Visuocode: A software development environment that supports spatial navigation and composition

Daniel R. Bradley
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Australia, 4072
Email: daniel.bradley@uq.edu.au

Ian J. Hayes
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Australia, 4072
Email: ian.hayes@itee.uq.edu.au

***Abstract*—Navigating through software is an integral part of software development. Studies have identified that during navigation programmers often become disoriented and lose task awareness. The method-flow visualisation technique displays traversed methods in adjacent editor columns. This paper presents the Visuocode software development environment, which is an implementation of method-flow that, in addition to navigation, supports program composition.**

I. Introduction

Navigating through software is an integral part of software development. Such navigation often follows the software's flow of execution from source file to source file. As an opened source file usually hides the existing editor – either by replacing the current editor, or by opening the source file in a new active tab – programmers can become disoriented and lose task awareness [1].

During a study, Ko et al observed that nearly 34 percent of developers' navigations "seemed to be for the purpose of juxtaposing a set of code fragments" [2]. While, some environments allow source files to be opened in separate windows, or for a tab to be dragged to split the window between two files, this is only workable for a small number of files. Additionally, not all environments support the simultaneous viewing of multiple locations within a single source file. As a result, programmers often switch back and forth between different files or different locations in the one file – a behaviour referred to as *thrashing* [1].

Method-flow is a visualisation technique that allows a programer to navigate the call-graph of a program using adjacent editor columns that are contained within a scrollable "flow-view". Method-calls are represented as clickable hyperlinks that, when clicked, cause just that method to be displayed in a new editor column adjacent to the calling method. The programmer may scroll the flow-view back and forth to see prior methods in that branch of the call-graph. Thus method-flow allows the programmer to navigate based on software structure rather than file structure. We have previously implemented method-flow within the Code Flow software exploration tool.[1]

This paper presents the Visuocode software development environment, which is an implementation of method-flow that, in additional to navigation, also supports program composition. For development, adjacent columns can be used to create new software structures without affecting the existing programming context. For navigation, the programmer may scroll the flow-view to see, or edit, any previously traversed method in the current branch of the call-graph.

The structure of this paper is as follows: Section II summarises background and related research; Section III details the method-flow visualisation technique; Section IV describes the Visuocode programming environment; Section V discusses similarities and differences between Visuocode and related work; Section VI lists the research questions we seek to answer; Section VII briefly outlines future work.

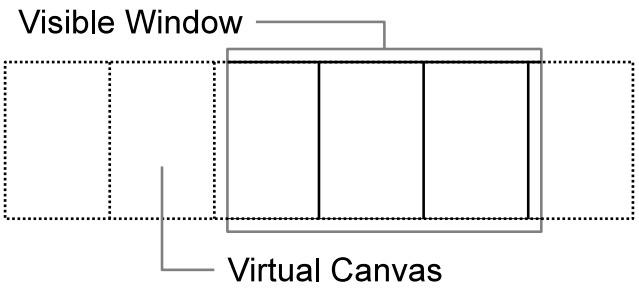

Fig. 1. The method-flow visualisation technique uses an arbitrary number of editor columns, which are contained within a scrollable *flow window*. As the programmer navigates down a branch of the program's call-graph, new editor columns are displayed adjacent to the calling method.

II. Background

Programmer disorientation has been observed during software maintenance tasks on unfamiliar code [2], as well as while browsing class hierarchies for the purpose of code reuse [3]. De Alwis and Murphy describe disorientation as occurring when "developers lose the context or relevancy of their recent actions to their overall goal" [1]. They identified three factors that may lead to disorientation:

1) The absence of connecting navigation context during program exploration;

[1] Available from www.code-flow.com.

2) Thrashing between displays to view necessary pieces of code; and
3) The pursuit of sometimes unrelated subtasks.

This paper focuses on the mitigation of the first two factors – related strategies include: history analysis; working sets; spatial desktops; and inlined code.

A. History analysis

History analysis can provide navigation suggestions based on prior behaviour. These systems seek to reduce disorientation by reducing the number of classes navigated. The NavTracks system [4] creates a model of relationships between files based on prior navigation history then unobtrusively suggests files of immediate interest using a "Related Files" view [5]. Mylar is a related system that uses a degree-of-interest model to drive filtered views that highlight likely task related elements.

B. Working sets

Software navigation is often for the purpose of finding code "fragments" that are related to the current task – this is referred to as the "working set" [6], or alternatively as a "task context" [2]. Environments that support working sets allow the programmer to group arbitrary sections of source files together that are related to a particular task. These systems seek to reduce disorientation by eliminating the need for navigation. Knuth's WEB "literate programming" system might be viewed as a pre-cursor to such environments as it emphasised creating literate source where "concepts have been introduced in an order that is best for human understanding" [7].

The Sheets Hypertext Editor [8] provides similar capabilities through the use of file-like "sheets", which contain linear groupings of arbitrary code "fragments". The Code Bubbles environment [6] provides a virtual workspace upon which methods may be arbitrarily placed (as bubbles) and called methods may be "budded off" adjacent to the calling bubble allowing the programmer to view methods involved in a certain task. As the workspace is many times wider than the display, multiple bubble groups can be arranged for different tasks.

C. Spatial desktops

In mainstream programming environments, navigation is usually achieved by using an auxiliary representation of the class hierarchy. While some environments list the methods available within a class, others require the programmer to scroll through the file to find the method. Apart from the time taken, such navigation also assumes that the programmer can remember the names of the package, class, and desired method.

In contrast, spatial desktop systems provide a map-like layout of source code on a 2D plane so that the programmer can find a method "perceptually rather than cognitively" [9]. These systems seek to reduce disorientation by making navigation quicker and less of a cognitive activity.

Software Terrain Maps [9] presents a map of a code-base that is inspired by a cartographic map. In one manifestation, each method corresponds to a region of a hexagonal grid, where the size of the method's region is proportional to the method's textual size. As method positions will only change as the code-base changes, the programmer may select a desired method by selecting the map area corresponding to the method. To keep the programmer oriented a "vapour trail" could highlight recently accessed methods.

Code Thumbnails [10] and Code Canvas [11] present a more traditional representation in the form of a thumbnail view of the code base. The Code Thumbnails desktop displays each source file as a thumbnail image that can be selected to take the programmer to the indicated file. As the arrangement of thumbnails is spatially consistent, the programmer may make use of spatial memory to find the location desired. Code Canvas [11] extends the Code Thumbnails concept by placing the code thumbnails on a desktop that supports "semantic zoom", which shows "different levels of detail at different levels of zoom" [11].

The previously mentioned Code Bubbles environment [6] may also be considered a spatial desktop as the arrangements of bubbles persist during, and between, programming sessions.

D. Inlined code

Environments that allow the inlining of code allow the programmer to expand the contents of a called method in place of the method-call instead of opening a new editor. This reduces "thrashing" between files as such behaviour is often due to attempting to compare a calling method and a called method simultaneously (see Fluid Source Code Views [12]). These systems seek to reduce disorientation by making navigation unnecessary.

III. The method-flow visualisation technique

'Method-flow' is the name we have given to a novel technique for software navigation. The *method-flow* visualisation technique explicitly supports software navigation and composition by allowing the programmer to traverse a method call-graph using adjacent editor columns.

In each editor column, methods are represented using individual text editors and method- calls are presented as hyperlinks within each editor. If a hyperlink is followed, a new editor column is stacked to the right of the calling column. As more text columns are added, the leftmost columns scroll off the left side of the screen.

Although only a limited number of editor columns can be seen, the use of a horizontally scrollable *flow-view* allows an arbitrary number of editor columns to be added (see Fig. 1). Because the window content can be easily scrolled, editor columns can be easily brought back into view to allow the programmer to follow the flow of execution.

The benefit of this technique is that a programmer can explore down a branch of the call-graph without destroying their initial programming context. At any time, the programmer is able to scroll back to the left to refresh their memory regarding the context of the calling methods. As the methods are consistently placed on a scrolling plane, we believe that method-flow is able to leverage the use of spatial memory.

Fig. 2. Visuocode consists of two types of window: the *workspace manager* (left), and the *flow window* (right). The workspace manager is used to manage projects and set the left-most editor column of the flow window. The flow window implements method-flow by allowing an arbitrary number of editor columns to be contained within a scrollable *flow view*. In the image above, the programmer selected the *nextToken* method, then navigated to the *parseNextToken* method, the *parseWord* method, then finally the *isLetter* method (see Fig. 3 for a closer look).

IV. Visuocode

Method-flow has now been implemented as a software development environment called Visuocode, which supports the Java programming language.[2]

The programming environment consists of two types of window: the *workspace manager* and associated *flow windows*. This terminology was chosen to be analogous to terminology used by other programming environments.

A. The workspace manager

The workspace manager (see Fig. 2) allows the programmer to manage a collection of projects. When a project is added to the workspace manager its Java source files are parsed and a tree representing its packages, classes, and methods, is added beneath the "Workspace Projects" node. Selecting either a class name or a method name will replace all existing editor columns displayed within the flow-window with an editor column corresponding to the selected method or class.

B. The flow window

The *flow window* (see Fig. 2) contains a horizontally scrolling view that contains one or more editor columns. Initially, the flow window contains a single column editor corresponding to a method or class that was selected from the workspace manager.

Each editor column is associated with a specific class and contains a *class attributes* area and a *method editors* area. The class attributes area contains an editable class definition field, as well as three editable field lists that allow imports, enumerations, and class members to be altered. The method editors area contains a vertically scrollable stack of one or more method editors. If the column editor was opened by clicking a class name in the workspace manager, there will be a method editor for each of the class's methods ordered alphabetically.

1) Support for navigation: Within method editors, resolving classes are represented as crimson hyperlinks. Resolving method-calls are represented as blue hyperlinks if the corresponding source code is available; otherwise (if a system method) in a dark grey colour. If a hyperlink is clicked, an editor column for that class is inserted to the right of the

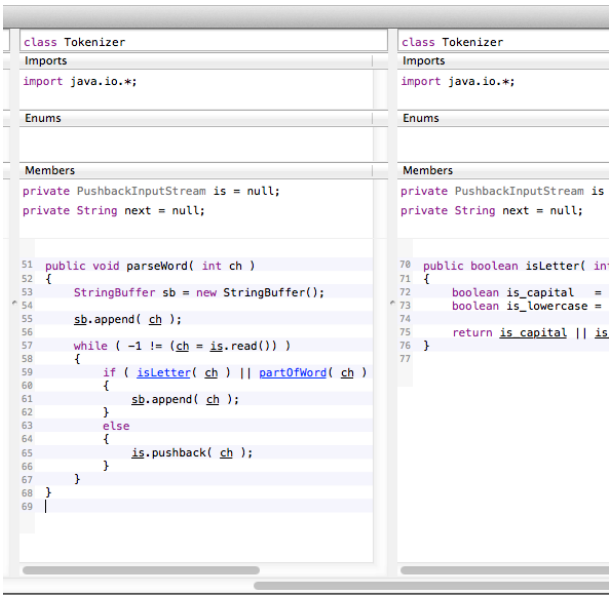

Fig. 3. Each editor column contains a *class attributes* area, which allows the class definition, imports, enumerations, and members to be edited.

[2]A recent development version is available from www.visuocode.com.

existing column. If the hyperlink was a class type, all the methods of that class will be presented alphabetically within the method editors area; otherwise just the called method.

2) Support for software creation: Method-calls that fail to resolve are represented as red hyperlinks. If clicked, the method-call is analysed to determine in which class the new method should be inserted and an editor column containing a method skeleton is displayed. Where a non-existent method is invoked upon a specific object, it would be inserted into the class corresponding to that object.

Non-resolving class types are also presented as red hyperlinks that, when clicked, present a simple dialog to quickly create the class. Once created, the class type may be clicked to reveal a corresponding editor column.

V. Discussion

Visuocode seeks to reduce disorientation by integrating navigation, comprehension, and modification, into the same user interface. Unlike *history analysis*, *working set*, and *spatial desktop* systems, Visuocode currently requires the programmer locate their intended navigation starting point using the *workspace manager* – a traditional package hierarchy representation. Like *working set* and *inlined code* systems, Visuocode can display a grouping of methods as long as they exist on the same branch of the call-graph.

A benefit of the Code Bubbles [6] approach is that arbitrary methods can be arranged so that complex interrelationships can be better understood, however desktop space scarcity limits the number of method "bubbles" that can be displayed. A benefit of Visuocode approach is that it allows the programmer to quickly perform a localised search of the call-graph until an appropriate branch, for the current task, is found.

A benefit of the Fluid Source Code Views [12] approach is that less horizontal screen space is used, while the benefit of the Visuocode approach is that it preserves the visual integrity of the calling method.

VI. Research questions

How can spatial memory be leveraged during programming?

Spatial desktop systems and method-flow both seek to leverage spatial memory, but in very different ways. Method-flow utilises spatial memory by presenting a spatially consistent representation of a branch of a call-graph that is intended to support working memory. Spatial desktops provide a spatially consistent representation of the code base that both supports working memory and long-term memory. How would the use of both systems at once interfere with each other?

Do alternative-paradigm software development environments affect software structure?

Visuocode and Code Bubbles [6] both present a novel paradigm for the composition of software that allows the easy creation of new methods juxtaposed to an existing programming context. We believe that this allows programmers to compose better structured code due to the reduction in mental and time overheads related to creating new classes and/or methods.

How do software composition and modification differ?

The majority of published empirical studies of programming ask participants to perform software modification tasks because of the time constraints involved with running a study. Unfortunately, this means that there is a little data on the affect of novel software development environments on software that is created 'from scratch'.

VII. Future work

This paper has presented the Visuocode software development environment, which implements the method-flow visualisation technique. We plan to evaluate the environment using a mixed qualitative/quantitative observational study. In the future, we will extend the system to better support the representation of method-calls to superclass methods, as well as add support for reverse navigation up the call-graph.

References

[1] B. de Alwis and G. Murphy, "Using visual momentum to explain disorientation in the Eclipse IDE," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, September 2006, pp. 51–54.

[2] A. Ko, B. Myers, M. Coblenz, and H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *Software Engineering, IEEE Transactions on*, vol. 32, no. 12, pp. 971–987, 2006.

[3] R. K. Bellamy and J. M. Carroll, "Re-structuring the programmer's task," *International Journal of Man-Machine Studies*, vol. 37, no. 4, pp. 503–527, 1992.

[4] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for ides," in *Proceedings of the 4th international conference on Aspect-oriented software development*, ser. AOSD '05. New York, NY, USA: ACM, 2005, pp. 159–168. [Online]. Available: http://doi.acm.org/10.1145/1052898.1052912

[5] J. Singer, R. Elves, and M. Storey, "NavTracks: Supporting navigation in software maintenance," in *ICSM'05. Proceedings of the 21st IEEE International Conference on Software Maintenance*, 2005, pp. 325–334.

[6] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr., "Code bubbles: A working set-based interface for code understanding and maintenance," in *Proceedings of the 28th International Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2010, pp. 2503–2512.

[7] D. E. Knuth, "Literate programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984.

[8] R. Stockton and N. Kramer, "The sheets hypercode editor," Department of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-02-80, 1997.

[9] R. DeLine, "Staying oriented with software terrain maps," in *International Conference on Distributed Multimedia Systems*, 2005, pp. 309–314.

[10] R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. Drucker, and G. Robertson, "Code thumbnails: Using spatial memory to navigate source code," in *IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 2006, pp. 11–18.

[11] R. DeLine and K. Rowan, "Code canvas: zooming towards better development environments," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 207–210. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810331

[12] M. Desmond, M. Storey, and C. Exton, "Fluid Source Code Views," in *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*. IEEE Computer Society Washington, DC, USA, 2006, pp. 260–263.