

81940 A Seminar on Reverse Engineering
Software Systems Laboratory
Tampere University of Technology
Fall, 2000

1. Introduction

On this assignment the team had to use a reverse engineering tool to analyze a drawing tool XFig 3.2.1. The particular tool used by our team was Rigi (www.rigi.csc.uvic.ca) on Windows NT 4.0 platform. The list of required tasks was given on the course webpage at www.cs.tut.fi/~tsysta/sem/Handbook.htm, with the actual modification work excluded. The purpose of this document is to present a short evaluation of Rigi, based on our experiences, and to give an overall opinion on the structure of Xfig.

Our reverse engineering process consisted of three parts. First, we had to parse the source code with Rigiparse in order to gain a graph representing the structure of Xfig. Second, we had to visualize this structure by using Rigiedit. Third, we tried to interpret the structure of Xfig by using these Rigi views, together with tools like Emacs and Grep.

2. Installation and parsing with Rigiparse

Installation of the tool was fairly straightforward, but Windows path and file names with spaces caused problems. The user's manual for the Rigiedit tool was quite clear, but in our view insufficient for the analysis of larger programs. Other documentation was poor. In particular, more documentation is clearly needed with parsing and TCL-scripts.

We were not able to parse the xfig program. We tried several approaches on different platforms, but encountered problems with environment settings and incompatibility with different compilers. In addition:

- Compiling xfig failed several times because incompatible environment settings. However, we were finally able to compile the program on Solaris workstation.
- We were unable to get cparse and Microsoft cl compiler to work together. This was partly due to the fact that xfig requires several X11 header files, but this still should not have caused a problem.
- We tried to use makefiles and ready-made scripts but due to environment settings and lack of comments we failed. It seemed that rigiparse is not compatible with Gnu gcc. There were few unclear comments about this, but using the provided profiles did not help. We ended up precompiling the xfig source files in one environment and using rigiparse in another one.
- Rigiparse itself does not accept directories or wildcards as command-line parameters, so one has to write a script to handle all the 100+ source files - a small but very annoying detail.

- The resulting, evidently corrupted .RSF file was transferred from Unix to NT. Rigiedit did read this file without reporting any errors, but was not able to visualize nodes, even though it reported having them available. Better feedback would have been appreciated.

Even though there were some support from people familiar with the Rigi environment, the team made several attempts to use Rigi without any actual results being achieved. We finally received a readily-parsed .RSF-file, but it required "cparse"-profile files which were not included in the NT installation package. These files, strangely enough, had to be extracted from a Linux distribution. Fortunately, this problem was finally resolved by an experienced user.

3. Using Rigiedit

The tool did provide a moderately clear visualization (in a form of a graph) of the program to be analyzed when dealing with very small programs, like the linked list tutorial example. However, the convention of illustrating a connection by drawing a line from the top of the providing node to the bottom of the dependent node was not very intuitive. In our opinion, using optional arrowheads would probably have been a better choice. If the graph is manually structured beforehand, navigation is fairly easy. The scripting mechanism of the tool was quite powerful and enables flexible user-defined extensions.

In the following, we give an overview of the problems we encountered with Rigiedit in arbitrary order, together with suggestions for improvements. Some problems may have been caused by the lack of experience in our team, but other were clearly not. According to our experiences, the tool suffered from poor performance and some design flaws. In addition, the documentation, even though it was clear and well-written, did only cover the very basics of Rigi and did not provide more advanced information.

- The tool should maintain a log file containing the operations and results performed by the user. This would help the user to trace back the performed actions, especially when learning to use Rigi.
- There is no cancel operation available when performing heavier functions.
- The user interface is quite poor. It is clearly directly imported from a Unix version using X-windowing, and is inconsistent with standard MS Windows GUI conventions. The windows, dialogs and operations are not intuitive and hard to use and comprehend.
- The program crashed several times. We were not able to trace any particular circumstances under which this occurred, it seemed quite random.
- The program was very slow, it sometimes took approximately 5 to 10 minutes to perform even simple operations like grepping - running scripts was especially slow. In addition, there were no feedback to the user suggesting that the software was still running, the program consumed 100 % of the CPU time and did not respond to the system.

- The documentation does not cover details that are necessary for reverse engineering larger pieces of software. This is, of course, not the responsibility of a program manual, but some hints on how to get started with this particular environment would have been valuable.
- The tutorials were quite superficial and not very helpful. It seemed that large programs were just automagically broke into well-structured graphs reflecting package and module level structure. This was, of course, not the case in real world. We spent quite some time searching for these magical but non-existent features.
- None of the essential TCL-scripts were documented, there was only documentation for less useful scripts for opening and closing windows etc. Considering the fact that the scripting mechanism is the key for using Rigi effectively, this was clear shorcoming.
- There were very few or no comments on the script files, so without some experienced user consultation it would have been impossible to guess what some of the parameters were used for.
- We were not able to load our own scripts using 'source' command, even though we used the same path as the build-in script files. We had to add our own scripts to these existing script files. Furthermore, whenever we edited a script, we had to start Rigidit again.
- All of the menus did not operate properly, probably due to invalid argument passing, so we had to use scripts to perform some of these operations. This situation was confirmed by an experienced user.
- The input field for TCL scripts did not accept necessary characters like brackets, slashes or tildes. Since the input field for TCL scripts also supported only cutting but not pasting, it was impossible to write arbitrary commands on the input field. This caused considerable stomach acid surges.
- The two abovementioned problems also made it impossible to edit rigi variables directly in the program.
- The program was sometimes caught in obscure "middle states" that blocked some operations. These states are exited when the user performed arbitrary interactions with any graph elements.
- The program did not provide clear and informative error messages. Therefore, it was very hard to know why errors occurred and when they were made by the team.
- The program halts every now and then. The maximum halting time was 30 minutes after which we terminated the program.
- There is no undo operation, but it would be very much needed and appreciated.
- The screen updating policy is extremely bad designed, the program should not visually update graphs until an operation is completed. The unnecessary screen updates seriously slow the system down. There should at least be a possibility form switching the automatic updating option off.
- When performing some other operations, the program should, however, give some indication whether it stills works or not. Sometimes all screen updating was halted and the windows became littered and eventually blank. There was no way of knowing whether to wait or not.

4. Evaluation of Rigiedit

All in all, according to our experiences, the rigiedit system was unstable and slow for a comprehensive analysis of Xfig. It was clear that Rigi is not designed for Windows NT and that fact was reflected by the program itself and by the level of its documentation.

Albeit its poor performance and usability, the tool would probably be quite useful for an expert user using mainly scripts. It became evident that the user interface was more of a burden than a help for a user, and it might actually be better to have a very simple but powerful shell application that would draw graphs only when required. Interpretation of the graphs requires a highly routined user.

There are many frequently needed basic operations for manipulating and structuring the graph that should be readily available as scripts, i.e. breaking the program into packages and modules according to source files, and so forth. With a collection of these kinds of middle-level operations, the tool would become much more attractive. We feel that currently the initial learning curve is too steep and involves much extra work in form of writing a collection user-defined scripts before any actual results can be obtained.

Next, we give some additional remarks on Rigiedit while reverse engineering Xfig.

- It should be possible to directly access source code files from nodes in Rigi.
- It is quite hard to comprehend connections between higher-level components when working at a lower level.
- Since Xfig uses X-Windowing, the resulting graph has too many cycles for the Sugiyama layout algorithm to work.
- There should be some kind of support for combining source and header files with the same name (modules).
- One of the most serious drawbacks is the fact that the C preprocessor removes all the include relationships between source and header files. Since header files can be seen as representing interfaces, this destroys important information on the relationships between different modules. In a sense, these preprocessed files reside at a lower level of abstraction. In real life, a designer usually first browses through header files in order to get a picture on the responsibilities and services different modules (files) provide.
- There should be a clearer distinction between the underlying model and the views in Rigi. Currently, the user is forced to make alterations to the underlying model by, for example, collapsing structures.

5. Analysis of Xfig

Since Xfig is an X-window program, it is infected with callback routines. Therefore, generating a call graph really does not provide much additional information on the structure of the program. More generally, it seemed that doing reverse engineering on GUI-intensive software is quite hard. The data structures and call mechanisms involved

clutter up the code and makes it harder to comprehend. This caused, amongst other things, strange interrelationships between different functions.

As other teams on this course have already pointed out in their presentations, there are functions and files that are too large and complex, like main function and e_edit file.

There are five goto statements in Xfig. The can, of course, be removed, but we feel that this is unnecessary and potentially dangerous. Gotos are used very locally in functions with deep and complex control structures.

After investigating Xfig we reached the following conclusions. First, there are signs of overall architecture that has been initially applied when designing Xfig, but it seems that the actual design and implementation has not followed this architecture with determination. The structure should have been more fine-grained. This can be seen in the current structure of the program which has eroded, even though it does still reflect the original architecture. There are also signs that some parts of the original program have been extended and eventually raised to a higher level as new files or modules.

The overall structure of Xfig is relatively clear. However, the level of abstraction of the data types and design of interfaces is not particularly good.

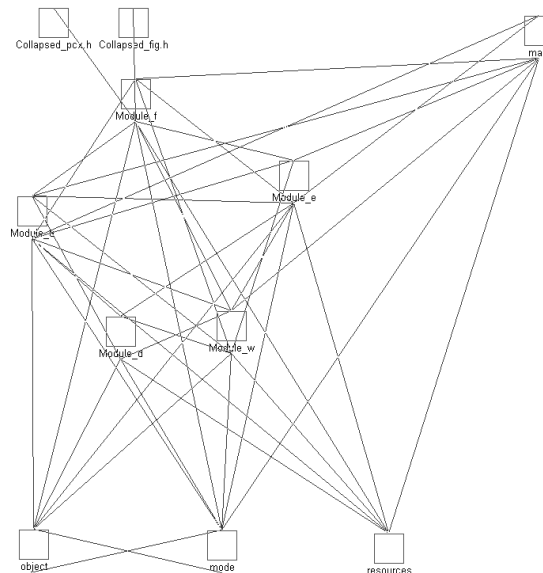


Figure 1. Main modules of XFig.

Figure 1 shows the main modules of XFig as composite nodes and their interrelationships as connecting arcs. These modules reflect the overall structure of XFig and reveal that program division is partly based on the organization of menu operations, for example Module_w for Windows menu, Module_e for Edit menu and so forth.

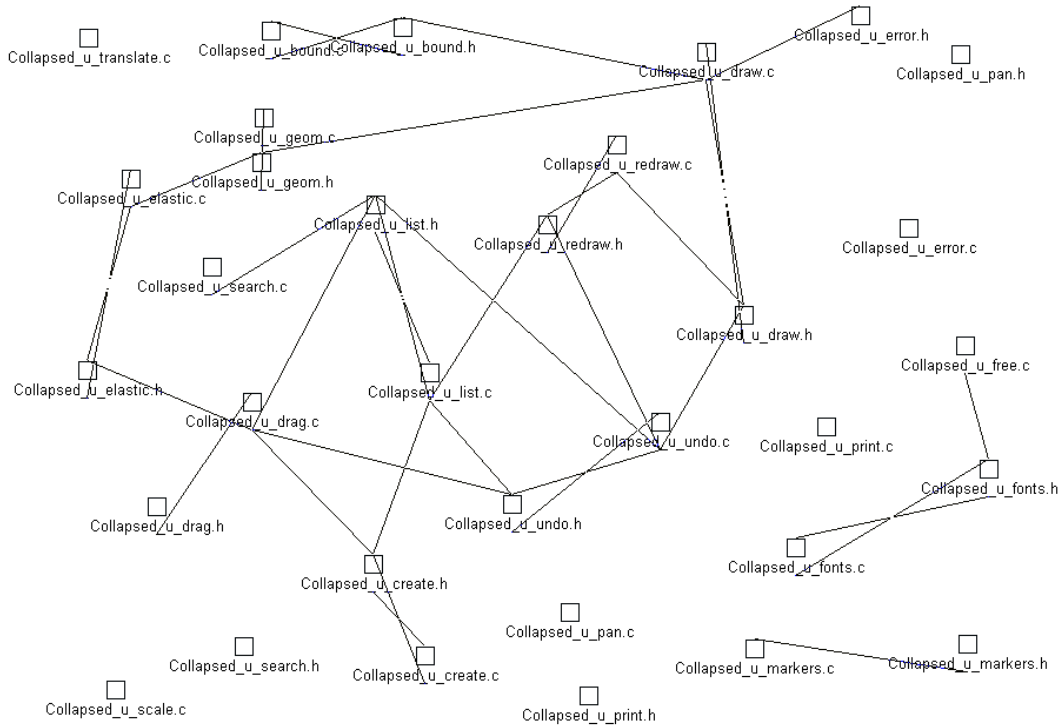


Figure 2. Source files for Collapse_u module.

Figure 2 shows the source files for one of the modules shown in Figure 1, the Collapse_u module. As mentioned before, information on which .c-files include which .h-files is lost during preprocessing stage, so this figure does not exhaustively show all the interrelationships between files, but it does give one projection on how different files inside this particular module are related.

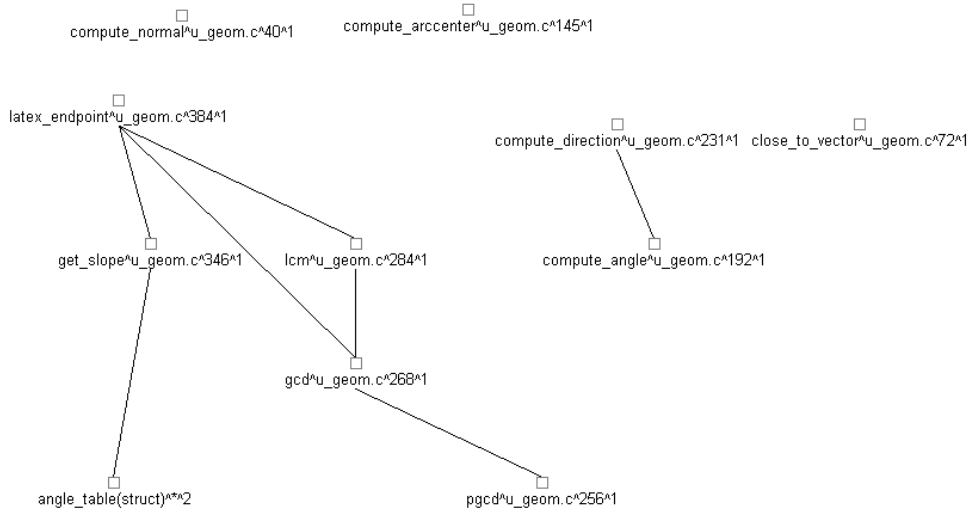


Figure 3. Internal call structure of u_geom.c.

Figure 3 shows internal function (and data) dependencies inside one source file, `u_geom.c`, which is used for geometrical calculations. In this source file the functions are relatively small and concentrate on performing single tasks, so it is readable and clarifies how these particular operations are performed.

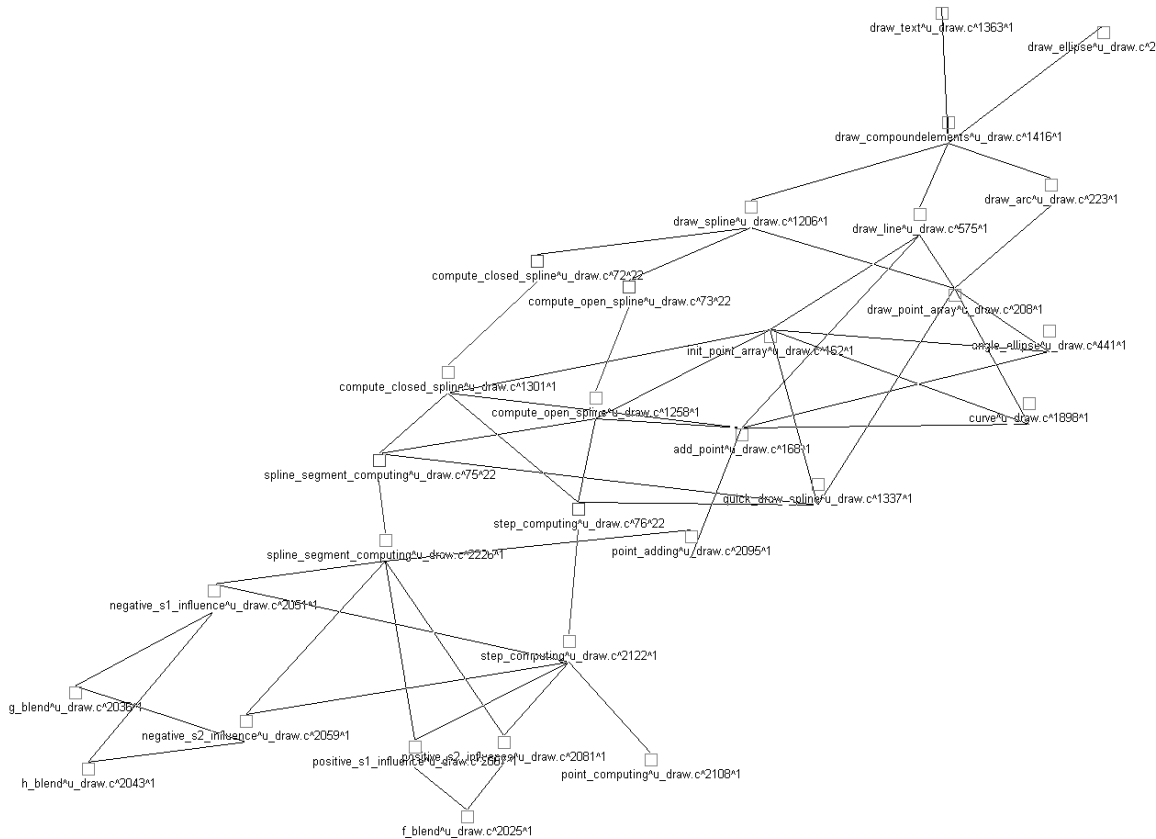


Figure 4. Internal call structure for `u_draw.c`.

Figure 4 shows another call structure, this time taken from source file `u_draw.c`. The structure shown in this figure is much more complex than the one shown in Figure 3. It can be seen that this particular file contains an excess amount of functions with quite complicated interrelationships. Even though this illustration can clarify how drawing functions relate with each other, it also compromises the rationale and manageability of the organization shown here.

6. Concluding remarks

We did use Rigi views to conclude that there are heavy interdependencies between functions and modules in Xfig. They also drew our attention to some overly complex functions. However, we feel that by using tools like Emacs, Grep and some kind of code analyzer, we would probably have completed our analysis and possible repairs on Xfig probably with less effort. Because of the high initial overhead involved, the net contribution of Rigi was not very large, at least in this case.

It did not become clear whether a reverse engineering tool can significantly improve understanding of programs similar to xfig. Of course, the ability to browse a program and to create different views from different viewpoints and levels of abstraction should significantly improve program comprehension.