Switch! Recommending Artifacts Needed Next Based on Personal and Shared Context

Alexander Sahm and Walid Maalej

Technische Universität München
Bolzmannstraße 3
85748 Garching
sahm@cs.tum.edu
maalejw@cs.tum.edu

Abstract: While performing a certain task software developers use multiple tools, read different artifacts and change others. As software developers are often interrupted during a task, they end up simultaneously using a vast set of tools and artifacts. They need to switch between those artifacts many times until a task is completed. In sum a lot of time gets wasted due to locating, reopening or selecting the right artifact needed next. To address this problem we introduce *Switch!*, a context aware artifact recommendation and switching tool for software developers. *Switch!* recommends artifacts that are likely needed in the current situation, based on task semantics, interaction history and community profile.

1 Introduction

Software engineers work on several tasks in parallel [KM06], and their work is often interrupted [GM04]. Several studies have shown the fragmented nature of software engineering work. In a survey with 800 software engineers [Ma09] we asked subjects how frequent do they switch their work focus because of, e.g. a high priority request or remembering something. We found that about 60% of respondents change the focus at least hourly. Other observational studies found that engineers switch their focus even more frequently: every five minutes on average [MGH05, KDV07]. Mark, Gonzalez and Harris observed 57% of tasks were interrupted. As a result work was often fragmented into many small work sessions [MGH05] and engineers have to deal with many task switches.

In order to perform a certain task, engineers usually use various tools, in average 5 tools for a single task [Ma09] (such as source code editors, bug trackers or version control systems). Moreover engineers read and change different artifacts, like source code files, bug reports or diagrams. A case study done by Zou [ZG06] found that 8 code files are read and 6 are changed during a task. As engineers typically spend only 50% of their time for code creation [MH09], and as Zou only observed code files inside IDEs, it is most likely that the overall number of used artifacts is even higher. Multiple tasks in conjunction with multiple artifacts per task increase the amount of artifacts the engineer has to deal with in parallel.

In this paper we introduce *Switch!*, a recommendation tool that supports software engineers to switch artifacts by employing task semantics, engineer's interaction history and shared community profiles. The contribution of the paper is twofold. First, it introduces a conceptual model for a context-aware artifact switching to support software engineering (SE) work. Second, it presents a framework and user interface concept, which implements this model. In section 2 we describe the problems *Switch!* addresses. In section 3 we present our solution concept and the metrics required to recommend artifacts depending on engineer's context. Section 4 introduces our tool and gives an overview of its user interface concept and architecture. In section 5 we present a paper-prototype based evaluation. Finally, section 6 discusses related work and section 7 summarizes the paper and gives an outlook on future directions.

2 Switching Problems

Modern desktop environments use windows as a main abstraction for artifact switching. Figure 1 shows the different implementations provided by *Microsoft Windows* and *Mac OS X*. We identified four drawbacks of this approach.

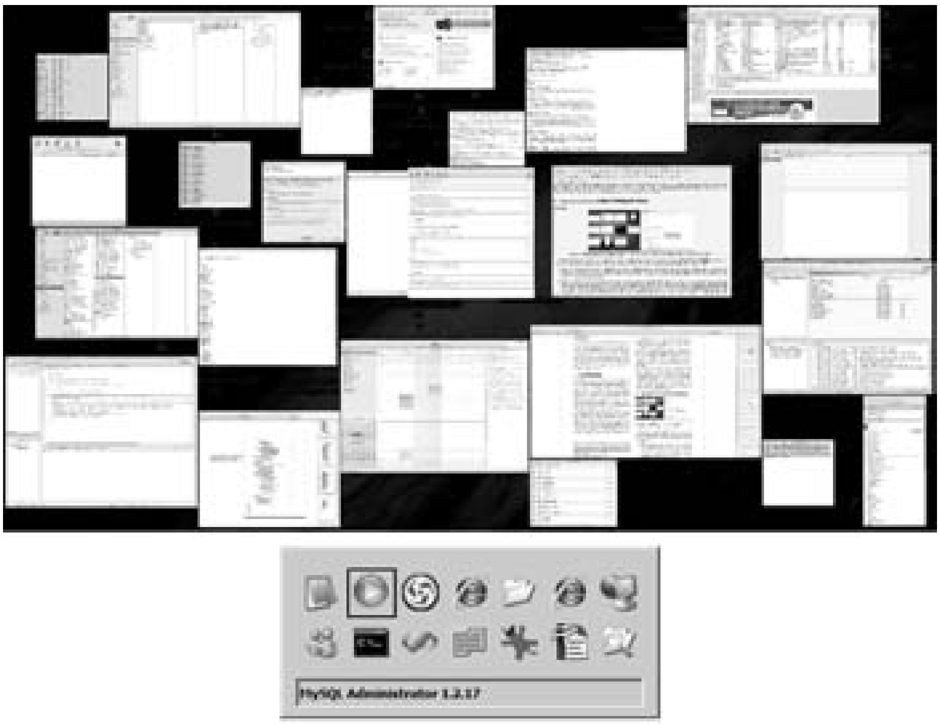

Figure 1: Window switching on Mac OS X (top) and Microsoft Windows (bottom)

1. *Mixture of task-specific windows*: Different tasks involve different artifacts and tools to use. While e.g. the debugger window is used for the bug fixing task, the text editor is used for the documentation refinement task. Window management systems are not aware of engineers' tasks and usually show windows of all tasks the engineer is working on. When switching artifacts, task irrelevant windows create a screen clutter.

2. *Window-centric instead of artifact-centric*: Desktop environments provide mechanisms to switch between open windows or running applications. However many applications display several artifacts in a single window. A web browser window, e.g., might contain several tabs. Thus engineers are not able to select one of these artifacts directly. Instead, first the window containing the artifact has to be identified, then the artifact inside the window.

3. *Task history is lost:* Information on artifacts and tools used during past tasks is not considered. If an engineer does not manually track the used artifacts in the current task, this information will be lost. When resuming a postponed task or working on a similar task in future the engineer will need to search and locate the same set of artifacts again.

4. *Open vs. closed windows:* Window management systems only consider open windows and running applications when switching artifacts. If an artifact is currently represented by a window, it is open. The engineer in this case can switch to it. If an artifact is not represented by a window, it is closed, and can not be a switch target. Engineers first need to find the artifact and open it manually. This interrupts the work-flow and can be time consuming.

3 Context-aware Artifact Recommendation

Our context model for a context-aware artifact recommendation consists of three dimensions: the semantics of the task being performed, the personal interaction history of the engineer and a shared community profile of co-workers. These dimensions respectively represent the "universal", the "personal" and the "social" perspective on SE work. We first describe the two dimensions and then introduce the recommendation algorithm combining them.

3.1 Task Semantics

SE tasks can be grouped based on the similarity of their goals and the software engineering activity they represent. Bug fixing tasks have a different goal than code-review or requirement analysis tasks. We distinguish between the following types of SE tasks (from [KDV07] and [MH09]): writing code, fixing bugs, reasoning about design, testing a program, maintaining requirements, integrating components and maintaining awareness. When working on one of these tasks (e.g. fixing a bug), engineers follow specific workflows (e.g. read report, reproduce bug, debug, commit), by using specific tools (e.g.

bug tracker, debugger) and specific types of artifacts (e.g. bug reports, source code). We annotate artifacts used during SE work with their semantic types. For example, a website rendering a bug report is of type *bug report*, a text file describing a test case is of type *test case*. The complete ontology describing the artifact and task types is available for download on the teamweaver site[1]. The ontology associates different artifact types to different task types. For example the engineer is likely to use a test case during testing, but not during implementation. The ontology enables to determine a fuzzy set of artifact candidates to be recommended, based on the type of current task.

3.2 Engineer's Interaction History

Even when working on similar tasks, software engineers differ in their work habits, work environments, work-flows, programming languages and tools. A need for a specific artifact highly depends on any of those factors. In order to address these needs we use the interaction history of the engineer. Analyzing engineer's interaction history gives insight on personal preferences. For example an engineer might use a specific folder for storing stable system builds. By analyzing her interaction history this preference can be detected and the folder can be recommended. We define three metrics to describe the engineers interaction with a single artifact.

- Usage Duration (*du*): The time in seconds the artifact was used by the engineer. Using includes reading as well as editing of the artifact.
- Usage Frequency (*fr*): The count how often an artifact has been used by the engineer.
- Switched To (*st*): The number of times this artifact was switched to from an other particular artifact. This value is relative to the current active artifact. The number is incremented if the user switches from the current active artifact to the given artifact.

These metrics are calculated for the artifact instances and the artifact types. While *du* and *fr* apply to a single artifact, the *st* value applies to two artifacts. The value of *st* describes the direct relationship between two artifacts (resp. artifact types). For example, a software engineer uses three different artifact types A, B and C. The engineer switches most of the time between A and C, and between C and B, but never directly between A and B. The *st* value is high for C when A or B is the active artifact type, or high for A and B if C is the active artifact type.

3.3 Shared Community Profile

Software engineers work in teams and share common characteristics with other engineers, i.e. members of particular communities. We argue that the context of co-workers

[1] http://www.teamweaver.org/wiki/index.php/ontology

is also of interest for recommendations. To reason about the context of co-workers their interaction history information is needed. To provide interaction history data, every developer of a project team or organization can share parts of their own interaction history. Before being shared, the interaction history is condensed to a profile, containing only abstract usage information. The data at this point cannot be traced to single tasks or interactions. This community profile contains the same numeric criteria as the personal interaction history described above. The client, depending on the profile provider's team role or experience level, aggregates the collected profiles.

By analyzing shared community profile team specific knowledge and best practices can be used for recommendation. The project or organization dependent work environment can also be detected. For example when the community profiles of the teammates are shared, the recommendation for a developer new to the project will also contain the project specific bug tracking system.

3.4 Recommendation Algorithm

The recommendation process is divided into two steps. First the number of possible artifacts to recommend is reduced by using the task semantic and focusing on a subset of the artifact types. In the second step artifacts of this types are ranked based on engineers interaction history. This reduces the number of rankings to be calculated and improves the performance of recommendation creation.

I. Artifact Type Suitability In order to identify the artifact type subset to use, we calculate a value representing the importance of each type. Only the artifact types with an importance rating higher than a given threshold are used. The used criteria refer to the interaction data: *du*, *fr* and *st*. The values are all normalized to be between 0 and 1, with 0 being unimportant and 1 being very important. Since the *st* is the only value directly linked to the current active artifact type, it is considered to be a higher relevance indicator and hence has a factor 2. The suitability of an artifact type *t* is calculated as follows:

$$Suitability(t) = du(t) + fr(t) + st(t) * 2$$

II. Artifact Ranking All artifacts of suitable types are ranked according to their relevance. In contrast to the first step, here we calculate the relevance of concrete artifacts. The same interaction data is available for the artifacts themselves. Interaction data is available for the following time slots:

- Current Session (cSe): The artifact interaction data collected during the current session (sessions result from postponing and resuming a task).
- Current Task (cTa): The artifact interaction data during the whole task (i.e. several sessions belonging to the same task).
- Similar Tasks (sTa): Artifact interaction data during tasks of the same type.

- External Tasks (*eTa*): Summarized artifact interaction data from co-workers during tasks of the same type than the current task.

The sTa adds the task semantics to the ranking of the artifacts while eTa represents the social aspect. The ranking overall is the combination of the universal task semantics, the personal interaction history and the social community profiles. We assume that recent interaction data is more important than old interaction data and personal interaction data is more important than universal or shared interaction data. We therefore assign the importance factors 2 for *cSe*, 1 for *cTa*, 0.5 for sTa and 0.25 for eTa. The artifact ranking can be calculated as follows:

$$\begin{aligned}
du(a) &= 0.25 * du(eTa, a) + 0.5 * du(sTa, a) + du(cTa, a) + 2 * du(cSe, a) \\
fr(a) &= 0.25 * fr(eTa, a) + 0.5 * fr(sTa, a) + fr(cTa, a) + 2 * fr(cSe, a) \\
st(a) &= 0.25 * st(eTa, a) + 0.5 * st(sTa, a) + st(cTa, a) + 2 * st(cSe, a) \\
Rank(a) &= du(a) + fr(a) + st(a) * 2
\end{aligned}$$

4 SWITCH!

We introduce *Switch!*, a tool that uses the described context model in order to recommend artifacts needed next. We present a short scenario on how *Switch!* is used. Then we describe its user interface and architecture.

4.1 Usage Scenario

Alice is implementing a new feature to a existing subsystem. She uses *Switch!* which monitors her interactions and the used artifacts and tools in the background. While trying to understand how the subsystem she is extending is designed, she presses the *Switch!* keyboard shortcut. *Switch!* evaluates her interactions and creates a recommendation of artifacts that Alice want to use next. The recommendation contains PDF files of UML diagrams, the email communication about this subsystem with a co-worker, as well as some web pages Alice looked at providing additional framework information. Alice can navigate through the offered artifacts by mouse or keyboard. Alice selects the artifact she wants to switch to. If the respective document is already open it will be brought to front, if it was not, it will be opened.

4.2 User Interface Concept

The user interface of *Switch!* is used by the engineers very often but only for short periods of time. Thus the user interface needs to be designed to foster fast recognition of the offered artifacts. We present the recommendations using a visual graphical interface instead of textual ones. This gives the engineer the ability to select the needed artifact quickly on a visual basis. Figure 2 shows the concept of *Switch!* user interface. We designed and implemented the user interface for the *Mac OS X* operating system. Though, its concepts can be implemented on different platforms as well.

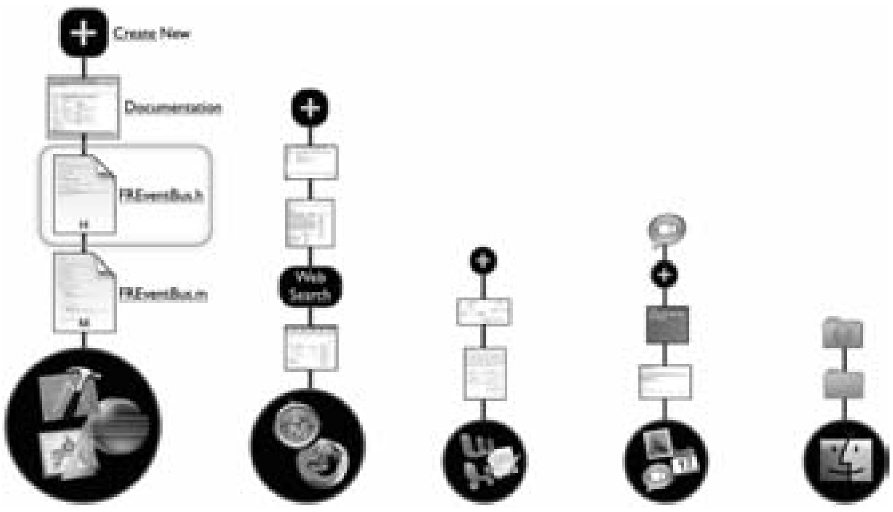

Figure 2: Switch! user interface showing a recommendation

Artifacts with similar types and usually manipulated with similar applications, are grouped together. A group is visually identified by composing icons from its most important applications. Every group contains a maximum of seven artifacts ordered by importance from bottom to top. In order to take advantage of the user's spatial memory, these groups are positioned in a same order according to their importance for the SE work from left to right.

For a quick overview of recommended artifacts each artifact is displayed using a preview image of its content. Every artifact of currently selected group has a bigger size and includes a title. Besides special functions enable the engineer to fulfill common tasks directly from Switch!, like creating a new class or searching in the address book.

The user can move a selection along the offered elements either by hovering over the elements with the mouse or by using the keyboard. The selection can thereby be moved between the groups or between the recommended group artifacts. The artifact being active before starting *Switch!* is not displayed as switching the same artifact is senseless. The engineer then either clicks the needed artifact or presses the enter key while it is selected. The *Switch!* window is hidden and the artifact is shown in front of other windows.

4.3 Framework and Architecture

Switch! is build on top of the knowledge sharing framework *TeamWeaver* [MH08]. The two main layers of *TeamWeaver* we are using are the *Context System* and the *Distributed Knowledge Model* layer. The *Context System* layer monitors and interprets the engineers' behavior while the *Distributed Knowledge Model* layer stores and exchanges the behavioral model. An architecture overview is shown in Figure 3. We shortly describe the main components and how they are used inside *Switch!*. The *TeamWeaver Ontologies* describe the semantic model of tasks and artifacts and the relationships to each other. The ontologies incorporate the task semantics. The *Context Monitor* contains various sensors to existing tools and information sources - such as email clients, web browsers and development environments. These sensors monitor the raw interaction events triggered by the engineer and their semantic representations. This component builds the personal interaction history. The *Context Interpreter* then classifies the current task based on the interaction history data monitored by the *Context Monitor*. The *Local Metadata Store* manages metadata about the artifacts and the tasks of the engineer. In order to use the social aspects, in particular the sharing of community profiles the *P2P Infrastructure* component is used. It enables the sharing of own interaction history data, as well as the receiving of community profiles shared by co-workers. The *Profiler* component creates the own profile to share and aggregates received ones. *Switch!* uses the collected and analyzed interaction data provided by the *Context System* and integrates it with the universal assumptions and the metadata provided by the *Distributed Knowledge Model*. The result is used to create the recommendation by selecting and ranking suitable artifacts.

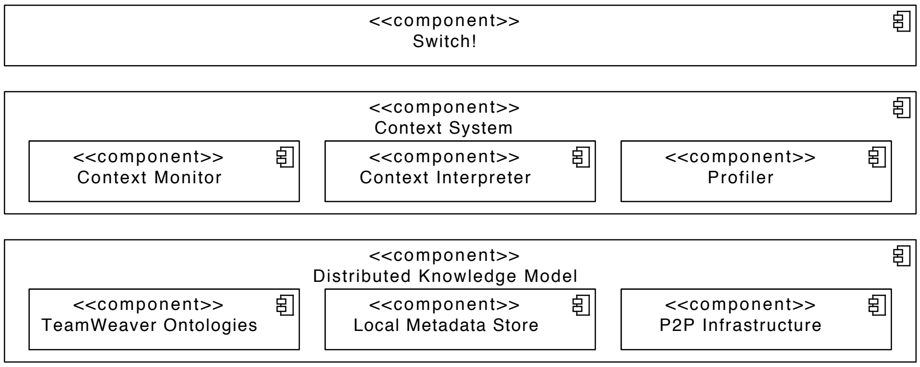

Figure 3: Switch! architecture overview

5 Evaluation

We present the results of a paper prototype study conducted to evaluate the *Switch!* concepts and user interface. Then we discuss the prototypical implementation.

5.1 Paper Prototype Study

To evaluate our recommendation approach, we conducted an experiment with experienced software engineers by using a paper-based prototype. This prototype was created based on the user interface concepts shown in Figure 2.

The elements of the user interface are separately printed on paper to simulate needed combinations. If a participant selects an artifact the appropriate screenshot is presented to simulate the desktop environment. The prototype is presented by the experiment supervisor on a table, with the participant sitting in front. The supervisor simulates the recommendations depending on the participant input and the algorithm introduced above. Each experiment session is recorded by a video camera to track the physical actions and participants' verbal reactions. Participants should use the paper prototype, as they would work in their everyday environment. They should "think loudly" [Sea99] and express in words what they are doing and why. Each participant has to perform one implementation, one bug fixing and one documentation tasks. The concrete task instructions can be downloaded from the teamweaver site[2].

Since Switch! is developed for *Mac OS X* we focused on engineers primarily working on this platform. We recruited seven engineers from a Munich-based company specialized on *Mac OS X* development and from the TUM Mac development Lab. While *XCode* is used as main IDE by 2 participants, 5 participants mostly use *Eclipse*. In average each simulation session lasts 40-60 minutes including the procedure explanation and discussion.

Six participants claimed that with current environments, is difficult to keep an overview of the needed artifacts and distinguish them from the ones not needed next. Our solution presented to them was rated positively. Five stated they would use a tool with the presented functionality. We observed that all seven participants were not sure what to do and how the artifacts are arranged, when introduced to the *Switch!* user interface. But as they started pointing with the mouse or pressing the cursor keys, the concept of interaction was understood by all of them. It took participants in avg. 4-5 min. to get familiar with the interface. Participants identified the instant magnification and tool reaction to be very useful in such situation. Five engineers recognized the artifacts they should select very quickly by their icon (< 2 sec.). The other two needed to take a closer look at the titles or asked the supervisor for help. Participants did not understand the recommendation criteria quickly. Four participants asked the supervisor several times why the particular artifacts are recommended. Three engineers asked what to do if the needed artifact is not included. Two engineers stated artifact switching using *Switch!* would be too complicated for daily use. These two engineers exclusively use Eclipse during the work.

Overall the purpose of *Switch!* and its features were quickly understood. But the reason why a particular artifact is recommended is not sufficiently understood by the engineers. Additional graphical presentation of most relevant relationships and recommendation criteria might considered. Providing a detailed rationale for a recommendation would increase confidence in the recommendation, but increases the information overload as well [RWZ10]. Furthermore engineers wanted to be able to select artifacts not recommended by *Switch!* if necessary. During a discussion at the session end, participant mentioned a) the accuracy of the recommendation, b) performance and c) the possibility to customize artifact groups as additional requirements.

[2] http://www.teamweaver.org/wiki/index.php/switchExperiement

5.2 Prototypical Implementation

The current version of *Switch!* uses the described algorithm to create, visualize and recommend the following artifact types: source code, PDF files, *XCode* documentation, emails, web pages and plain text files. When hitting a global keyboard shortcut the main switching window is shown. It includes (both open and closed) artifacts identified by *Switch!* to be relevant. The artifacts are categorized into the tool groups and sorted inside the groups according to calculated ranks. The preview images are created using the *Quicklook* service of *Mac OS X*. *Quicklook* by default has the ability to create preview images from many different file formats. In this version *Switch!* supports the task types bug fixing, testing and documenting. For well-defined tasks with a limited set of possible artifacts, the current version provides suitable recommendations. We are using *Switch!* when dealing with several bug fixing and testing tasks at once.

6 Related Work

Several related systems have been proposed to address engineers information needs as well as the encountered information overload. These can roughly be separated into three categories: Application Launchers, Recommender Systems for software development and Window Switchers.

Application launchers like *Quicksilver* [qui] and *LaunchBar* [lau] for the *Mac OS X* platform or *Launchy* [Kar] for *Windows*, enable the user to quickly start applications, open files or execute predefined commands. These tools usually work by pressing a shortcut and typing short queries containing a few letters. The results of possible documents to open or tools to launch are immediately listed. Frequently used commands are rated higher for the next similar query (equal prefixes). Application launchers allow users to efficiently execute common tasks (e.g. open document, send email). However, these tools do not consider users' context. Users still need to know exactly the artifact they want to switch to, its name and its path.

Recommender Systems for software engineering offer a more advanced functionality. Their goal is to address the current information needs of the engineer by recommending suitable artifacts [RWZ10]. A landscape of recommendation systems used in software development is described in [HM08]. *Dhruv* [ASH06], for example, is a tool that presents information on a bug report taking into account the information given by the bug report itself. *Mylyn* [KM06], hides not needed classes and methods from the Eclipse IDE, by considering previously selected and edited ones. Other recommendation systems such as *Malibu* [Sh08] and *TaskTracer* [DDJ05] are not tightened to software development. They recommend windows depending on the current user activity. Thereby, the activity needs to be recorded manually. *TaskPredictor2* [Sh09] applies machine learning mechanisms to detect unrecorded activities.

Operating systems often use open windows as basis for switching artifacts, assuming that windows represent artifacts. They are designed to offer the open windows in a way the user is able to find the needed ones more easily. The *GroupBar* [Sm03] assigns several windows to one task. *SWISH* [Ol06] groups similar windows by using the semantics of the titles and interaction history. A similar approach is implemented by *Taskposé* [BSW08] which groups miniature images of the open windows according to their relatedness. Artifacts not represented by a single window - such as a method of a class - cannot be considered by these systems.

7 Conclusion

Driven by the problems software engineers face when working simultaneously on multiple tasks and using a large number of artifacts, we designed and implemented a context-aware artifact switching tool, called *Switch!*. First evaluations based on simulations and short usage sessions are promising. Our approach seems to protect engineers from "interruption noise" and reduces the time needed for locating and reproducing needed information. The current version is able to handle source code documents, PDFs, emails and web pages. For the next step we plan to evaluate *Switch!* in a large industrial setting, by measuring saved context switching time. For this *Switch!* requires refinements and extensions to other artifact and task types. At the same time we use this earlier version to identify further heuristics and fine-tune the recommendation approach. Future directions are to investigate graph based artifact relationships modeling for work-flow matching and to integrate community aspects by accessing social media platforms.

References

[ASH06] Ankolekar, A.; Sycara, K.; Herbsleb,J. et al.: Supporting online problem-solving communities with the semantic web. In Proceedings of the 15th international conference on World Wide Web, 2006.

[BSW08] Bernstein, M.; Shrager,J.; Winograd,T.: Taskposé: exploring fluid boundaries in an associative window visualization. In Proceedings of the 21st annual ACM symposium on User interface software and technology, 2008.

[DDJ05] Dragunov, A.; Dietterich, T.; Johnsrude, K.: TaskTracer: a desktop environment to support multi-tasking knowledge workers. In Proceedings of the 10th international conference on Intelligent User Interfaces, 2005.

[GM04] González, K.; Mark,G.: "Constant, constant, multi-tasking craziness": managing multiple working spheres. In Proceedings of the SIGCHI conference on Human factors in computing systems, 2004.

[HM08] Happel, H.J.; Maalej, W.: Potentials and challenges of recommendation systems for software development. In RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering, 2008.

[Kar] Karlin, J.: Launchy. http://launchy.net.

[KDV07] Ko, A.; DeLine, R.; Venolia, G.: Information needs in collocated software development teams. In Proceedings of the 29th international conference on Software Engineering, 2007.

[KM06] Kersten, M.; Murphy, G.: Using task context to improve programmer productivity. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering, 2006.

[lau] LaunchBar. http://www.obdev.at/launchbar/.

[Ma09] Maalej, W.: Task-First or Context-First? Tool Integration Revisited. In Proceedings of the ACM/IEEE International Conference on Automated Software Engineering. IEEE Computer Society, 2009.

[MGH05] Mark, G.; Gonzalez, V.; Harris, J.: No task left behind?: examining the nature of fragmented work. In Proceedings of the SIGCHI conference on Human factors in computing systems, 2005.

[MH08] Maalej, W.; Happel, H.: A lightweight approach for knowledge sharing in distributed software teams. In Proceedings of the 7th International Conference on Practical Aspects of Knowledge Management, Jan 2008.

[MH09] Maalej, W.; Happel, H.: From work to word: How do software developers describe their work? In Proceedings of MSR ’09. 6th IEEE International Working Conference on Mining Software Repositories, 2009.

[Ol06] Oliver, N.; Smith, G.; Thakkar, C.; Surendran, A: SWISH: semantic analysis of window titles and switching history. In Proceedings of the 11th international conference on Intelligent user interfaces, 2006.

[qui] Quicksilver. http://docs.blacktree.com/quicksilver/what_is_quicksilver.

[RWZ10] Robillard, M.; Walker, R.; Zimmermann, T.: Recommendation Systems for Software Engineering. IEEE Software, 2010.

[Sm03] Smith, G.; Baudisch, P.; Robertson, G.; Czerwinski, M.: GroupBar: The TaskBar Evolved. In Proceedings of OZCHI, 2003.

[Sea99] Seaman, C.: Qualitative Methods in Empirical Studies of Software Engineering. IEEE Trans. Softw. Eng., 25(4):557–572, 1999.

[Sh08] Shen, J.; Geyer, W.; Muller, M.; Dugan, C.; Brownholtz, B.; Millen, D.: Automatically finding and recommending resources to support knowledge workers’ activities. In Proceedings of the 13th international conference on Intelligent user interfaces, 2008.

[Sh09] Shen, J.; Irvine, J.; Bao, X.; Goodman, M.; Kolibaba, S.; Tran, A.; Carl, F.; Kirschner, B.; Stumpf, S.; Dietterich, T.: Detecting and correcting user activity switches: algorithms and interfaces. In Proceedingsc of the 13th international conference on Intelligent user interfaces, 2009.

[ZG06] Zou, L.; Godfrey, M.: An Industrial Case Study of Program Artifacts Viewed During Maintenance Tasks. In Proceedings of the 13th Working Conference on Reverse Engineering, 2006.