# ActivitySpace: A Remembrance Framework to Support Interapplication Information Needs

Lingfeng Bao[1,2], Deheng Ye[2], Zhenchang Xing[2], Xin Xia[1], and Xinyu Wang[1]

[1]College of Computer Science, Zhejiang University, Hangzhou, China
[2]School of Computer Engineering, Nanyang Technological University, Singapore
{lingfengbao, xxkidd, wangxinyu}@zju.edu.cn; {ye0014ng, zcxing}@ntu.edu.sg;

***Abstract*—Developers' daily work produces, transforms, and communicates cross-cutting information across applications, including IDEs, emails, Q&A sites, Twitter, and many others. However, these applications function independently of one another. Even though each application has their own effective information management mechanisms, cross-cutting information across separate applications creates a problem of *information fragmentation*, forcing developers to manually track, correlate, and re-find cross-cutting information across applications. In this paper, we present *ActivitySpace*, a remembrance framework that unobtrusively tracks and analyze a developer's daily work in separate applications, and provides various semantic and episodic UIs that help developers correlate and re-find cross-cutting information across applications based on information content, time and place of his/her activities. Through a user study of 8 participants, we demonstrate how *ActivitySpace* helps to tackle information fragmentation problem in developers' daily work.**

**Tool website:**
**http://baolingfeng.weebly.com/ase2015-demonstration.html**

## I. Introduction

Software development involves not only software development tools but also many other software applications specializing in different tasks. Our study of developers' daily work [1] shows that developers can use 6 or more different desktop and web applications in their daily work, and access over 79±41 (mean±standarddeviation) distinct documents in these applications everyday. Furthermore, the information associated with developers' work is often cross-cutting across applications. For example, while a developer was working on a source code file, he was also accessing over 40 other documents (e.g., web pages, text files, and office documents) in other applications. Finally, developers frequently revisited the previously used document in the following days. Although most of the revisits occurred in 2-7 days, developers sometimes revisited some documents even after 20 days.

Desktop and web applications that developers use in their daily work function independently of one another. Even though each application provides effective information management mechanism, cross-cutting information associated with developers' work creates a problem of *information fragmentation*, forcing developers to manually correlate and re-find the different related information across separate applications. The information fragmentation problem in developers' daily work calls for innovations to support *interapplication information needs* in software development, going beyond tool integration (e.g., IBM Jazz [2], SeaHawk [3]) and information management within separate applications (e.g., Mylyn [4], Context Web History [5]).

In this paper, we present a remembrance framework (called *ActivitySpace*) that supports tracking, correlating, searching, and interactively exploring cross-cutting information in developers' daily work. The framework is composed of three components as shown in Fig. 1: 1) *Activity Tracker* (green color) collects low-level action records using operating-system (OS) level instrumentation and computer vision techniques; 2) *Activity Analysis* (blue color) infers action transcripts from action records, aggregates action transcripts into activities, and identifies correlated activities and documents based on temporal locality [6]–[8]; and 3) *Activity Interface* (red color) provides semantic and episodic user interfaces (UIs) to help users view, search and explore cross-cutting activities and information across applications.

We have implemented a prototype of the *ActivitySpace* framework on Microsoft Windows platform. We invited 8 developers in a user study of the *ActivitySpace* prototype. We collected 417 hours activity data in developers' daily work. Analyzing the collected activity data suggests that the *ActivitySpace* prototype provides effective mechanisms for tracking, correlating and re-find cross-cutting information across applications. The *ActivitySpace* framework could provide an enabling infrastructure for *activity-centric interapplication virtual workspace* and *community of practice*.

## II. The *ActivitySpace* Framework

In this section, we describe the implementation of the *ActivitySpace* framework. We focus on *Activity Interface* in this paper. Interested readers are referred to our technical paper for the detailed discussion on the design of *Activity Tracker* component and the data analysis steps of *Activity Analysis* components.

### A. Activity Tracker

*ActivityTracker* runs on the user's computer. It unobtrusively collects low-level Human-Computer Interaction (HCI) data when the user interacts with software applications. *ActivityTracker* uses mouse and keyboard hook to monitor developer actions in different applications. It allows the user to specify which application(s) to (not to) track and what mouse/keyboard actions to (not to) monitor. Once an interested mouse or keyboard action occurs in a to-be-monitored application, *ActivityTracker* generates an action record that records the low-level HCI data, including window information,

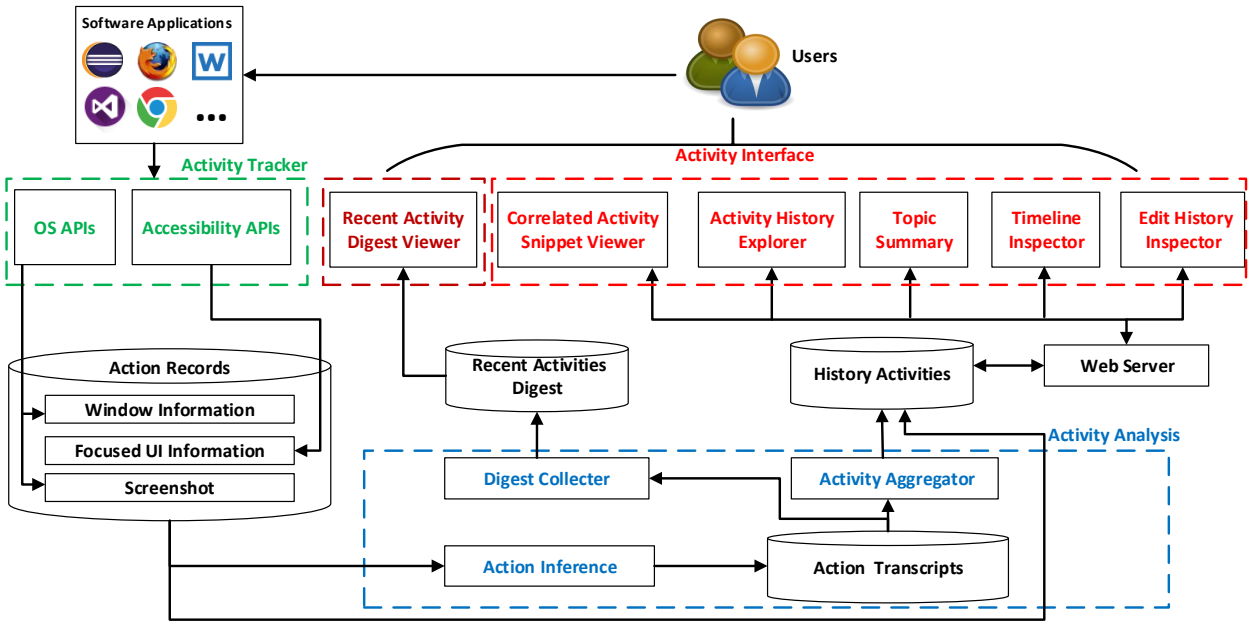



Fig. 1: The Architecture of the ActivitySpace System

focused UI information, and a screenshot, using Windows APIs and accessibility APIs. As the user interacts with software applications, *ActivityTracker* generates a continuous stream of actions records, i.e., a sequence of time-ordered actions regarding when the user uses which application and access what information. This stream of actions records is stored in action records database (can be on individual user's computer or on a centralized server in team setting)

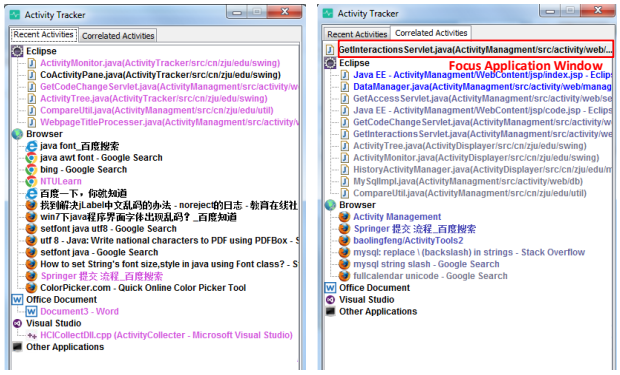
### B. Activity Analysis

*Activity Analysis* components run on the server. These activity analysis steps are time consuming, and thus cannot be performed by the *ActivityTracker*. Otherwise, the *ActivityTracker* cannot be *unobtrusive* and *efficient*.

As new action records arrive, the *ActionInference* component processes the new action record to infer application being used, developer action, document involved, and document content (if available) from window information, focused UI information, and the screenshot. It produces a corresponding *action transcript* stored in the action transcripts database.

The *DigestCollector* component updates application usage and document usage statistics for the most recent action transcripts within a fixed-length time window (30 minutes by default). These usage statistics are referred to as *recent activities digest* stored in the recent activities digest database.

The *ActivityAggregator* component periodically aggregates archived action transcripts into *history activities* stored in the history activities database. *ActivityAggregator* can be configured to run at a given time interval (60 minutes by default). *ActivityAggregator* assumes that the information used by the developer across separate applications exhibit *temporal locality* [6]–[8], i.e., actions which occur at closer points in time are more likely to be conceptually related. Thus, *ActivityAggregator* regards as an activity all the actions on a particular document within an application that occur within the time interval threshold (30 minutes by default). If the time




(a) Recent Activity Digest Viewer (b) Correlated Activity Snippet Viewer

Fig. 2: DigestViewer and SnippetViewer

span of the two activities overlap, *ActivityAggregator* considers the two activities as correlated activities, and the documents involved in the two activities as correlated documents.

### C. Activity Interface

The *ActivitySpace* framework implements several episodic and semantic UIs that provide different *information scents* for correlating and re-finding cross-cutting information across applications. These UIs has been implemented as desktop and web applications. Semantic UI enables the developer to find information based on the content (i.e., *what*) he has used. Episodic UI enables the developer to find information based on the time (i.e., *when*) and place (i.e., where) of his actions.

*1) Recent Activity Digest Viewer (DigestViewer):* *DigestViewer* provides a semantic UI that shows the most recently used applications and documents. The view is an observer of recent activities digest. It will be updated automatically once recent activities digest has been updated by *DigestCollector*. *DigestViewer* displays recent activity digest in a tree view

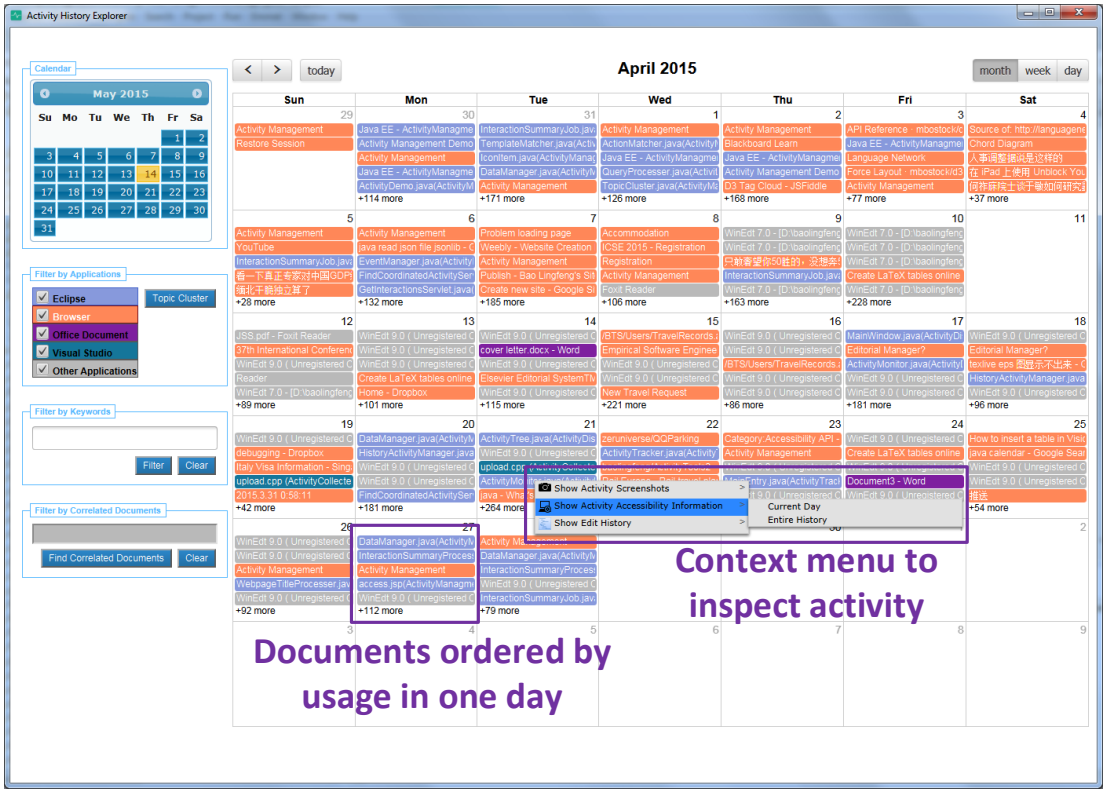



Fig. 3: Activity History Explorer

(Figure. 2(a) [1]). The first level shows the applications, and the second level shows the documents used in each application. The applications and documents are ordered by the time spent on them in the latest activity digest time window (30 minutes by default). If a document has been used beyond the latest activity digest time window, the document will be highlighted in pink. This can remind the developer that he has used this document in the past.

*2) Correlated Activity Snippet Viewer (SnippetViewer):* When the developer works in an application, if he presses the keyboard shortcut "Alt+X", *SnippetViewer* first identifies the document that the developer currently uses and generates a request to the web-server component *SnippetCollector*. Then, *SnippetCollector* searches the history activities for the time when the current document has also been used before. By default, the search is done for all the history activities. But the user can specify a time limit for search. Next, *SnippetCollector* retrieves the activities involving the current document and other correlated activities. Finally, it generates an activity snippet regarding application usage and document usage in those correlated activities, in terms of applications being used and the time spent on each application, and document being used and the time spent on each document and the time a document has been last used.

*SnippetCollector* displays the generated activity snippet in the *SnippetViewer* (see Fig. 2(b)). *SnippetViewer* provides a semantic UI that shows the documents that have been used before together with the document the developer currently uses. *SnippetViewer* displays the activity snippet in a tree view similar to that of *DigestViewer*. The applications and documents are ordered by the total time spent on them in the past. In *SnippetViewer*, the older a document has been last used, the brighter color the corresponding document item is.

*3) Activity History Explorer (HistoryExplorer):* *HistoryExplorer* provides an episodic UI that allows the developer to search, filter, and explore the documents he has used in a calendar view (see Fig. 3). The calendar view allows the developer to view the documents by month (week or day) that provides an overview of the documents used in a month (week, or day). The documents used in a particular day are ordered by the total time spent on them in that day. The documents with the longest usage time are shown in the view. More documents can be shown by clicking "+xx more" link. *HistoryExplorer* highlights documents used in different applications using different colors. *HistoryExplorer* allows the user to select date and move backward/forward in time.

*HistoryExplorer* provides three ways to filter the documents shown in the calendar view. The developer can use any combination of these three ways. First, the developer can filter the documents by application(s). By checking or unchecking an application, the documents used in the application will be included or excluded in the view. Second, the developer can search the documents using keywords. *HistoryExplorer* shows only the documents whose window title and document content (if any) contain the keywords. Third, the developer can double-click a document in the calender view, and this document will fill in the "Filter by Correlated Documents" search box. Then, the developer can "Find Correlated Documents" which cause *HistoryExplorer* show only the documents correlated to the selected document. The developer can also select a document

[1]Full size screenshots can be found at our tool website.

in *DigestViewer* or *SnipperViewer* and then open *HistoryExplorer*. This is equivalent to "Find Correlated Documents" by the selected document.

*4) Topic Summary:* In *HistoryExplorer*, the developer can click *Topic Cluster* button in the "Filter by Application" panel. *HistoryExplorer* calls the web-server *TopicSummarizer* component. *TopicSummarizer* uses a topic mining tool Carrot2 [9] to mine the topics in the window title of the applications involved in the history activities for a period of time (10 days by default). All the checked applications in the "Filter by Application" panel are considered in topic mining. *TopicSummarizer* visualizes the mined topics in a foam tree using Carrot2 FormTree [2]. The foam tree provides a semantic UI for the developer to recall some important topics for a period of time. The developer can select topics in the foam tree as keywords to filter the documents in the *HistoryExplorer*.

*5) Timeline Inspector:* The developer can select a document in *DigestViewer*, *SnipperViewer* or *HistoryExplorer* (see Fig. 3), and view the timeline details of activities involving the selected document in the two timeline inspectors: *ScreenshotInspector* and *AccessbilityInspector*. The two inspectors are complementary. They provide episodic UIs for the developer to inspect and recall when and where he did what. The developer can view the details of activities involving the selected document in the current day or in the entire history.

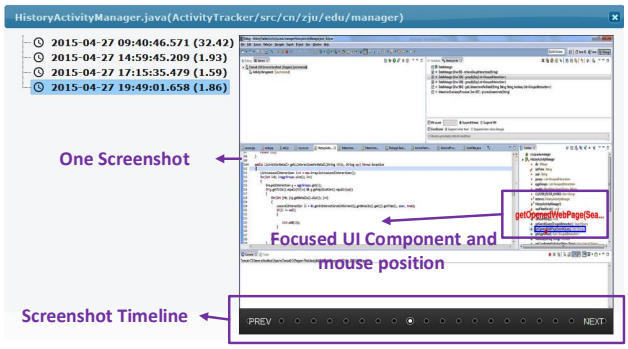



Fig. 4: The Screenshots of Activities in One Day

*ScreenshotInspector* shows the screenshots of action records in the activities involving the selected document in a time line (see Fig. 4). The activities are listed in the left panel by chronological order of their start time, and we also show time span (minutes). The right panel shows the screenshots of a selected activity. Each screenshot is represented as a circle in the time line. For each screenshot, *ScreenshotInspector* highlights the boundary of focused UI component and mouse position on the screenshot. This can remind the developer of his action in the screenshot. *ScreenshotInspector* can replay the screenshots according to the recording time of the screenshots. The developer can also select the screenshots in the time line.

*AccessibilityInspector* lists the focused UI information of action records in the activities involving the selected document in a table view (see Fig. 5). The activities are listed in the left panel by chronological order of their start time and its time span. Each row in the table view shows four pieces of focused UI information: time stamp, name and type of UI and name of parent UI. The table view can be sorted by one of the four

[2]http://download.carrotsearch.com/foamtree/demo/

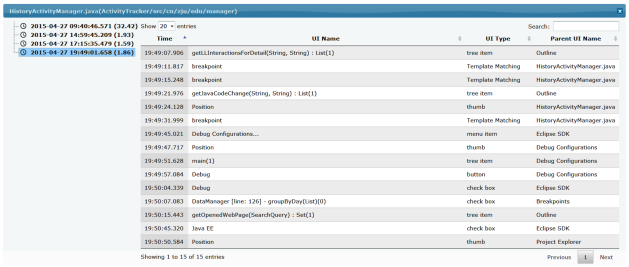



Fig. 5: The Focused UI Information of Activities in One Day

columns. The developer can enter one or more keywords in the top-right search box. *AccessibilityInspector* will show only action records that contain the entered keywords.

By inspecting the screenshots in Fig. 4 and the accessibility information in Fig. 5, the developer can recall that he was debugging *HistoryActivityManager.java* in the evening of April 27, 2015 and what he actually did in this debugging activity.

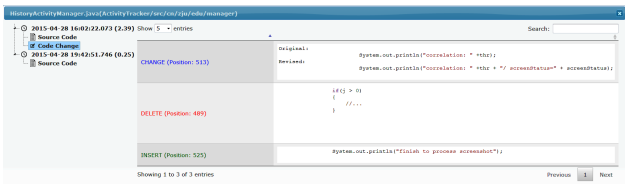



Fig. 6: An Example of Code Edit History

*6) Edit History Inspector:* The developer can select a source file in *DigestViewer*, *SnipperViewer* or *HistoryExplorer* (see Fig. 6), and view the complete edit history of the file in *EditHistoryInspector*. The system currently supports types of source files commonly used in Eclipse and Visual Studio. But it can be easily extended to support more types of source code file, such as those of Text Editor if the content of source code can be extracted from the focused editor UI component when the developer works in the editor.

*EditHistoryInspector* provides an episodic UI that lists all the activities in which the selected source file has been used (see Figure 6). The left tree view shows the start time of these activities in reverse-chronological order. For each activity, *EditHistoryInspector* shows the content of the source file at the end of this activity. It also computes the code changes between the contents at the end of the two consecutive activities. It uses the file differencing tool [3] that reports code line changes, inserts and deletes. *EditHistoryInspector* shows the code changes in chronological order in the right table. The user can search code changes by keywords. If there is no code change in an activity, there will be no corresponding "code change" item under the activity.

## III. Use Scenarios

We invited 8 volunteer graduate students in a user study of the *ActivitySpace* prototype. The participants installed the *ActivitySpace* prototype on their working computer. The system was configured to track user activity in web browsers (*Firefox, Chrome, Internet Explorer*), office software (*Word, Excel, PowerPoint*), PDF reader (*Adobe Reader, Foxit Reader*),

[3]https://code.google.com/p/java-diff-utils/

text editors (*Notepad, Notepad++*), latex editor (*WinEdt*), and IDEs (*Eclipse, Visual Studio*). Activity in all other applications are categorized as others. In this section, we report the two use scenarios from the user study. Detailed discussion on the study results can be found in our technical paper [1].

### A. Recalling Intermediate Code Changes

The participant S1 experimented gradient color scheme for rendering tree items. During the process, he tried out the Java class "java.awt.Font". A few days later, he needed to use "java.awt.Font" again. He wanted to find the code fragment he experimented before. But he cannot find it in the latest source code, because he did not commit the intermediate code changes to the repository. He used *HistoryExplorer* to find the source file "ActivityTree.java" that he worked on a few days ago. The entry also reminded him that he searched "setfont java" and read a web page about how to set font size. He revisited the web page and found it useful for his current work. Furthermore, he viewed edit history of "ActivityTree.java" on that day which help him recall what he experimented with the class. The re-found web page and the fine-grained code edit history helped the participant S1 use "java.awt.Font" in his current work.

### B. Finding Correlated Documents

The participant S5 was working on a research project involving web application development and paper writing. When he worked on the "preliminarystudy.tex" in latex editor, he activated the *SnippetViewer* to view the correlated documents during his previous work on "preliminarystudy.tex". This showed him not only the relevant latex source files, but also the source files of his web application (e.g., "preprocess.py", "taskSolution.py"), the relevant web pages (e.g, "STAR Laboratory: SRI Language Modeling Toolkit") that he visited, and the search queries (e.g., "srilm tutorial", "vtk", "opencv") that he used to search Google. He then opened *HistoryExplorer* from *SnippetViewer* to get an overview of when he used these correlated documents. *ActivitySpace* helped S5 refresh his memory of his previous work in the project to continue his work.

## IV. Related Work

IDEs have been designed to improve developer productivity by integrating then-separate software development tools. In addition to development tools, other applications have also been integrated within the IDE to ease the access of relevant tools and data in software development. For example IBM Jazz [2] integrates messaging tools, SeaHawk [3] integrates Stack Overflow. Other than tool integration, another direction is to support task-centric information management in knowledge work. Mylyn [10] supports task-focused interface for task management in the IDE. Other tools can analyze user activity to recommend additional information or help user re-find information, such as TaskPredictor [11], UMEA system [12], Reverb [13]. Different from tool integration and task management, *ActivitySpace* focuses on correlating and re-finding cross-cutting information across separate applications without application-specific support and tool integration.

The *ActivitySpace* framework is inspired by the activity theory [14] and the recent development of activity-centric computing tools [15]–[20]. Activity-centric computing has been proposed as a computing paradigm that supports the users' activities rather than the resources and tools used to perform such activities. Existing activity-centric systems mainly focus on activity construction and activity resumption. In contrast, *ActivitySpace* unobtrusively tracks developer's daily work and automatically infers activities and their correlations.

## V. Implications on Future Research

The *ActivitySpace* framework is the first step towards an *activity-centric interapplication virtual workspace* that would allow the developer to work in a shared information context across application boundaries. For example, when the developer is editing the code in the IDE, the virtual workspace may inject code snippets from the opened web pages into the code auto-completion assistant. The developer may integrate a code snippet from the web without switching between the IDE and the web browser. Next time the developer visits the web page, the virtual workspace could remind the developer that he reused a code snippet from this web page, and inject in the web page a summary of how he modified the code. It can present warnings to the developer if the code snippet was later removed, suggesting that the page may not be that useful for the task.

The *ActivitySpace* framework could be deployed in a development team or an online community. As a team or community remembrance framework, the *ActivitySpace* system could advance the paradigm of knowledge sharing from community of knowledge (e.g., open source projects, Q&A sites) to *community of practice*, where developers can share not only their work products but also working process within the team or community. The working process usually contains valuable context-based experiences which are often considered as tacit knowledge (i.e., *know how*), because they cannot easily be captured, codified and stored [21].

## VI. Conclusion and Future Work

In this paper, we present the *ActivitySpace* framework and its prototype for tackling information fragmentation problem in developers' daily work. *ActivitySpace* tracks and analyzes cross-cutting information across separate applications in developers' daily work. Its semantic and episodic UIs create interapplication information scents to ease the process of correlating and re-finding cross-cutting information across applications in developers' daily work. A pilot study demonstrated the usefulness of the *ActivitySpace* for supporting interapplication information needs. In the future, we will investigate activity-centric interapplication virtual workspace and community-of-practice for more effectively supporting integrated knowledge work in today's software development practices, which requires higher-level of information integration and management than current focus on tool integration and within-application information management.

## Acknowledgment

This work was partially supported by the Major State Basic Research Development Program of China (973 ProgramNo.2015CB352201)and National Key Technology R&D Program of the Ministry of Science and Technology of China (No. 2013BAH01B01). This work is supported by NTU SUG M4081029.020 and MOE AcRF Tier1 M4011165.020.

## REFERENCES


[1] L. Bao, Z. Xing, X. Wang, and B. Zhou, "Activityspace: An interapplication remembrance agent for integrated knowledge work," in *Proc. ASE*, accepted.

[2] R. Frost, "Jazz and the eclipse way of collaboration," *Software, IEEE*, vol. 24, no. 6, pp. 114–117, 2007.

[3] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the IDE," in *Proc. ICSE*, pp. 1295–1298, 2013.

[4] M. Kersten and G. C. Murphy, "Mylar: a degree-of-interest model for IDEs," in *Proc. AOSD*, pp. 159–168, 2005.

[5] S. S. Won, J. Jin, and J. I. Hong, "Contextual web history: Using visual and contextual cues to improve web browser history," in *Proc. CHI*, pp. 1457–1466, 2009.

[6] N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran, "Swish: Semantic analysis of window titles and switching history," in *Proc. IUI*, pp. 194–201, 2006.

[7] T. Rattenbury and J. Canny, "Caad: An automatic task support system," in *Proc. CHI*, pp. 687–696, 2007.

[8] C. A. N. Soules and G. R. Ganger, "Connections: Using context to enhance file search," in *Proc. SOSP*, pp. 119–132, 2005.

[9] J. Stefanowski and D. Weiss, "Carrot and language properties in web search results clustering," in *Web Intelligence, First International Atlantic Web Intelligence Conference (AWIC)*, pp. 240–249, 2003.

[10] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proc. SIGSOFT '06/FSE-14*, pp. 1–11, 2006.

[11] J. Shen, L. Li, G. Dietterich, Thomas, and L. Herlocker, Jonathan, "A hybrid learning system for recognizing user tasks from desktop activities and email messages," in *Proc. IUI*, pp. 86–92, 2006.

[12] V. Kaptelinin, "UMEA: Translating Interaction Histories into Project Contexts," in *Proc. CHI*, no. 5, p. 353, 2003.

[13] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: Recommending code-related web pages," in *Proc. ICSE*, pp. 812–821, IEEE, 2013.

[14] Y. Engeström, R. Miettinen, and R.-L. Punamäki, *Perspectives on activity theory*. Cambridge University Press, 1999.

[15] W. Geyer, M. J. Muller, M. T. Moore, E. Wilcox, L.-T. Cheng, B. Brownholtz, C. Hill, and D. R. Millen, "Activity Explorer: Activity-centric collaboration from research to product," *IBM Systems Journal*, vol. 45, no. 4, pp. 713–738, 2006.

[16] J. Bardram, J. Bunde-Pedersen, and M. Soegaard, "Support for activity-based computing in a personal computing operating system," in *Proc.CHI*, pp. 211–220, 2006.

[17] S. Jeuris, S. Houben, and J. Bardram, "Laevo: A Temporal Desktop Interface for Integrated KnowledgeWork," in *Proc. UIST*, pp. 679–688, 2014.

[18] S. Houben, P. Tell, and J. E. Bardram, "ActivitySpace: Managing Device Ecologies in an Activity-Centric Configuration Space," in *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces - ITS '14*, pp. 119–128, 2014.

[19] S. Houben, J. E. Bardram, J. Vermeulen, K. Luyten, and K. Coninx, "Activity-centric support for ad hoc knowledge work: A case study of co-activity manager," in *Proc. CHI*, p. 2263, 2013.

[20] S. Houben, S. r. Nielsen, M. Esbensen, and J. E. Bardram, "NooSphere: An Activity-Centric Infrastructure for Distributed Interaction," in *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia - MUM '13*, pp. 1–10, 2013.

[21] T. H. Davenport and L. Prusak, *Working knowledge: How organizations manage what they know*. Harvard Business Press, 1998.