

Task Board: Tracking Pertinent Task Artifacts and Plans

Chris Parnin, Carsten Görg, Spencer Rugaber
College of Computing, Georgia Institute of Technology
chris.parnin@gatech.edu, {goerg,spencer}@cc.gatech.edu

Abstract

Developers must actively maintain plans and goals about their programming tasks. Unfortunately, much of this knowledge about tasks does not exist in a tangible form. Commonly, developers must recover the details of this knowledge after encountering an unanticipated interruption. In this paper, we present a technique for allowing developers to quickly reengage in a task after an interruption. A dashboard visualization of intermediate task knowledge is prepared from recent interactions and program executions and then displayed to assist in restoring task knowledge. Additionally, developers can annotate the artifacts with task sketches and prospective cues.

1. Introduction

Development tasks typically require coordinating software changes across multiple locations in a program's source code. Integrated development environments (IDEs) have traditionally provided limited support for managing the active artifacts relevant to the programming task. This includes tabbed editors for editing multiple files, hierarchical file lists that can be collapsed and expanded, forward/backward navigation commands to return to previous locations, and a list of recently visited files.

Unfortunately, there are problems with these interfaces. In studies of developer navigation histories, a common finding is that developers frequently visit many locations in rapid succession in a phenomenon known as *navigation jitter* [3]. Navigation jitter has been commonly attributed to developers flipping through open tabs and file lists when trying to recall a location [3, 2]. To alleviate this problem, researchers have proposed various alternatives. Relo [4] displays a complementary UML view that is built from recently visited source code. Another system, SHriMP [5] uses a focus-and-context paradigm to allow multiple code locations to be viewed simultaneously. Despite these improvements, a problem exists with these ways of viewing and navigating code: items in file lists and tabbed windows

are spatially instable forcing developers to rely on symbolic names or trial-and-error to locate relevant artifacts. Code Thumbnails instead proposes using spatial memory in a peripheral view to assist navigation [1].

Although these systems improve to some extent the representations of relevant task artifacts, they do not expand the scope of task artifacts. Second, when developers perform a task, they often need to refine that task into several subtasks, perform tangential investigations, and track and evaluate their progress. Current systems represent tasks monolithically instead of representing the distinct stages and distinct sets of task artifacts. Finally, these systems do not have a way of supporting task planning. In this paper, we present a tool called *Task Board* to address those limitations.

2. Task Board

We have developed TASKBOARD as a Visual Studio plugin for the purpose of managing access to artifacts referenced in a programming task and its subtasks. In TASKBOARD, a set of active task items are displayed on a peripheral monitor. The task items include source code, runtime program values, and a list of code locations. As the programmer actively develops, these task items maintain visual cues to assist the developer in getting an overview of the artifacts, recent activity, and progress. Further, these items can be annotated with task planning information and prospective cues.

To use TASKBOARD, a developer begins planning how they want to perform a task. Using a process we call *trailblazing*, developers can investigate the code to evaluate and plan the proposed changes. During this time, the developer may add a task artifact to TASKBOARD. Developers have several choices in the types of artifacts they can add:

- code files,
- results of a query such as *find references* or *search*,
- runtime values from a debugging session, and
- prospective notes and implementation plans.

When an artifact is in TASKBOARD, a developer may then associate the item or group of items with a subtask.

Visual cues are given to the artifacts to note the planned sequence of steps and they can be used to filter which artifacts appear or to adjust the level of detail. For example, elements belonging to the subtask following the current subtask would be smaller and muted. In Figure 1, an example of artifacts and annotations is shown. As seen in the visualization, code files are represented with a Code Thumbnail view [1] that can be dynamically adjusted in size. Further, code files are annotated with two visual cues indicating recent editing (green) and navigation activity (blue).

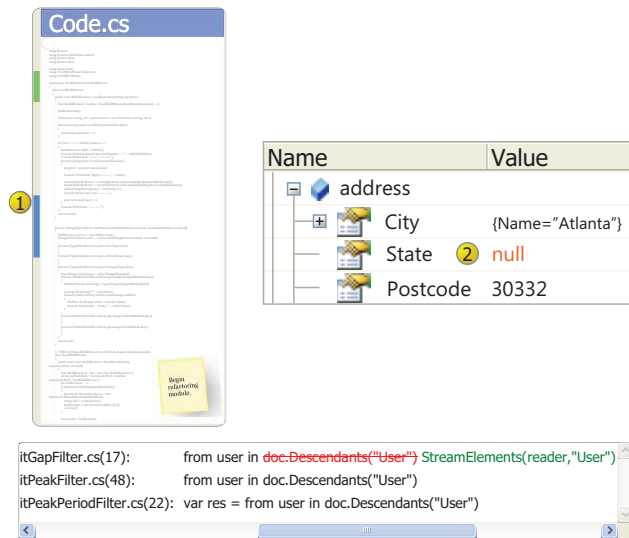


Figure 1. Three task artifacts are displayed: a code file, runtime value, and search results. In addition, the artifacts are annotated with subtask sequences and a prospective note.

2.1. Examples

In this subsection, we give three examples of how TASKBOARD can be used to solve common problems with performing a programming task.

Developers commonly use the list of search results in an IDE to track the locations where changes need to be made. Two problems developers face when using this approach are that the search results are not easily persisted, and it is difficult to evaluate whether the change was made successfully in all places. To address these two concerns with TASKBOARD, first, search results can be persisted as a task artifact. Second, we take a snapshot of the code when the query was made and display differences next to the original line of code. The differences allow a developer to evaluate if the change was effectively made and to then optionally mark it as completed.

When developers test their intended changes, they may find some unexpected errors. Typically, developers debug the program to investigate the cause of failures. After the developer has identified a potential cause for the failures,

they end the debugging session and attempt to make a fix. Unfortunately, the runtime values that were instrumental in diagnosing the problem are not persisted after or between multiple debugging sessions. In TASKBOARD, these values can be persisted and associated with the class they were collected from.

Finally, failure to remember prospective tasks is a common result of interruption. A popular tactic to remember prospective tasks is to leave a *TODO* comment as a reminder. However, these notes are often used to mark possible design flaws or suggestions for future preventive maintenance. To allow developers to specify temporary notes in the context of the current task, we support the specification of implementation plans. An implementation plan is a memory tactic that uses a conditional cue to activate a reminder. For example, a developer could specify the following conditional cue and reminder: If `UnitTest` available for class `=>begin refactoring module.`

3. Conclusion

We have presented a system for managing task artifacts, planning subtasks, and evaluating task progress. In future work, we would like to expand and improve our proposed features. In addition, we are interested in conducting experiments that measure how developers access the task artifacts and which visual cues prove the most helpful.

References

- [1] R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. Drucker, and G. Robertson. Code thumbnails: Using spatial memory to navigate source code. In *Proceedings of the Visual Languages and Human-Centric Computing*, pages 11–18, 2006.
- [2] C. Parnin and C. Görg. Building usage contexts during program comprehension. In *ICPC '06: Proceedings of the 14th IEEE International Conference on Program Comprehension*, pages 13–22, 2006.
- [3] J. Singer, R. Elves, and M.-A. Storey. Navtracks: Supporting navigation in software maintenance. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 325–334, 2005.
- [4] V. Sinha, D. Karger, and R. Miller. Relo: Helping users manage context during interactive exploratory visualization of large codebases. In *Proceedings of the Visual Languages and Human-Centric Computing*, pages 187–194, 2006.
- [5] M.-A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu, and M. Musen. Shrimp views: an interactive environment for information visualization and navigation. In *CHI '02 extended abstracts on Human factors in computing systems*, pages 520–521, 2002.