# NavClus: A Graphical Recommender for Assisting Code Exploration

Seonah Lee*, Sungwon Kang*, Matt Staats†

*Department of Computer Science
KAIST
Daejeon, Republic of Korea
{saleese|kang}@cs.kaist.ac.kr

†Division of Web Science and Technology
KAIST
Daejeon, Republic of Korea
staatsm@kaist.ac.kr

***Abstract*—Recently, several graphical tools have been proposed to help developers avoid becoming disoriented when working with large software projects. These tools visualize the locations that developers have visited, allowing them to quickly recall where they have already visited. However, developers also spend a significant amount of time exploring source locations to visit, which is a task that is not currently supported by existing tools. In this work, we propose a graphical code recommender *NavClus*, which helps developers find relevant, unexplored source locations to visit. NavClus operates by mining a developer's daily interaction traces, comparing the developer's current working context with previously seen contexts, and then predicting relevant source locations to visit. These locations are displayed graphically along with the already explored locations in a class diagram. As a result, with NavClus developers can quickly find, reach, and focus on source locations relevant to their working contexts. http://www.youtube.com/watch?v=rbrc5ERyWjQ**

## I. Introduction

Developers, as humans, occasionally forget what they have done and plan to do. On a daily basis, developers perform multiple tasks, with interruption being an unavoidable part of software development. In order to understand and modify large, complex software systems, they must visit many source locations. While visiting these locations, two problems can occur. First, as the number of visited locations increases, developers may struggle to recall where they have already visited, and thus become disoriented in the code base [1]. Second, developers generally cannot remember the detailed structure of complex code bases, and must actively explore to understand how to modify the system, a task that can be quite challenging [2].

To assist developers confronted with these problems, several graphical tools have been developed to visualize the locations developers have visited in the code base. For example, Relo visualizes methods and fields visited by a developer in a class diagram [3]. Similarly, Code Bubble visualizes the code fragments and documents visited by a developer [4]. However, these tools only address the first problem, developer disorientation [1]: these tools remind developers where they have visited, but cannot suggest unvisited locations to explore further. NavTracks suggests files to visit, but limits its visualization to showing clusters of visited files [5].

In this paper, we propose a new graphical code recommender to visualize source locations that developers have already visited, as well as suggest source locations that developers are likely to find relevant. This tool addresses the second problem [2] by suggesting unvisited locations for further exploration. It is based on our previous work on the technique that mines developer interaction traces, identifies the developer's working context, and uses this information to recommend other relevant source locations [6]. The demonstrated tool *NavClus* implements this recommendation technique with the extension of a graphical user interface based on class diagrams with the goal of improving the usability of our recommendations [6][7].

To the best of our knowledge, the NavClus tool is the first tool to visualize source location recommendations in a class diagram, and the first to both visualize already visited source locations and recommended source locations. We believe the addition of recommendations in a class diagram will greatly facilitate developers' code exploration activities.

This paper is organized as follows. Section II presents an example use case for NavClus. Section III describes the architecture of NavClus. Section IV explains our evaluation plan, and Section V discusses how this work can be extended with additional functionality. Section VI concludes this paper.

## II. Example Use Case

Let us suppose that Jim, a developer, is working on the JHotDraw 6.01 editor. Jim is trying to fix a menu in the editor which fails to draw the correct arrow tip[1]. Jim begins by opening the file that includes the main function, and searches for possibly relevant source locations in a top-down fashion. He eventually discovers the method "`createArrowMenu()`", but does not understand how to modify it, as the method consists of over ten other methods scattered across several files. Accordingly, Jim navigates over ten paths through the source code to determine which methods should be edited to correct the bug. During this extensive exploration process, Jim becomes disoriented, and forgets which source code locations he has investigated. Furthermore, Jim must continually identify

[1]This example is derived from a programming task outlined in [8].

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Formal Demonstrations

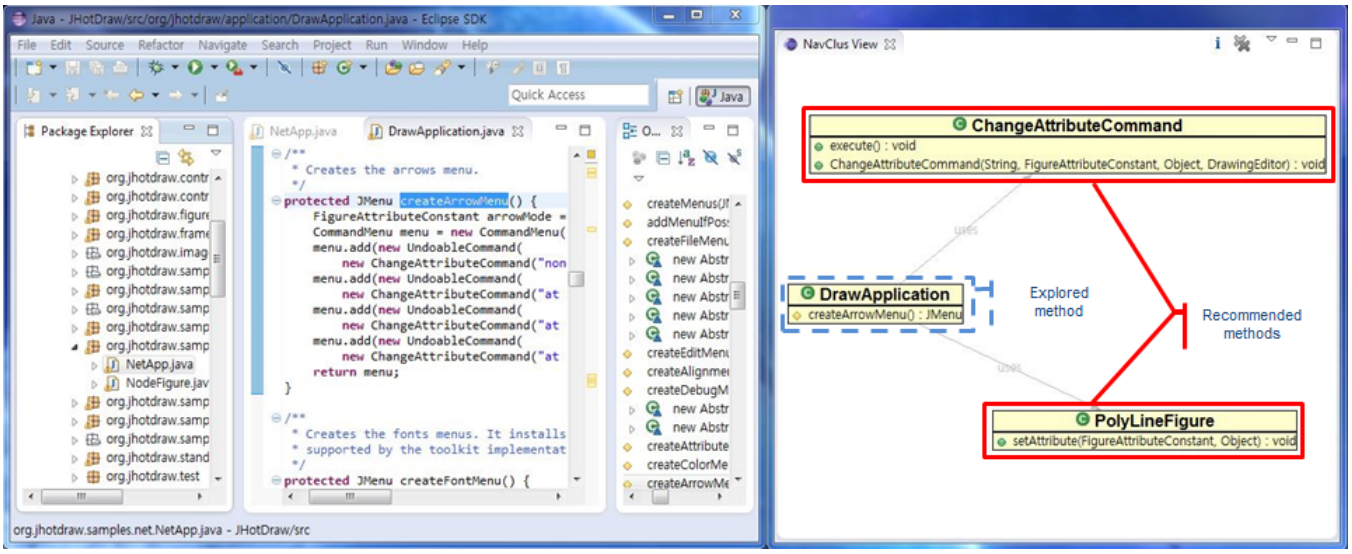


Fig. 1. NavClus User Interface

new source code locations to explore, a challenging process given the large number of candidate methods.

NavClus is designed to help Jim both in coping with disorientation during exploration [1] and in identifying new source locations to explore [2]. To reduce disorientation, NavClus incrementally displays where the developer has visited. For example, as Jim explores beyond `DrawApplication.createArrowMenu()`, this method is shown in a class diagram, with relationships to newly visited classes and methods displayed. This was done by a previous tool Relo [3], and we have adopted their approach for helping developers understand how their current source locations relates to already visited ones.

To aid in identifying new source locations, NavClus recommends relevant but unexplored methods. For example, when Jim visits `DrawApplication.createArrowMenu()`, other relevant methods to visit such as `PolyLineFigure.setAttribute()` and `ChangeAttributeCommand.execute()` are recommended in a class diagram, as shown in Figure 1. By referring to these locations, Jim can easily determine the source locations to visit and examine.

To the best of our knowledge, NavClus is the first tool to implement this function in a class diagram. By using the same format for both displaying explored source code locations and for recommending new locations to visit, we believe developers can more easily explore the code base.

## III. NavClus Overview

NavClus is built around the notion of *navigation contexts*, which roughly corresponds to a network of method views and edits [6]. This approach is implemented as an extension to the Eclipse IDE for easy use by developers. Figure 1 shows the user interface of NavClus, which can be separated from the Eclipse IDE. This separation helps a developer who uses multiple monitors be able to refer to the diagram in the side monitor, while performing coding tasks in the main monitor. Within this interface, NavClus performs five core functions:

- **Display history**: As a developer navigates the code base, the methods and classes already explored incrementally appear in a class diagram.
- **Create and display recommendations**: Based on the current working context and previously seen working contexts, recommendations for additional methods to explore are presented in the class diagram.
- **Update diagram layout**: Developers can manually rearrange the class diagram via the mouse, or can let NavClus automatically update the layout for readability.
- **Jump to source locations**: Previously visited or recommended source locations can be visited by double clicking on the methods in the class diagram.
- **Collecting interaction traces**: As a developer visits and edits methods and classes, the developer's actions are recorded as interaction traces. This information is used to improve later recommendations.

Figure 2 shows the architecture of NavClus, which consists of four components: the user interface, the mining engine, the recommendation engine, and the data collector.

### A. User Interface

The user interface monitors the developer's actions and updates the displayed class diagram, showing two types of locations. First, the interface displays source locations that have been visited or edited by the developer, providing an at-a-glance record. Second, the interface displays source locations that may be relevant to the developer's current task, with the goal of reducing the effort and time needed to complete the task. Both previously visited source locations and recommended source locations can be visited by double clicking on the appropriate graphical element. The user interface consists of two components: the *monitor* and the *visualizer*.

- **Monitor:** Keeps track of developer actions such as visits and edits, and creates the records of these actions. The monitor was implemented using *Mylyn*[2].
- **Visualizer:** Receives source locations as input and and presents them in a class diagram. The visualizer was implemented using *zest* and *draw2d* libraries of the *GEF*[3].

Note that NavClus currently has two separate modes of visualization: displaying history, and displaying recommendations. History is displayed in scenarios where recommendations cannot be made (i.e., when not enough data has been collected). During recommendation already explored locations are not suggested. This separation was done to avoid overly cluttering the diagram and confusing the developer.

### B. Mining Engine

The mining engine mines contextual collections of source code locations in interaction traces [6]. Essentially, the mining engine performs clustering over sets of source locations, represented as methods [6]. This set is passed to the collection database as mined collections. Three steps are performed during mining:

1) **Sequence Generator:** Creates short navigation sequences using a developer's interaction trace. The trace is cut when a developer revisits a method, resulting in a short navigation sequence.
2) **Micro-cluster Generator:** Collects navigation sequences beginning with the same method. As each sequence starts with the same method, this results in collections of code that have elements connected via developer navigation actions, so-called micro-clusters.
3) **Macro-cluster Generator:** Clusters similar collections of code using a cosine similar metric, which produces a similarity value between zero and one [9]. Two collections of code are assumed to be similar if they show higher than 0.5 cosine similarity value. The resulting collections of code are called macro-clusters. To create a macro-cluster, the k-nearest neighbour clustering algorithm is used [9]. In this algorithm, each micro-cluster is merged with the first similar macro-cluster in a comparison loop. If there is no similar macro-cluster, the micro-cluster becomes a macro-cluster.

### C. Recommendation Engine

The recommendation engine recommends locations to visit after a developer visits several locations [6]. The recommendation engine selects the set of locations which best matches the developer's current context. This set is passed to the user interface as a recommendation. Two steps are performed during recommendation:

1) **Context Former:** Forms a query after a developer has performed three actions (visiting or editing source code). The query represents the developer's current context. The query is then passed to the next step.
2) **Collection Selector:** Retrieves the results matching the query. The best result is passed to the user interface.

[2]http://www.eclipse.org/mylyn/
[3]http://www.eclipse.org/gef/

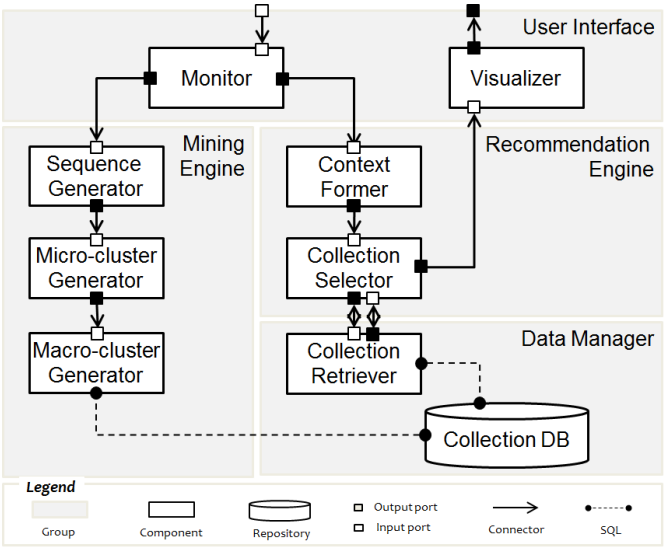


Fig. 2. NavClus Architecture

### D. Data Manager

Collections of code are kept for later recommendations. The data manager consists of two components: Collection Retriever and Collection DB.

- **Collection Retriever:** Retrieves the macro-clusters that have the highest similarity value when being compared with the query. To best judge the similarity of macro-clusters with a few locations of the query, the TF/IDF similarity metric is used [9]. For performance, an inverted index algorithm is used when accumulating and retrieving macro-clusters [10].
- **Collection DB:** Stores macro-clusters, updating whenever user actions occur. We intend to implement this as a database eventually, but the current prototype uses history files mined during NavClus startup.

## IV. Evaluation Plan

To evaluate and understand NavClus's effectiveness, we intend to explore two questions:

1) Can the recommendation of source locations help a developer navigate the code base more effectively?
2) How many days of development are required before the initial recommendations can be made?

To answer these questions, we have planned a diary study. In a diary study, subjects are asked to keep a long-term record of their everyday experiences, from which researchers can infer underlying patterns [11].

We plan to recruit ten or more subjects for our diary study. The subjects will be required to develop their projects in the Java language using the Eclipse IDE in conjunction with NavClus. Three phases are currently planned.

In the first phase, *preparation*, the subjects will be asked to list background information, such as their development histories and information related to their projects. The subjects will then be asked to install NavClus and a screen capture program.

In the second phase, *reporting*, the subjects will be asked to write a daily diary report. In each diary report, they will be asked to describe situations in which they select to use NavClus, to describe NavClus recommendations they found useful or interesting, and to provide open-ended, honest feedback on NavClus's effectiveness and usefulness. The subjects will be asked to send more than seven diary reports within 3 weeks.

In the final phase, the subjects will be interviewed. Each subject will be asked to numerically evaluate NavClus (1-5) and to describe about any past experience dealing with diagramming tools and experience with NavClus.

By conducting this diary study, we intend to determine the effectiveness of NavClus in daily routine development, both quantitatively and qualitatively. In particular, we hope to determine which aspects of NavClus are preferred by users: the functionality for tracking where the developer has visited, the functionality for recommending new source locations to visit, or both.

## V. Future Work

NavClus is focused on improving individual developers' ability to explore and understand large code bases, and our evaluation reflects this. However, we believe NavClus can also improve software development teams' ability to understand their code bases. In particular, we believe we can improve the current NavClus system by reducing the time required for new team members to understand the code base, and by using data from developers simultaneously working on the code base.

### A. Contextual Knowledge Transfer

Software development teams often experience turnover, with new developers struggling to understand an unfamiliar, complex code base. We are interested if developer interaction histories from more experienced developers can be used by new developers to reduce the spinup time when joining a project, allowing new developers to accomplish tasks with less effort than if they had used their own interaction histories. Evaluating this, however, requires both an extension of NavClus and a substantially more complex evaluation, and thus we leave it for future work.

### B. Collaborative Comprehension

A software development team generally consists of two or more developers, who collaborate together. From the perspective of computer supported cooperative work / collaborative filtering research, NavClus is an attempt to resolve the *cold start problem* of addressing who will be the first to provide the data that others will use. When developing systems like NavClus, information providers should receive a direct benefit, as otherwise they will be reluctant to invest effort into the system [12]. We achieve this by focusing benefiting individual developers using NavClus. In later work, we plan to consider how multiple programmers' interaction traces can be best combined to improve the productivity for multiple developers.

## VI. Conclusion

In this work, we presented a graphical code recommender, *NavClus*, for reducing disorientation when exploring large code bases and for recommending additional locations to visit. This approach is based on automatically visualizing source code locations, both visited and recommended, in a class diagram based on the developer's current and past working contexts. We believe that presenting this information alongside the source code editor, will reduce the mental effort required to understand the task at hand. As this approach is completely automated, no additional burden is placed upon the developers.

We plan to evaluate our approach in a realistic development environment using a diary study. This study is intended to help us better understand how NavClus helps developers navigate large code bases, and how it can be further improved. In future work, we will also explore how the effort of several developers working on the same project can be best aggregated to improve source recommendations for all developers involved.

## Acknowledgment

This research was funded by the Ministry of Knowledge Economy (I2001-12-1095), and supported by the World Class University program (R31-30007) and the Basic Science Research Program (2012-0007069) funded by the Ministry of Education, Science and Technology, Republic of Korea.

## References

[1] B. D. Alwis and G. C. Murphy, "Using Visual Momentum to Explain Disorientation in the Eclipse IDE," in *IEEE Symposium on Visual Languages and Human Centric Computing*, 2006.

[2] T. D. LaToza and B. A. Myers, "Developers ask reachability questions," in *Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering*, 2010.

[3] V. Sinha, D. Karger, and R. Miller, "Relo: Helping users manage context during interactive exploratory visualization of large codebases," in *Visual Languages and Human-Centric Computing*, 2006.

[4] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr., "Code bubbles: rethinking the user interface paradigm of integrated development environments," in *Proc' of the 32nd ACM/IEEE Int'l Conf. on Software Engineering*, 2010.

[5] J. Singer, R. Elves, and M.-A. Storey, "Navtracks: Supporting navigation in software," in *Proceedings of the 13th International Workshop on Program Comprehension*, 2005.

[6] S. Lee and S. Kang, "Clustering and recommending collections of code relevant to tasks," in *27th IEEE Int'l Conf. on Software Maintenance*, 2011.

[7] ——, "A study on guiding programmers code navigation with a graphical code recommender," *Studies in Computational Intelligence*, vol. 377, pp. 61–75, 2012.

[8] I. Safer and G. C. Murphy, "Comparing episodic and semantic interfaces for task boundary identification," in *Proc. of the 2007 Conf. of the Center for Advanced Studies on Collaborative research*, 2007.

[9] J. Han and M. Kamber, "Data mining : Concepts and techniques," *Techniques*, 2006.

[10] D. A. Grossman and O. Frieder, *Information Retrieval: Algorithms and Heuristics*, 2004, vol. 461.

[11] W. Terry, "Everyday forgetting: Data from a diary study," *Psychological reports*, vol. 62, no. 1, 1988.

[12] J. Grudin, "Groupware and social dynamics: eight challenges for developers," *Communications of the ACM*, vol. 37, no. 1, 1994.