

Reconstructing and Visualizing Programmer Activities

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

Abstract

Task management is an essential skill for developers to manage the many responsibilities they are assigned during software development. Developers must be able to suspend and later resume work on a programming task in the event of an interruption or the need to switch to a different task. However, little tool support is available for developers needing to recall details relevant to an interrupted task. In this paper, we present a technique that assists developers in reconstructing the details of activities performed during development work. Our technique provides a timeline visualization of recent development sessions, significant events, shifts in attention, and summaries of activities performed during a period of time. From these visualizations, developers can recover a task context from the code elements used during a selected time range.

1 Introduction

Task management is an essential aspect of software development. Completing, documenting, switching, and tracking tasks are activities that are expected to be performed by developers on a daily basis. But, as a developer's time becomes fragmented by interruptions and performing different tasks, the capability to manage those tasks degrades.

Making task management a first class citizen in both the development process and development tools could alleviate these problems. Kersten and Murphy proposed capturing development activities in a *task repository* [8]. A task repository stores *tasks* which are units of assigned development work. Additionally, each task can be associated with a *task context* [8] using tools such as Mylyn*. In practice, task repositories are often realized via an issue tracking system in which tasks exist alongside reported bugs and change requests.

However, several problems still exist. The scope of a task context is primarily limited to the files and methods accessed during the performance of the task. It is unclear to what extent a task context, as currently portrayed, supports developers in resuming or summarizing a task. Certainly in current systems, a task context does not include plans, solutions, events, unresolved issues, or the actual changes made to method bodies. When developers perform a task, they often need to refine it into several steps, perform tangential investigations and track and evaluate progress. Upon resuming a suspended task, developers need to reconstruct this knowledge. Additionally, the accuracy of developers self-reporting activities is doubtful: self-reporting was observed to be an unreliable method of reporting activities such as refactoring [10] or bug fixing [1]. Finally, we suspect developers may prefer informal methods of tracking and managing tasks as it has been observed that developers place informal task annotations in comments [19]. Developers may need to perform exploratory tasks or handle details not originally specified in an assigned task but often fail to reflect these unspecified tasks in a task repository. Indeed, in previous work, we observed less than 1% of over 10,000 Mylyn users used commands for new subtasks or new local tasks which are not stored in the team task repository [14].

In this paper, we address problems related to recalling details about past work activities that are often needed when a developer resumes an interrupted task or summarizes a completed task. These problems become even more difficult to address because developers may not have performed the work with a complete task specification that can be linked to the activities. We introduce an activity timeline that provides an overview of work performed for a task and helps identifying different activities performed – these activities can be then be linked to a task specification.

We make the following contributions:

- A visualization for summarizing and highlighting important events during development sessions.
- An interface for extracting task contexts from a time window.

*www.eclipse.org/mylyn

- A tool for supporting developers with task management even with incomplete task specifications.

The benefit of this approach is that developers will have better support for transcribing and controlling how they develop software. Although we have not yet formally evaluated the tool, we believe it can provide a baseline for exploring alternative designs and spur interest in this problem.

In the next section, we describe related work. In Section 3, we discuss concepts underlying interaction histories. In Section 4, we present details about our approach to reconstruct and visualize activities. Finally, we conclude with ideas about future work and experiments.

2 Related Work

Researchers have proposed several approaches for understanding and visualizing human activity in software development. Initial work focused on the aspects of activity in source code repositories and alleviating the problems associated with coordinating development between teams [20, 2] and has been shown to reduce conflicts [18]. Similarly, Ellis and colleagues have visualized bug repositories in support of prioritizing development tasks [5, 4].

More recent work has shifted the focus to include other aspects of the development process [12], including awareness of personal plans and activities. *Spyware* [15] collects developer interaction history and provides an interface for visualizing and exploring development sessions. The visualization provides a good overview for characterizing development sessions, but is not designed to provide more in depth information that would be necessary for when reconstructing development activity.

Researchers have also worked on approaches to better identify the tasks corresponding to activities within development sessions. Coman and Sillitti have proposed methods of automatically identifying switches between tasks [3] in a development session. Alternatively, other researchers have designed interfaces for developers to identify the task boundaries within a development session. Specifically, Safer and Murphy [17] investigated various designs of user interfaces for recalling information about recent tasks. In their study, the user identified when a recent task was completed or deferred for another task. One interface displayed a timeline of screenshots capturing different stages of performing the task. The other used a timeline of a filtered view indicating the files and methods visited or edited. An overall subjective preference was found for the screenshots, although the results overall remain inconclusive. In a similar vein, a later version of *Spyware* introduced a mechanism for replaying a development session [16].

Although the previous approaches have provided strong guidance toward reconstructing activities, several issues re-

main. Automated detection of task shifts can be problematic because of the difficulty in determining which items in a context belong together. Unfortunately, in our experience, this can change widely with tasks [13]. Concerning the benefit of instant replay, psychology researchers have doubted its effectiveness for resuming interrupted activities and cite an overall negative effect [7]. Developers may have preferred the screenshots over the task context view because it contained more details about the source code. Indeed, although a screenshot can be a rich source of cues that trigger recall of performed activities, choosing the granularity of screenshots every few minutes may impose an interface that is too fine-grained and too difficult to form a collective image of what took place — especially for a development session lasting a few hours. Furthermore, the interface based on screenshots is not well-suited toward extensions such as aggregation. Conversely, an interface displaying the file names and method names modified and visited within a small segment of time provides too little context to properly identify the activities that occurred. Identifying the activities from only a method name can be difficult because many actions may have taken place within the context of a method, but those details are not exposed nor easily distinguishable. Finally, most of the visualization approaches require the developer to individually view each atomic change in the development history instead of providing a method for getting an overview.

In this paper, we allow developers to have a broader view of their activity timeline. Instead of using a strictly automated approach, we identify and provide visual indications of shifts in the items that a developer can consult. Instead of using a replay mechanism, we focus on providing better summarization of development activity through an information mural [6] inspired timeline. To enable episodic recall of activities, we mark events on the timeline. To provide a better overview of the code content we use a launchpad to display links to code symbols within method bodies. Additionally, we support incorporating any task structure that may exist and then elaborating its definition and linking it to the activities that occurred.

3 Development Activity

In this section, we discuss how we capture and record development activities. We also describe other information collected and representations formed from these recorded activities.

3.1 Interaction and Change History

In order to visualize and reconstruct recent development activities, it is necessary to first capture and record this activity. In general, this activity is referred to as the *interac-*

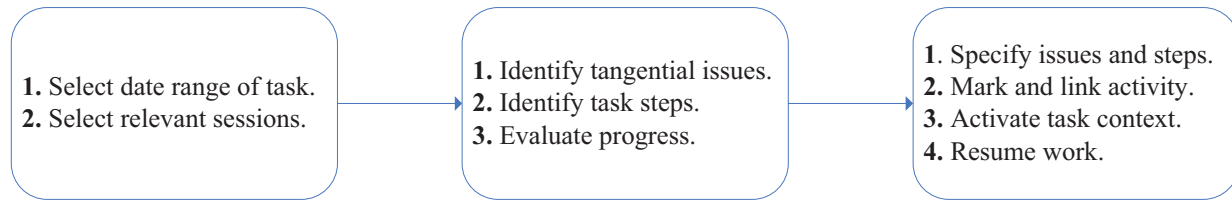


Figure 1. Typical user workflow using our tool to resume task.

tion history. Interaction history is the record of low-level events (including navigation and edit events) arising from a programmer using an IDE. It may also include other events such as build commands, perspective changes, or refactoring operations.

The information contained in an event typically includes its type, timestamp, and the name of the affected code element. Unfortunately, many interaction histories do not include the actual change made to the program. An exception to this is the `Spyware` tool which captures the complete fine-grained history for every source code change made. In this paper, we use a simplified notion of change history by periodically capturing and comparing program snapshots.

3.2 Representations

Researchers have considered different representations for grouping and characterizing interaction events. For example, the number of events generated during the course of a development task can be in the order of tens of thousands. To better structure the data, the events can be grouped into sessions, or aggregated into a single representative event.

One common representation is to group the events into *development sessions*. A session is a period of work – operationally we delineate a new session when there is a break in activity of 15 minutes or more. In previous work, we found for 98% of 4.5 million events in our data from 85 programmers, the time between adjacent events was less than a minute [14]. This means tight groups can be formed with any threshold above a minute. But to a developer, a session should naturally align with how the work was performed. In addition to grouping events into sessions, the events within a session can be clustered into single events. For example, all the edit events within a minute can be clustered into a single event describing the edits per minute.

Another common representation is a *usage context* [11]. A usage context is a model containing a subset of elements belonging to the program source code. A usage context can be derived implicitly from the interaction history. Contrasting an implicit usage context with an explicit task context, the usage context may be more appropriate when no formal task specification was available to be associated with the development activity. A useful measure is to take the

momentum of an element. Momentum measures the degree-of-interest of a program element, which decreases exponentially when the element is inactive. Momentum approximates the relevance of a code entity between any two points in time, which is useful in a timeline view of activity.

Developers reconstructing their activities may need assistance identifying the different steps performed, recalling tangential investigations or unexpected problems, and evaluating progress of a task. As such, indications of when a developer’s focus shifted or events marking transitions in the stage of performing a task may better assist a developer in reconstructing the task structure. Build events as well as run events may serve as effective indicators for boundaries for different parts of a task. When developers complete a step in their task, they may wish to evaluate how well the changes made address the goal of the task. For instance, a common strategy is to build the program to check for compile errors and then execute the program to check for problems.

4 Visualization and Reconstruction

In this section, we provide an overview of our approach and describe the workflow of a developer using the visualization to resume or summarize past work.

The visualization has two main components:

- **TaskPad.** The TaskPad is a view for displaying a complete or partial task specification.
- **Activity Timeline.** The Activity Timeline is a visualization providing an overview of the recent development activity.

The workflow of a developer using the system is depicted in Figure 1. The developer performs work as normal. The task the developer works on may be exploratory in nature, and thus does not have a well-defined task specification, and does not include a plan on how to implement the proposed change. After making progress, the developer becomes interrupted and must attend to another matter. Eventually, the developer returns to the original task. Previously, the developer had begun implementing a possible solution to the task but had become derailed in resolving an tangential issue, and does not remember the details of the solution or if the issue was resolved.

To resume working on the task, the developer can use the activity timeline to get an overview of the past activity. In order to select the correct time frame, the developer has the choice of activating the timeline from a task specification in the TaskPad or selecting the time range the task was last worked on. From this time range, the developer can narrow down the range or removing irrelevant sessions. Once the developer is satisfied with the time frame, the developer then inspects the activities on the timeline.

The activity timeline contains several cues and indicators to help recall activities and issues encountered in the past sessions. Using the error indications on the timeline, the developer is able to recall that a tangential issue had occurred and still needed to resolve some aspects of the issue. Using the summary cues, the developer is able to recall the details of the solution being sketched prior to the unexpected problem.

Using the reconstruction interface, the developer is able to indicate the time frame for the problematic issue. Next, the developer marks the time frame related to working on the solution for the original task. These work breakdowns become available in the task specification seen in the TaskPad. Further, the code elements from selected time frames of the task can be captured and activated as separate task contexts. Finally, the developer is able to complete the task; moreover, a complete task specification is now available for task tracking.

In the following subsections, we provide more details about the visualizations and interfaces of the components.

4.1 TaskPad

The TaskPad is an important component for capturing task specifications and activating the task context associated with a task or a task breakdown. The TaskPad is designed to allow a developer to concurrently work with several distinct but inter-related tasks. Each task may have steps which are not formally stored in a task repository but can be associated with their own task context or parent task context. This allows the developer to refine a task as it is performed and formulate tentative plans without involving heavy-weight task specification. The intended form factor of the TaskPad was designed to be docked on the side of the IDE in a manner similar to the Solution Explorer in Visual Studio, or the Package Explorer in Eclipse.

On the TaskPad (see Figure 2), each active task is given a separate section which can be expanded or collapsed. Within the task section, the task specification is displayed in an indented list of task breakdowns. Each breakdown is preceded by a breakdown type and then followed by a brief text description. Each task has a thumbnail of an activity timeline. The thumbnail can be previewed or clicked to activate and display the activity related to the task element.

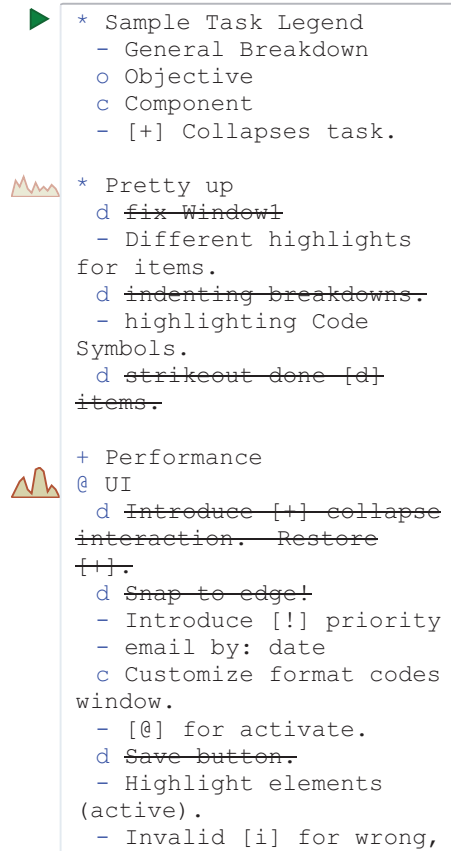


Figure 2. TaskPad.

4.2 Activity Timeline

The Activity Timeline displays the sessions occurring during a specified time range. We designed the Activity Timeline to provide an effective but compact overview of the recent development activities. In the design, we focused on constraining the display form factor in a manner that would be suitable for use during development, but in a way that would not detract from the normal layout of the IDE. A long but thin horizontal space allows developers to focus on the code while still maintaining an overview of recent activities. Further, this would be consistent with the layout and metaphor of a timeline.

The design of the timeline is as follows: each session is placed on the timeline in the order of its occurrence and separated in proportion to the length of inactivity between sessions. As many events can take place over several hours or days, by default the events are clustered into a period of one minute. This clustering allows a session of one hour to be displayed using an area only 60 pixels wide. In our experience with interaction histories and visualizing sessions, when trying to understand what happened in a session, little value was gained in trying to display each event or showing

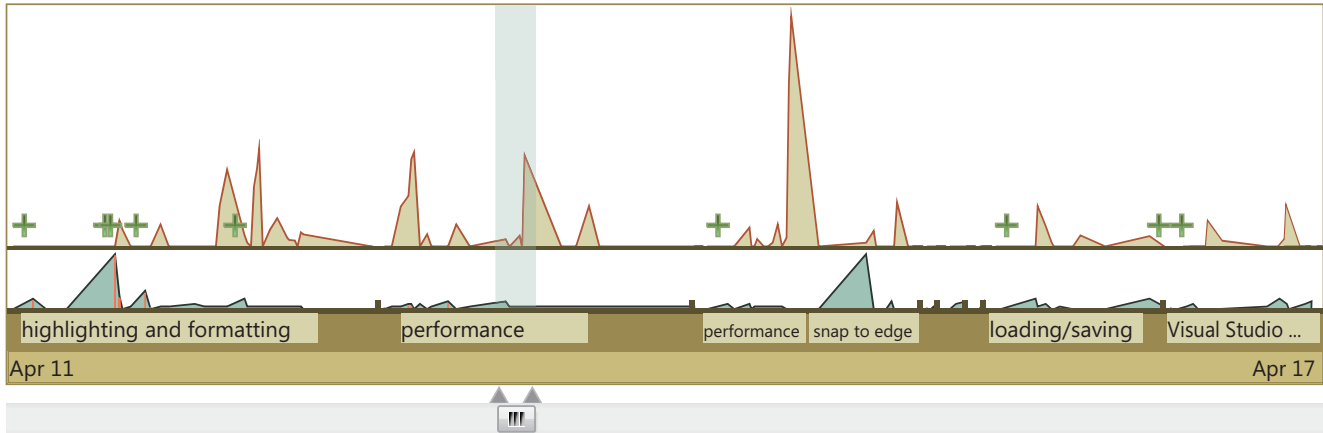


Figure 3. Activity timeline illustrating development of TaskPad.

the level of activity such as edit per minutes versus navigation per minute. Also displaying the usage context or changed code within one particular moment of time provides too little context to be understood independently. Instead, we focus on highlighting and summarizing significant events within the session.

An example instance of the Activity Timeline is shown in Figure 3. Starting from the bottom of the figure, a dynamic range slider can be used to select an arbitrary time range along the horizontal axis and slide in either direction for detailed inspection of information. Above the slider, the start and end dates are displayed. Next, ranges of activities can be labeled and associated with task elements specified in the TaskPad. Above the activity labels, two graphs are displayed. The lower graph represents the number of methods edited at a point of time, whereas the top graph illustrates the total momentum of code elements.

We selected three primary characteristics of the events to focus on:

- **Workload Size.** Charted in the lower graph of the timeline is the number of methods in the usage context at a particular point in time. The graph can be useful for different phases of programming activity. For example, in the first phase, a developer often undergoes a period of fleshing out many methods. A second phase then follows that is longer but involves fewer methods at a given point of time. Relying on the graph of method size alone, a viewer might be misled into thinking that the developer was focusing on the same few methods over a period of time. Rather, during this time a developer might be rapidly shifting between different contexts to handle many different issues that may arise. For this reason, the momentum graph is used to illustrate the consistency of the developer's context.
- **Structural changes.** Displayed along the timeline are

icons indicating major structural changes to the program such as the addition of new classes (plus icon) or removal of methods (minus icon).

- **Boundaries.** To assist identifying task boundaries, several indicators are given. In the lower graph, whenever a method is first encountered within a session, it is highlighted with a vertical mark. Second, the momentum graph indicates shifts in attention often characterized by many small humps that occur because a developer is rapidly moving between contexts.

The activity timeline offers details that can be queried on demand. Developers can hover over different indicators or markers causing a modeless popup with more information to display. The interaction is intended to be a fluid and quick operation. The different types of modeless popups are the following:

- **ContextView.** A treeview containing the projects, classes, and methods belonging to a time range of activity. An example is shown in Figure 4.
- **LaunchPad.** The launch pad is a view for displaying relevant code symbols from the method bodies that were involved in recent changes. The code symbols are displayed in a tag cloud of word parts that link to the source code locations. An example is shown in Figure 5.

The ContextView and LaunchPad views can also be docked in the IDE. When docked, the developer can freely slide the dynamic range and see the updated content in the views. This is useful for quickly characterizing the activities that have occurred: the developer can see the locations in the ContextView and the changes made to the source code within an arbitrary range of time. After recognizing an activity that was performed, a developer can persist this knowledge in the timeline by labeling the activity and associating it with a task in the TaskPad.

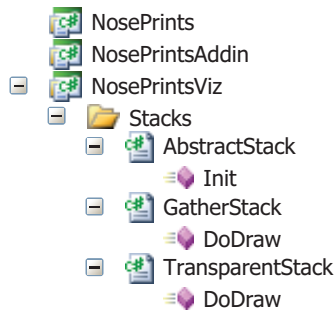


Figure 4. Context View.

5 Discussion

In the previous section, we described a tool for assisting a developer when resuming or summarizing a task. In this section, we discuss some of the benefits and drawbacks of our approach and mention other possible alternative designs and open questions for future research.

Although our approach has several advantages over other proposed designs, several questions still remain. Our approach assumes that a development session is the most logical and natural method for reasoning and understanding development activity. Even though this may be true, the method may still suffer from our definition of a session. For example, instead of using the length of inactivity to define a session, a developer might think that a session defined by a commit to a source code repository is a better working definition. Another problem is that our approach assumes that the developer is easily able to identify the relevant sessions related to a task by selecting a time range and then narrowing down sessions. For this approach to be successful, the work on a task must occur within a relatively close time frame; however, if a task was worked on sporadically interspersed with other tasks, then a developer could have problems to identify the task. In that case, a method for automatically identifying and suggesting related sessions might be needed. A final issue is that we do not adequately address how a developer might track the progress of the same task occurring in multiple segments within a session or across multiple sessions. For example, if a developer works on a particular issue over several sessions, they may want to associate certain progress toward the goal of the task that may not directly be aligned with a task breakdown.

In this paper, we did not take advantage or account for different types of tasks or task breakdowns which might need different types of tool support. For example, one type of task could be describing how to implement a new feature. The feature might be discussed in terms of capabilities or components that need to be developed. A component may be a useful abstraction for attaching task knowledge and linking activity, whereas a capability may be useful for eval-



Figure 5. Launch Pad.

uating progress. Another type of task might be a change that includes a solution sketch or describes a desired architectural modification. When summarizing how these changes were made, the architectural modification might need different types of tool support in comparison to summarizing localized changes.

There are several ways to expand upon our approach. Certainly, there are more effective ways of summarizing code changes in a manner beyond displaying a code difference. One starting point would be to include better structural summaries using the approach described in recent work by Kim and Notkin [9]. Another way to improve this approach is to introduce more proactive ways of reminding a developer of her activity. Currently, we rely on the developer using an exploratory interface for triggering memories of past activities. An alternative approach is to allow the developer to leave prospective notes that could be activated under certain conditions. These sort of reminders would be useful in the situation where a developer is blocked from completing a task while waiting for another team member to complete a task, or the project is not in a buildable state. When the situation is resolved, then the developer could be reminded to complete the task. Finally, we mainly focused on the source code artifacts for collecting activity. However, a more interesting approach might be to instrument

and gather information from the runtime of the program execution or from a debugging session. Several relevant elements of interest could be collected:

- **Runtime exceptions.** Capture runtime exceptions encountered during the execution of the program. These events can be displayed on the timeline and expanded in a popup view.
- **Runtime values.** Display values of code symbols, including both typical and extreme values.
- **Runtime paths and slices.** Display paths or code slices from the program execution in a particular debugging session.

6 Status

We have implemented our tool as a Visual Studio plugin. We have informally experimented using the tool for inspecting several recorded programming sessions to understand how well the different measures serve for recalling activities. In one case, we were able to use the tool to inspect development activity that occurred over a year ago and identify the associated development tasks without referring to source code. We are also in the process of designing experiments for better understanding what are effective cues for recalling development activities.

7 Conclusion

In this paper, we have presented a technique that allows a developer to resume or summarize a task by reconstructing the task specification from previous development activity. We described the components used for viewing partial task specifications and past activity in a timeline visualization from which the task specification can be reconstructed and linked to the previous activity.

An important contribution of this work is to organize the many aspects explored by several previous efforts and introduce several new ideas in the context of a single workflow addressing the specific problem of resuming or summarizing the work performed for a task. Future work can explore and evaluate other workflows for resuming or summarizing a task. In addition, the effectiveness of different visual cues should be investigated for their ability to assist recall of information. The metaphor of an activity timeline can be a powerful concept for assisting developers in task management; however, the question remains: what are the right choices for abstractions of activities to place on the timeline?

References

- [1] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. Threats on building models from cvs and bugzilla repositories: the mozilla case study. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 215–228, New York, NY, USA, 2007. ACM.
- [2] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1313–1322, New York, NY, USA, 2007. ACM.
- [3] I. D. Coman and A. Sillitti. Automated identification of tasks in development sessions. In *ICPC '08: Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, pages 212–217, Washington, DC, USA, 2008. IEEE Computer Society.
- [4] J. B. Ellis, S. Wahid, C. Danis, and W. A. Kellogg. Task and social visualization in software development: evaluation of a prototype. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 577–586, New York, NY, USA, 2007. ACM.
- [5] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 39–48, New York, NY, USA, 2006. ACM.
- [6] D. F. Jerding and J. T. Stasko. The information mural: a technique for displaying and navigating large information spaces. In *INFOVIS '95: Proceedings of the 1995 IEEE Symposium on Information Visualization*, page 43, Washington, DC, USA, 1995. IEEE Computer Society.
- [7] M. S. John, H. S. Smallman, and D. I. Manes. Recovery from interruptions to a dynamic monitoring task: the beguiling utility of instant replay. In *Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting*, 2005.
- [8] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *SIGSOFT '06/FSE-14: Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 1–11, New York, NY, USA, 2006. ACM.
- [9] M. Kim and D. Notkin. Discovering and representing systematic code changes. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, 2009.
- [10] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. In *ICSE '09: Proceedings of the 31st international conference on Software engineering*, New York, NY, USA, 2009. ACM.
- [11] 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, Washington, DC, USA, 2006. IEEE Computer Society.
- [12] C. Parnin and C. Görg. Design guidelines for ambient software visualization in the workplace. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 18–25, June 2007.

- [13] C. Parnin, C. Görg, and S. Rugaber. Enriching revision history with interactions. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 155–158, New York, NY, USA, 2006. ACM.
- [14] C. Parnin and S. Rugaber. Resumption strategies for interrupted programming tasks. In *ICPC '09: Proceedings of the 17th IEEE International Conference on Program Comprehension*, Washington, DC, USA, 2009. IEEE Computer Society.
- [15] R. Robbes and M. Lanza. Characterizing and understanding development sessions. In *ICPC '07: Proceedings of the 15th IEEE International Conference on Program Comprehension*, pages 155–166, Washington, DC, USA, 2007. IEEE Computer Society.
- [16] R. Robbes and M. Lanza. Spyware: a change-aware development toolset. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 847–850, New York, NY, USA, 2008. ACM.
- [17] I. Safer and G. C. Murphy. Comparing episodic and semantic interfaces for task boundary identification. In *CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 229–243, New York, NY, USA, 2007. ACM.
- [18] A. Sarma, D. Redmiles, and A. van der Hoek. Empirical evidence of the benefits of workspace awareness in software configuration management. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 113–123, New York, NY, USA, 2008. ACM.
- [19] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer. Todo or to bug: exploring how task annotations play a role in the work practices of software developers. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 251–260, New York, NY, USA, 2008. ACM.
- [20] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 193–202, New York, NY, USA, 2005. ACM.