The Decomposition Slice Display System

Bradley M. Kuhn

Dennis J. Smith

Keith B. Gallagher*

{bkuhn,densmith,kbg}@cs.loyola.edu
Computer Science Department
4501 North Charles Street
Baltimore, MD 21210-2699
410–617–2854

Abstract

The *Surgeon's Assistant* is a CASE tool for software maintainers and developers of ANSI C programs. It is based on the premises that maintainers need to be able to limit the scope of changes and that (most) development starts from some existing, similar systems.

The Decomposition Slice Display System (DSDS), a component of the Surgeon's Assistant, automates the task of inspecting the relationships between a program's decomposition slices. It allows the software maintainer to visually comprehend the relationships between the decomposition slices. The DSDS has a powerful interface that easily represents decomposition slices for the user's perusal. It allows the user to manipulate the graph by moving and deleting nodes. The interface assists the software maintainer by providing an easy visualization of the software. The maintainer can see the importance that various decomposition slices have to the program and plan maintenance accordingly.

1 Introduction

Program slicing is frequently used to assist software maintainers. Software maintenance tools are available that utilize program slicing. These tools usually only provide a textual interface to represent program slices to the maintainer.

The Surgeon's Assistant is one such tool. This CASE tool uses decomposition slicing to assist maintainers in limiting the scope of their changes. The Surgeon's Assistant allows for selection of variables for slicing and then textually displays the decomposition slices. Syntax directed editors and other such components allow the maintainer to comprehend the scope of her change and perform her change without affecting the slice's component.

Unfortunately, textual representations limit the maintainer's understanding of the program, particularly for large systems. Currently, few systems provide graphical representations of program slices. The Decomposition Slice Display System (DSDS) is an attempt to provide such graphical representations for the Surgeon's Assistant. Although graphical interfaces are the panacea for the problems caused by limited textual representations, they still provided a superior understanding of the program slices.

The rest of this paper is organized as follows: Section 2 provides background information and discusses related work. Section 3 describes the DSDS in detail. Section 4 discusses the implementation of the DSDS. Section 5 proposes possible future work in the area, and Section 6 provides a summary.

2 Background and Related Work

This section provides background information on decomposition slicing, the Surgeon's Assistant, graph visualization, and software visualization.

2.1 Decomposition Slicing

Decomposition slicing [GL91] is an extension of the technique of program slicing [Wei84]. A program slice captures all the computation that contributed to the value of a variable at a particular point, usually a line number. Program slicing can be applied to program integration [HRB90], data-flow testing [KL90], debugging [LW86] and program understanding [Wei82]. Methods to extend and improve the techniques of program slicing are still appearing [Bin93, RHSR94].

*This work has been supported in part by the National Science Foundation under grant CCR-9110965.

Decomposition slicing applies program slicing to the maintenance problem. A decomposition slice captures all computation on a variable. This permits a maintainer to analyze all effects of proposed changes to a particular variable and to isolate the effect of those proposed changes. Decomposition slicing does not depend on any specific technique of program slicing, per se, but faster and more accurate slicing algorithms certainly improve the quality of the decompositions.

2.2 The Surgeon's Assistant

The *Surgeon's Assistant* is a CASE tool for software maintainers and developers of ANSI C programs. It is based on the premises that maintainers need to limit the scope of changes and that most development starts from some existing, similar systems (and can therefore be classified as "maintenance").

The Surgeon's Assistant provides an environment that is based on a new software maintenance process model proposed in [GL91]. The maintainer can perform decomposition slices on given variables. Then, a textual display of the slice's source code is displayed beside its complement[1]. The maintainer is permitted to edit portions of the slice that do not affect the complement. After testing the slice as an independent program, the maintainer then merges the modified slice with its complement, thus changing the original program. Note that the maintainer only needs to test the slice, not the complement, to validate the correctness of the change (i.e., under this model, no regression testing is required).

Most of the Surgeon's Assistant provides only a textual interface to view the slices and to affect changes. Textual viewers are provided to view the entire program or individual slices. These textual viewers present a complete analysis of how a change to one slice will affect the program. However, the amount of information can still be overwhelming.

Once a slice is chosen, The Surgeon's Assistant presents a context sensitive editor that only allows changes to the chosen slice. The maintainer may use this editor to change statements that are part of the chosen slice, and test the program with these changes. When finished, the maintainer uses the Surgeon's Assistant to merge the changes with the complement to again create a full program.

This textual interface, although useful, does not provide enough information to truly visualize the effects that the proposed change would have. Pictorial representations of the slices would be a better method to understand these effects. Decomposition slices of a given program form an acyclic directed graph where a node represents a decomposition slice, and an edge represents the predicate "is contained in" [GL91]. If the maintainer can be made aware that the chosen slice is at the top of the graph (i.e. contains many of the other slices), the maintainer may reconsider the proposed change. The textual interface may provide this information by number of lines of code, but gleaning this information from the textual interface becomes impractical when large programs are sliced.

2.3 Graph Visualization

The Surgeon's Assistant is a useful tool for *textual* analysis of software. However, it has been shown that graph visualization can be a useful tool in understanding software [GKNV92]. As was mentioned in Section 2.2, decomposition slices of a program form an acyclic directed graph. In the original Surgeon's Assistant, this graph was maintained only abstractly. Gasner, et al., note that graphs often only become understandable when displayed in a visual, concrete form [GKNV92]. Such is the case with the graph of decomposition slices. However, no systems have been developed that display decomposition slices.

The only other tool that we are aware of that provides visualizations of graphs to aid the software maintainer is VIFOR [RDLK90]. VIFOR shows module relationships. Our work would fit nicely in such an environment. Once the relevant modules were obtained, the code could then be decomposed.

2.4 Software Visualization

In this section, we discuss software visualization and the various software visualization systems that have been proposed. Note that few software visualization systems deal with program slicing. However, we feel that it is helpful to look at software visualization in general, since this can give insight to aid in the visualization of program slices.

Domingue, et al., define software visualization as "systems that use visual (and other) media to enhance one programmer's understanding of another's work (or his own)" [DPE92]. Since the 1980's, software visualization has become popular both in education and industry. Researchers have found that software visualization can aid educators when teaching software development and programming. Software visualization can also help professional programmers when debugging and looking for algorithmic solutions.

[1] A complement is the remainder of the program that does not contain the selected slice.

Price, et al., set forth a detailed taxonomy of software visualization. They discuss many systems that use software visualization [PBS93]. Their taxonomy has categories that rate each software visualization system in the areas of scope, content, form, method, interaction, and effectiveness.

Eisenstadt, et al., discuss the use of software visualization systems in teaching environments [EPD]. They discuss methods for developing software visualizations that span the range from novice student to experienced developer.

Current software visualization systems evaluate and understand algorithms [Bro91], visualize parallel programs [Kun94, KC], assist in debugging [EB88], aid understanding of control flow or data flow of programs [Int91] and present source code in a more understandable manner [Knu84]. Those that aid in software maintenance [MK88] do not utilize program slicing.

We are aware of only one system that currently visualizes program slices [BE94] (called SeeSlice). However, SeeSlice does not serve the same purpose as the DSDS. SeeSlice is based on slices of program statements rather than decomposition slices of variables. In addition, SeeSlice provides an interactive system to dynamically view separate program slices. The DSDS provides a static graph of the relationships between all decomposition slices in a given program.

Few slice visualization systems exist probably because software developers often feel that there are no demonstratable gains from software visualization systems [PBS93], and thus are unwilling to invest time and money in them. We feel that the DSDS, along with the Surgeon's Assistant, provides useful information that will save the software developer valuable maintenance time.

3 Decomposition Slice Display System

The DSDS adds software visualization techniques to the Surgeon's Assistant. It provides a useful augmentation to the already existing textual interface. Thus, it is easily used in collaboration with the textual interface.

The DSDS provides a visual representation of the decomposition slices of a given program. The DSDS allows the software maintainer to visually comprehend the relationships between the decomposition slices.

The DSDS creates a acyclic labeled directed graph where a directed edge represents the relationship, "is contained in," between decomposition slices. The descendants of a node (i.e., a decomposition slice) for a given variable, `n`, are the set of variables whose values affect the value of `n`. Thus, a node that has many descendants relies heavily on other variables in the program. Nodes that have no children are completely independent slices that are not affected by any other slices. Thus, the maintainer can easily tell which variables are key to the program.

The DSDS also uses color to increase the understanding of the slices. When the maintainer selects a slice on a particular variable for change, the DSDS colors those nodes that represent all slices contained in (i.e., that depend on) the chosen variable's slice. The maintainer can get a visual "feel" for how drastic her change will be. For example, if the slice selected highlights a large portion of the DSDS graph, the maintainer will know that her change is quite drastic. This may perhaps lead the maintainer to decide that her new feature would be better as an independent program rather than part of the current program.

The DSDS user interface is also helpful to the maintainer. The maintainer can click on nodes to delete them from the graph. This is useful for larger programs with many decomposition slices. The maintainer can remove the clutter made by slices in which she has no interest.

Currently, the system for deleting nodes removes all in and out edges of the deleted node. This system is a "user beware" system since the node and edges deleted could have represented containment relationships in the graph that still exist after the removal of the node. In the future, we hope to add a system whereby the integrity of the relationships can be maintained even when nodes are deleted.

The maintainer may also move nodes in the graph. This is useful since the maintainer may not want to delete a particular slice from the graph, but would like it "out of the way" so that she can better visualize the slices with which she is most concerned.

4 Implementation

The DSDS was implemented in C on DECStations and Linux machines at Loyola College. The Tk toolkit [Ous91] along with an embeddable interface [KB94] was used to create an X window interface for the DSDS. The layout of the graph was implemented using C code, independent of the Tk toolkit used. The user interface and graphical display are provided completely by Tk code.

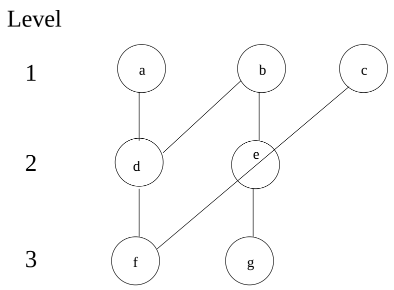

Figure 1: An example of an edge that crosses more than one level

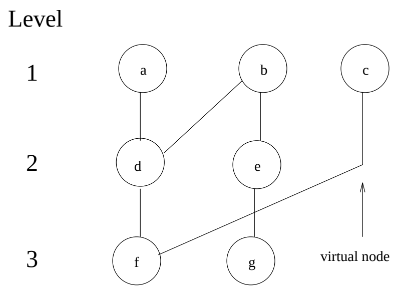

Figure 2: An example of an edge that crosses more than one level using virtual nodes

4.1 Layout Algorithm

Our layout algorithm was loosely based on the *dag* algorithm proposed by Gasner, et al. [GKNV92]. However, we used only their concepts of virtual nodes and rank (level) assignment. Using only these two simple concepts, we were able to quickly implement an algorithm that produced a relatively visually acceptable graph. We realize, however, that a better algorithm will be necessary to represent the graphs of larger systems. Work is currently underway in this area.

4.1.1 Level Assignment

The first step of our layout algorithm is to assign each node to a level. Nodes with no parents are placed on the first level. Next, direct children of those nodes on the first level are placed on the second level. Third level nodes are the direct children of nodes of the first and second levels. This process continues until all nodes have been assigned a level. If a given node's children are not placed on the level directly below them, we use virtual nodes to assure connections are made for nodes which are separated by more than one level.

4.1.2 Virtual Nodes

Once the levels have been assigned, there still exists the problem of edges crossing nodes because two nodes can be separated by more than one level. An example of this is shown in Figure 1. The edge between node `c` and node `f` crosses level 2.

To avoid this problem, we introduce virtual nodes into our graph. Virtual nodes are created on every level in between the separated nodes. Edges are then drawn connecting each virtual node and the virtual nodes on either end to the original nodes. This guarantees that edges will no longer cross nodes on intermediate levels. Figure 2 shows Figure 1 after virtual nodes have been inserted. The virtual node on level 2 prevents the edge from crossing through node `e` on level 2.

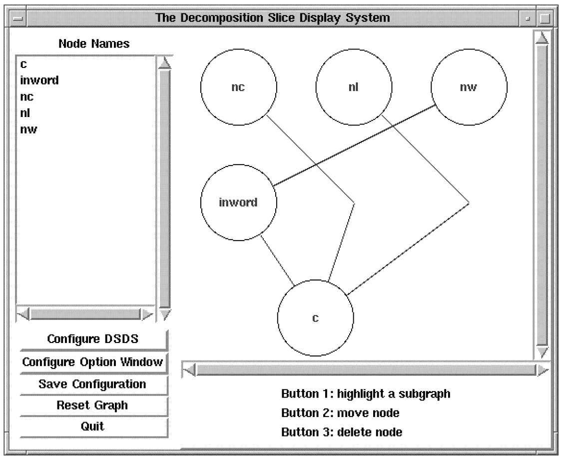

Figure 3: An DSDS window from the program `wc`

At this point, we have had few problems with expressing virtual nodes in this manner. However, there is a potential for this virtual node system to become problematic in larger graphs. Currently, we a researching ways to better express virtual nodes (perhaps through splines).

4.2 Usefulness

The DSDS is a useful tool to the maintainer. It provides a new graphical dimension to the already powerful textual interface of the Surgeon's Assistant. The visualization of the slices on variables gives the maintainer a good idea of how data elements in the program affect each other.

Figure 3 shows a run of the DSDS on the program

```
#define YES 1
#define NO 0
main()
{
   int c, nl, nw, nc, inword ;
   inword = NO ;
   nl = 0;
   nw = 0;
   nc = 0;
   c = getchar();
   while ( c != EOF ) {
      nc = nc + 1;
      if ( c == '\n')
         nl = nl + 1;
      if ( c == ' ' || c == '\n' || c == '\t')
         inword = NO;
      else if (inword == NO) {
         inword = YES ;
         nw = nw + 1;
      }
      c = getchar();
   }
   printf("%d \n",nl);
   printf("%d \n",nw);
   printf("%d \n",nc);
}
```

Figure 4: The source for `wc`

`wc`, whose source is shown in Figure 4. From the figure, one can easily see that the slices on variables `nc`, `nl`, and `nw`, all contain the slice of variable `c`, and that the slice of variable `inword` also contains the slice of variable `c`. This information allows the maintainer to make decisions about what to change. For example, the maintainer who wishes to change the slice of variable `nc` may decide against it, if he does not wish to affect the slice of variable `c`.

5 Future Work

The DSDS currently provides a powerful visualization of decomposition slices. However, as can be seen in Figure 3, edge crossings are still a problem. We hope to use more of the methods proposed by Gasner, et al. [GKNV92] to remove edge crossings from the DSDS graphs.

In addition, we hope to evaluate the tool by slicing much larger systems. To do this we will need to address the question raised by having multiple source files. We will need a method to display decomposition slices that are contained in multiple files in a manner that gives the maintainer an indication of what is going on.

We also hope to add more software visualization techniques to the Surgeon's Assistant. The methods proposed by Ball and Eick for program slice visualization [BE94] appear very useful, and would complement the DSDS well.

Finally, our own exercises in building the DSDS suggest to us a metric for measuring decomposability and interaction between components. This is a new twist on Weiser's proposed "overlap" metric [Wei84].

6 Summary

This paper describes the design, implementation, and usefulness of the Decomposition Slice Display System (DSDS). The DSDS fits into the Surgeon's Assistant's already powerful interface for software maintenance.

The DSDS provides a graphical representation of the relationships of a program's decomposition slices. This is the first system to graphically represent slices in this manner. Based on already existing techniques of software and graph visualization, the DSDS takes these fields to new arenas by incorporating their ideas with program slicing.

Acknowledgments

Daniel Strack and Shreya G. Shah helped in the design and early implementation of the DSDS.

Arthur Delcher gave useful advice concerning the algorithm of the DSDS.

The SEKE referees provided useful comments for the improvement of this paper.

References

[BE94] Thomas Ball and Stephen G. Eick. "Visualizing Program Slices". In *Proceedings of the 10th International Symposium on Visual Languages*, 1994.

[Bin93] D. Binkley. "Slicing in the Presence of Parameter Aliasing". In *Software Engineering Research Forum*, Orlando, FL, November 1993.

[Bro91] M. H. Brown. "Zeus: A System for Algorithm Animation and Multi-View Editing". In *Proceedings of IEEE Workshop on Visual Languages.*, pages 4–9, Kobe, Japan, October 1991. IEEE Computer Society Press.

[DPE92] John Domingue, Blaine A. Price, and Marc Eisenstadt. "A Framework for Describing and Implementing Software Visualization Systems". In *Proceedings* *of Graphics Interface '92*, Vancouver, Canada, May 1992.

[EB88] M. Eisenstadt and M. Brayshaw. "The Transparent Prolog Machine (TPM): an Execution Model and Graphical Debugger for Logic Programming". *Journal of Logic Programming*, 5(4):1–66, 1988.

[EPD] Marc Eisenstadt, Blaine A. Price, and John Domingue. "Software Visualization as a Pedagogical Tool". *Instructional Science*, 21:335–365.

[GKNV92] E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. "Graph Visualization in Software Analysis". In *Second IEEE Symposium on Assessment of Quality Software Development Tools*, pages 226–237, May 1992.

[GL91] Keith Brian Gallagher and James R. Lyle. "Using Program Slicing in Software Maintenance". *IEEE Transactions on Software Engineering*, 7(18):751–761, August 1991.

[HRB90] S. Horwitz, T. Reps, and D. Binkley. "Interprocedural Slicing Using Dependence Graphs". *ACM Transactions on Programming Languages and Systems*, 12(1):35–46, January 1990.

[Int91] Interactive Development Environments, 595 Market Street, 10th Floor, San Francisco, CA 94105. *Software Through Pictures*, 4.2d edition, December 1991.

[KB94] Bradley M. Kuhn and David W. Binkley. "In Search of a Customizable and Uniform User Interface". *Crossroads: The International ACM Student Magazine*, 1(2), December 1994.

[KC] James Arthur Kohl and Thomas L. Casavant. "A Software Engineering, Visualization Methodology for Parallel Processing Systems". Technical report, Parallel Processing Laboratory, Department of Electrical and Computer Engineering, University of Iowa.

[KL90] B. Korel and J. Laski. "Dynamic Slicing of Computer Programs". *Journal of Systems and Software*, pages 198–195, 1990.

[Knu84] D. E. Knuth. "Literate Programming". *The Computer Journal*, 27(2):97–111, 1984.

[Kun94] Thomas Kunz. "Visualizing Abstract Events". Technical Report TI–2/94, Institut für Theoretische Informatik, Technische Hochschule Darmstadt, January 1994.

[LW86] J. R. Lyle and M. D. Weiser. "Experiments in Slicing-Based Debugging Aids". In Elliot Soloway and Sitharama Iyengar, editors, *Empirical Studies of Programmers*. Ablex Publishing Corporation, Norwood, New Jersey, 1986.

[MK88] Hausi A. Müller and Karl Klashinsky. "Rigi—A System for Programming-in-the-large". In *ICSE '10: Proceedings of the 10th International Conference on Software Engineering*, pages 80–86, Raffles City, Singapore, April 1988. IEEE, IEEE Computer Society Press (Order Number 849).

[Ous91] John K. Ousterhout. "An X11 Toolkit Based on the Tcl Language". In *Proceedings of the Winter Conference*. Usenix Association, January 1991.

[PBS93] Blaine A. Price, Ronald M. Baecker, and Ian S. Small. "A Princpled Taxonomy of Software Visualization". *Journal of Visual Languages and Computing*, 4(3), September 1993.

[RDLK90] V. Rajlich, N. Damaskinos, P. Linos, and W. Khorshid. "VIFOR: A Tool for Software Maintenance". *Software—Practice and Experience*, 20(1):67–77, Jan 1990.

[RHSR94] T. Reps, S. Horwitz, M. Sagiv, and G. Rosay. "Speeding Up Slicing". In *Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 11–20, 1994. Published in ACM SIGSOFT Notes v19 n4.

[Wei82] M. Weiser. "Programmers Use Slices When Debugging". *CACM*, 25(7):446–452, July 1982.

[Wei84] M. Weiser. "Program Slicing". *IEEE Transactions on Software Engineering*, 10:352–357, July 1984.