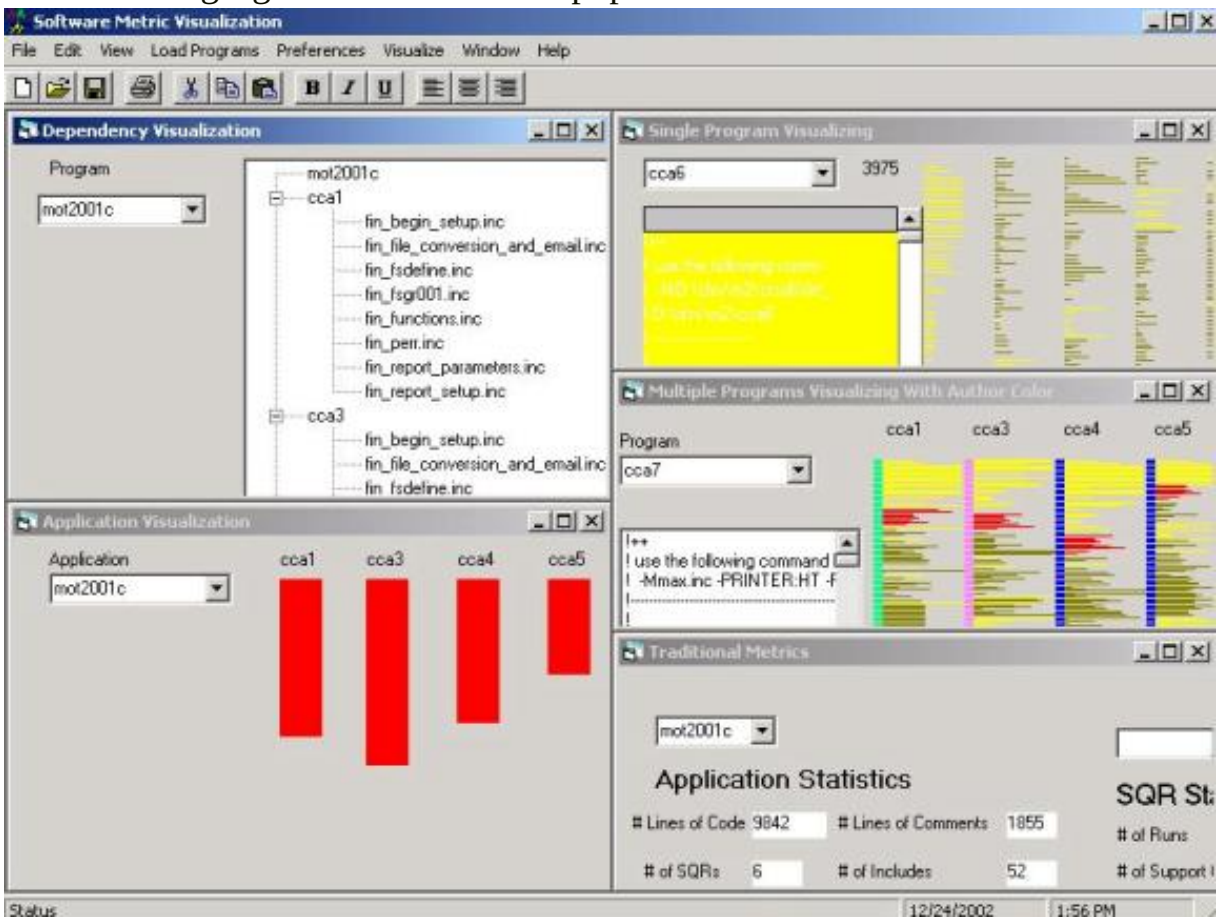

Figure 21 Most popular visualization of large SQR application suite.

In figure 19, the prospect of a dashboard is presented. This could be the forerunner of a true dashboard for software metric visualization for SQR in real time. Figure 20 was the most popular view of the source code from the initial group or panel - not the individual views or visualizations. This dashboard view could be launched automatically upon start up of the SMV and be refreshed say hourly and/or on source code change events. Sustaining the dashboard this way would allow system development managers to monitor underlining source code development activities across an application domain in real time. This option may be attractive for large-scale application suite developments. Adding dashboard features of signal lights, highlighted drill down, and key indicator tracking would make the SMV a very useful tool.

4.5.2 Summary of Main Findings on SMV Interface

The main human computer interface findings of the SMV interface are presented below in table form. Each of the nine identified interface components is listed in their own box and simple comments are provided as captured from the initial group or panel. These comments also provide insight into the state of the design of SMV at the time it was presented to the users. The fourth prototype was the final design presented to the end user group. The headings were selected from among comments of the group.

Table 2 SMV Interface Evaluation

Simple and natural dialog • The natural dialog is in English and is with a low fog index • The ease of learning to use SMV was medium speed.	**Speak the user's language** • SMV provides the user with language constructs in SQR • SMV provides the user information in both numbers and in terms of views	**Minimize the user's memory load** • The present design of SMV requires the user to remember why they are drilling down on a view. By using midi forms the user can see a limited form of progress • There are not example scenarios for a user to follow with example SQR data.
Be consistent • The SMV uses a Windows look and feel. • All colour coding of change events remains fixed until the user changes these colours.	**Provide feedback** • The view is at the speed of processor to the user. • Feedback is not in auditory form. This would be addressed in a new version.	**Provide clearly marked exits** • In performing a series of visualizations the exits are to other office productivity or developer software.
Provide short cuts • Short cuts are provided as part of the menus of SMV. • Additional short cuts and user defined short cuts to selected views or visualizations could be added to a new version.	**Deal with errors in a positive manner** • Errors that result from coding errors in SMV at this time result in the lost of visualizations. An error recovery or debugging module is now required.	**Provide help** • There is no help provided in SMV at this time. • User scenarios and a help compiler could be provided in a newer version.

So from inspection of the comments and from the interviews it is apparent that additional efforts in terms of dealing with errors and providing help in using SMV is required.

4.5.3 Objectives and Task Meeting

The objectives of this research included four main points. The first point was to assist large SQR development projects in the construction and maintenance of SQR artifacts. SMV does this through dashboard like visualizations of the artifacts. The second point was to augment software managers in assessing productivity of SQR developers and overall project progress. SMV does this through providing colour-coded views of individual developer contribution to a SQR application suite. In addition the application suite can be viewed in its entirety at any point of time - resulting in a progress statement at that point in time. The third point or objective was to provide software engineers with an estimating tool for SQR artifact upgrades. SMV does this by colour highlighting the difference in the upgraded application suite with the old application suite. In heavy ERP uses of upgrades this becomes an estimation tool for effort that could be refined over the years with history of colour matching of actuals to estimates. Finally, the fourth point was that software managers would be provided with a rapid snap shot of code status. SMV does this.

4.5.3.1 Revised Task Example 1:

An SQR developer who has written 100 KLOCs in 12 SQRs for a large hospital over 2 years would put these SQRs into an application suite. He would also put his 'includes' in the application suite. He would take a snap shot of the code. When required to make changes to the appliss

activities. From these snapshots he would know developer performance and maintenance effort on any given suite of SQRs. Similar snapshots would occur when he was being asked that he upgrade SQRs for say annual tax changes for an ERP system.

4.5.3.3 Revised Task Example 3:

A delivery manager for a SQR software boutique who needs to convert customers from another report writer would not make extensive use of SMV. He could use the other language feature of SMV but this would require that the other language had flat file capability. Although the author and Darrin Miller [Miller2002] have written SQL Plus to SQR converters; SQR to SQR converters; and PL/SQL to SQR converters, there is no visualizer for language artifacts other than flat files or SQR. Some languages such as Crystal and Envision could be force fitted to create flat files of their software activities and then run through the SMV. However, there are likely alternatives for conversion estimation and visualization.

4.5.3.4 Revised Task Example 4:

A freelance SQR consultant who gets called in to make an SQR product suit more efficient would make extensive use of SMV. Each SQR of the suite would be viewed in SMV and all of the visualization features would be viewed. This would result in knowing where to make most likely code changes with highest likelihood of increasing efficiency. Code changes of programming style and language constructs would be immediately obvious. Replacement of database calls with *lookup table* command retrievals would be immediately obvious. The number of begin-selects in a SQR would be obvious and their possible replacement with dynamic begin-SQL would appear. Many additional applications of SMV would be possible to the extent the freelance SQR consultant was familiar with SQR and SMV.

4.5.3.5 Revised Task Example 5:

A delivery manager for Brio professional services gets tasked with rapidly creating financial reports for a data warehouse that is being fed data from a newly installed ERP system would use SMV. Indeed this was the sparking example for the creation of SMV to begin with. In a limited view, this was the reason SMV was created in the first place. All of the current design of SMV would apply in this revised task.

4.5.3 State of Design

The state of the design is of medium to high quality with the initial group users reporting usefulness of

the visualizer. However, the implementation in Visual Basic 5 of the prototype is low to medium quality. SMV should be redeveloped using Java, SQUEAK, or Visual Basic .NET and alternative architecture using a multi tier approach should be built. However, the simplicity of the design meets the entire core requirements identified earlier. The increased functionality as the prototypes were developed has led to improvements in SQR software metrics. In general, the design with the identified failures summarized above appears quite solid. No reported problems have been indicated from the initial group of original users and their tasks - just a desire for more features and to have a complete version. The overall design was a Visual Basic GUI client to a SQL database. Connectivity was achieved using ODBC. Component reusability was developed for many of the presentation graphics and as a by-product of using Visual Basic. A major redesign decision would be using vector scaled graphics as opposed to pixel-generated graphics. However, implementation of vector scaled graphics would require a virtual machine that addressed sub pixel drawings on what ever target display hardware that was used. SMV interrogates the display hardware and related driver and an attempt is made to optimize the visualizations to the display. This works well in a Windows environment and from inspection of the screen shots presented in this paper all recent versions of Windows are presented. Also presented is different granularity of displays, which can highlight more or less KLOCs per display.

The SQL database is a repository of the KLOCs and has many more attributes in it then have been presented in this paper. The loading of the database is accomplished by SMV through using Visual Basic data management language commands but could easily be accomplished using SQR. This was not done, as proliferation of a runtime only version of SMV was desired for end user interface testing. A disk of the runtime version is available along with GNU licensed SQRs from the author and from the SQR community. Ray Ontko’s contributed SQRs are also bundled with the CD for end user interface testing. However, bundling of a runtime SQR license was not possible with the first CD. During loading of the database, parsing of the lines of code occurs and some attributes are added. Parsing is necessary as the double quotes ‘”’ do not insert into a SQL database well. There are work- a-rounds provided by SMV.

Considerable more features are available in the final prototype of SMV then have been included in this paper. In particular, language filtering of begin-select compression and other constructs is included in the final prototype. Many additional features could have been added and will be added to subsequent prototypes, if a decision to commercialize SMV is made. SQR code optimization and security checking will be the first added features along with interfacing to the common source code management system files.

5.0 CONCLUSIONS AND RECOMMENDATIONS

This section of this document is self-explanatory. However, the conclusions and recommendations come from using SMV within the initial group and from SMVs application to a large commercial

application.

5.1 IMPACT OF INDUSTRIAL APPLICATION

The use of SMV could consist of the following:

- When you have three or more SQR developers doing a development project, measurement of their productivity and adherence to standards could be visualized by SMV;
- When doing upgrades to legacy software or purchased software SMV could be used to view the differences in SQRs from one version to another;
- When needing a year to year snapshot of software changes SMV could be used to measure/visualize source code activities throughout a given year;
- When the revision control system provides only textual information SMV could complement the textual information with visualization of changes; and
- When checking escrow source code SMV could be used as a rule of thumb measure of source code changes.

There are many other impacts that SMV could have; however, the above capture the main impacts. Visualization techniques combined with a useful interface provide another way of viewing source code, other than textual.

5.2 ADOPTION ISSUES

The major adoption issues could be solved with marketing and enhancement of the current SMV version. Marketing would result in easing the learning curve of use and the enhanced features of graceful degradation on SMV failure combined with HELP would result in easier adoption of the visualizer.

The following list some of the adoption issues that could be addressed:

- Compatibility with other metrics and when to use statistical measurements, hard counts, standard graphic representations, and when to use a visualizer. Software managers have their own ways of measuring and SMV could be tailored to adopt user preferences of measuring.
- Scalability to very large report suites that typically include mixed computing language reports. SMV does not support multiple language application suites and/or systems well. It has been optimized to visualize SQR. However, interfacing to other flat file source code languages is possible with language identification through file extension.
- Granularity of visualization and how to set appropriate defaults for the end user of the visualizer. SMV now creates its own optimization of the display of a given visualization. An alternative approach might be to allow user preference setting that would remember the last user display of a given visualization and remember the overall user selected defaults for the SMV tool itself.

- Simplicity in reporting and analysis and how the end user learns or acquires insight given the kind of media that is being presented. SMV could have a repository of routine visualizations that an end user usually does. This repository would then allow dashboard like timeframe setting snap shots of a collection of visualizations. This could result in many spin-off applications of SMV. For example; security is becoming a major concern. Many SQRs are attached to a Brio Portal and/or Brio Performance suite and could be subject to malcode attacks. Viewing snapshots with colored highlights of underlying changed source code could show possible security breaches. Also, searching SQR application suites for Trojans and other in-planted malcode by disgruntled developers would consist of looking for key language constructs. The author created an Easter egg in an application suite; however, to launch the Easter egg required a call command to the operating system and this could be trapped.
- Agency and visualization learning or can the visualizer learn the visualizations of the end user such that repetition of visualization with changes to underlining source code data occurs independent of end user initiative. This point is similar to the point previously made. However, in this case, SMV would 'learn' from the user what visualizations to run and perhaps compare these visualizations to previous ones or to known expected patterns. This would allow service like operations of SMV where a tune up of SQR was performed say annually or for a source code escrow agent.

These issues should be addressed in a separate document and would be the objective of further SMV research and development.

5.3 IMPLEMENTATION PLAN

The major implementation plan is to continue to add features to SMV. Rewriting is possible and may occur. Spreading of the revised SMV should likely occur through providing an evaluation copy to the end user community likely through a web call out and download.

5.4 CONCLUSION

Visualizing augments, it does not replace standard metrics. SMV has added direct SQL for those who prefer their own counts; however, SMV itself provides a form of interface to source code management that is useful.

SMV currently has a repository of 170,000 lines of SQR in 2 report suites. SMV has not loaded say Peoplesoft Payroll but think this is a natural for confirming upgrade activities for those who have copies of SMV and Peoplesoft. SMV is currently attempting multi language visualization for a web enabled wellhead software suite at 1.5 million lines of code.

The author has started to play with the granularity of the visual representation and is currently using twips; but thinks that language construct filtering may be a better visual than increased visual shrinkage.

Once a user has a collection of settings and properties that represent the kind of visualization of interest to oneself, it is very easy to do periodic snap shots of things like progress in development, changes in code migrated across a report suite. To those not intimately familiar with SQR it becomes a visual assist to managing.

Visualizing can provide additional management insights into SQR coding activities and its maintenance. These insights can be provided without intimate knowledge of SQR programming.

APPENDIX

- Ethical Research Form
- Human Computer Interface Questionnaire
- Search Engines Used
- References
- Participants Requiring Email Copy of the Report

A. Ethical Research Form

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ALBERTA
INFORMED CONSENT FORM

Research Project Title: WM099 Visualizing SQR Quality Metrics
Investigators: John James Willson

This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.

This study is concerned with evaluating created interfaces to a thermostat. Your experiences and comments during this study will be used to analyze problems and make recommendations for improving the system.

The study will require 10 minutes during which time you will be asked to carry out 3 tasks using the

system. After the tasks, there will be time for you to make additional comments about the system, and time to discuss the goals of the study.

As one way of thanking you for your time we will be pleased to send you a summary of the results of this study once they have been compiled (probably in about 4 months). This summary will outline, in general terms, the problems found with the interface and our recommendations for improving it. If you would like to receive a copy of this summary, please write down a contact address here.

Contact address or
email:_____

All of the information we collect from you (observations and notes made by the experimenters) will be stored so that your name is not associated with it (we will use an arbitrary participant number). The write-up of the data will not include any information that can be linked directly to you. The research materials will be stored with complete security throughout the entire investigation. Do you have any questions about this aspect of the study?

We may wish to quote one or more of your comments about the system in our final report, in order to better illustrate problems or possible solutions. Any such quotations will be carefully presented to ensure that it is not possible for anyone to trace them back to you. Do you have any questions or reservations about this?

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to not answer specific items or questions in interviews or on questionnaires. **You are free to withdraw from the study at any time without penalty.** Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact:

John James Willson: Department of Computer Science, University of Alberta
Telephone: (780)470-3744
email address: jwillson@dssltd

Participant's signature:_____

Date:_____

Investigator's signature:_____

Date:_____

A copy of this consent form has been given to you to keep for your records and reference. This research has the ethical approval of the instructor of Westmost WM-099 – Dr. Ken Wong.

.

B. Human Computer Interface Questionnaire

Research Project Title: **WM099 Visualizing SQR Quality Metrics**
Investigators: John James Willson

1. What tasks are required to manage or maintain an SQR software suite?

2. Who else is involved in determining what software metrics should be?

3. What software/SQR quality metrics or measurements do you use now?

4. How do you control your software development and/or maintenance projects now?

Work breakdown structure b program	
Work breakdown structure by knowledge domain	
Assignment of the most qualified resource	
Outsource to most qualified firm and/or resource	
Ad hoc	
Other (please explain)	

5. What features in a software quality visualizer are important to you ?

Interface to current version and /or revision control software	
Ease of loading and unloading SQRs and/or includes to be visualized	
Range of granularity of presentation of visualization	
Interface to an editor and/or other tools such as Brio report	
Clarity of visualization	
Ability of the visualizer to "learn" likely visualizations	
Ability of the visualizer to represent information through - language constructs - author efforts effective dated - SQR and other languages - Dependencies - Cross referencing - Other (please explain) -	
Ability of the visualizer to cut/paste/print into other office productivity tools	

6. How often do you use software quality metrics?

7. How many different types of software quality metrics or visualizations have you used?

8. What is the most important change that you would make to the software metrics that you have used?

9. What have we failed to ask about software metric visualization that you would like us to consider?

C. Search Engines Used

```
#URL
        NAME=Yahoo
        URL=http://www.yahoo.com/
#URL
        NAME=WebCrawler
        URL=http://www.webcrawler.com/
#URL
        NAME=Northern Light
        URL=http://www.northernlight.com/
#URL
        NAME=MetaCrawler
        URL=http://www.metacrawler.com/
#URL
        NAME=McKinley
        URL=http://www.mckinley.com/
#URL
        NAME=Lycos
        URL=http://www.lycos.com/
```

```
#URL
        NAME=LookSmart
        URL=http://www.looksmart.com/
#URL
        NAME=Internet address finder
        URL=http://www.iaf.net/
#URL
        NAME=InfoSpace
        URL=www.infospace.com
#URL
        NAME=HotBot
        URL=http://www.hotbot.com
#URL
        NAME=GoTo!
        URL=www.goto.com
#URL
        NAME=Google
        URL=http://www.google.com
#URL
        NAME=Go
        URL=www.go.com

#URL
        NAME=Excite
        URL=http://www.excite.com/
#URL
        NAME=Dogpile
        URL=http://www.dogpile.com
#URL
        NAME=Deja
        URL=http://www.deja.com
#URL
        NAME=Ask Jeeves
        URL=www.askjeeves.com
#URL
        NAME=AltaVista
        URL=http://www.altavista.com/
```

D. References

[Booch2002] Grady Booch *Quality Software and the Unified Modeling Language*, Rational Software Corporation, 2002, http://www.rational.com/

[Brio2002] *Brio Performance Suite 8*, Brio Software http://www.brio.com/

[Burton1994] Peter Burton *SQR User's Guide and Developer's Kit* 1994 MITI Long Beach, CA

[Dask1992] Michael K. Daskalantonakis *A Practical View of Software Measurement and Implementation Experiences Within Motorola* IEEE Transactions on Software Engineering, vol. 18, no. 11 (November 1992), pp. 998-1010.

[Dbminer2002] DBMiner, DBMiner SX 2002, http://www.dbminer.com/

[Dumke1999] R. Dumke *Metrics Tools - An Overview.* Metrics News, 4(1999)1, pp. 21-28

[Eick1992] S.G. Eick; Steffen, J.L.; Sommer, E.E.: *Seesoft -- A Tool For Visualizing Line Oriented Software Statistics*. IEEE Transactions on Software Engineering, 18(1992)11, pp. 957-968

[Eick1996] Thomas A. Ball and Stephen G. Eick. *Software visualization in the large,* IEEE Computer, April 1996, pp. 33-43.

[Emden2002] Eva van Emden, Leon Moonen, jCosmo – *Java Code Smell Browser Tool,* http://www.cwi.nl/projects/renovate/javaQA/

[Florac1999] William Florac and Anita, *Carleton, Measuring the Software Process,* Addison Wesley, Reading Mass.

[GM2002] FQS Poland, *User Manual*, Ghost Miner, 2002,

[Gutwin1999] Carl Gutwin, *Visualizations of Interaction*, Technical Report 99-1, HCI Lab, University of Saskatchewan, 1999

[Knight2000] Claire Knight, *System and Software Visualizations*, Handbook of Software Engineering and Knowledge Engineering, 2000, http://www.durham.ac.uk/

[Kuh1994] I. Kuhrau *A Tool-Based Analysis of Borland C++* Master's Thesis, University of Magdeburg, February 1994.

[Landres1999] Galina Landres and Vlad Landres *SQR in Peoplesoft and Other Applications* 1999 Manning Pulbications, Greenwich, CT

[Miller2002] Darrin Miller *Harnessing SQR* Brio Software User's Conference November 2002, http://www.brio.com/

[Mellen1998] Don Mellen *SQR Programmer Reference 1998,* Ray Ontko & Co, Richmond, Indiana

[Norman1990] Donald Norman, *The Design of Everyday Things*, Doubleday, New York, New York

[Nielsen1999] Jakob Nielsen *Interfacing with Jakob Nielsen* June 1999 http://www.useit.com/

[Qsm2002] QSM Company, *The SLIM Software Tool Suite*, 2002, http://www.qsm.com/

[Sgi2002] Silicon Graphics Mindset , 2002, http://www.sgi.com/go/mindset/

[Sorenson1993] Boloix, G., Sorenson, P.G. and Tremblay, J.P Software "Metrics using a Metasystem Approach to Software Development", *International Journal of Systems and Software* 20, 1993: 273-294.

[Swanson2001] Gregory Swanson and Lee Globus *Visualization for a Graphical Programming Environment,* Nielsen 2001, http://www.tslice.com/

[Tao1999] Xie Tao, Huang Huang, Xiangkui Chen *Object Oriented Software Metrics Technology* 1999 Ricoh Company Ltd. Tokyo, Japan & Software Quality Evaluation Group Peking University

[Toth1996] R. John, J. Madhur, R. Stewart, K. Toth, *Software Quality Metrics Process For Large Scale Systems Development*, 1996 INCOSE Symposium, July 1996

[Willson1998] John Willson, Stephanie Crafford *U.S.A. Computer Consulting – For Canadians and Other Aliens*, 1998,

DSS Ltd. http://www.dssltd.com/

[Willson2001] John Willson *EXtreme SQR Programming*, Unpublished Brio 2001 presentation, http://www.dssltd.com/whitepapers/

[Wong1996] Kenny Wong *On Inserting Program Understanding Technology into the Software Change Process* Fourth Workshop on Program Comprehension (WPC 1996)

[Wong1999] Kenny Wong *The Reverse Engineering Notebook*, PhD. Thesis, Department of Computer Science, University of Victoria http://www.cs.uvictoria.ca/

E. Participants requiring an email copy of the report

Tlr2@shaw.ca

Robert.goshko@axis-dev.ca

Wcheng@telusplanet.net

Darrin.miller@brio.com

Dan.Thornhill@motorola.com

willhan@shaw.ca

F. Example Citation Searching for Lexical, SQL, Parsing software

Search Terms	Citations or Hits
"lexical analyzer" software	10,300
"SQL analyzer" software	952
Lexical parsers software	10,600
SQL parsers software	15,700