Design Guidelines for Ambient Software Visualization in the Workplace

Chris Parnin and Carsten Görg*
College of Computing
Georgia Institute of Technology
{vector,goerg}@cc.gatech.edu

Abstract

Success in software development dictates that the right information reaches the right people. In the development process, information flows among developers, managers, and customers. The information is represented in a multitude of sources: customer requirements, application domains, product specifications, development processes, schedules, budgets, project status reports, and task priorities. Unfortunately, in the transmission of information, vital tidbits are filtered away, made inaccessible, or withheld from the stakeholders. Software visualization systems have traditionally focused on the complexities of the relations and detailed structure of the software artifacts. Recently, attention has been given to visualizing software artifacts from the perspective of supporting teams in coordinating efforts. In this paper, we describe the nature of this information source and provide design guidelines for developing ambient software visualizations in the workplace. In particular, we describe how developers can better understand more about project management, recall and perform tasks in their personal work flow, and coordinate project state that is continually changing.

1 Introduction

With rapidly advancing display technologies, new opportunities to alter the workplace environments for software developers emerge. Two major problems that have been identified in the software developer workplace are (1) the negative impact of distractions and interruptions and (2) failures in communicating and difficulties in understanding the project as a whole.

The responsibilities and tasks assigned to professional software developers require deep concentration in an environment constantly vying for their attention. Distractions are paramount in this environment; meetings, emails and phone calls fragment the usable work hours. Many tasks require coordinating with another team member, interrupting the current task of the assisting member. In this atmosphere, it is not unusual for developers to fail to recall the need to perform a specific task, known as *prospective memory failure*. In addition, productivity is negatively affected: Czerwinski's study [3] showed that tasks which resumed after interruption were more difficult to perform and took twice as long. O'Conaill's study [15] found 40% of interrupted tasks are not resumed at all. Further research [13] observed 57% of tasks were interrupted, and the time spent working uninterrupted was very small. A recent study [12] of software engineers working at Microsoft indicated that 62% of developers surveyed believed recovering from interruptions was a serious problem.

Opportunities for breakdowns in communication are possible across numerous boundaries [11]. A large divide exists in communicating between managers and programmers. The reasons for this are several: technical inexperience, differing personality types, team structures, and personnel removal. Moreover, programmers are often buffered from upper management concerns, budget constraints, schedule concerns, and task priorities.

In the software visualization community, several attempts have been made to address some of the above concerns. The `WAR ROOM` command console [16] is a shared visualization system that provides a view of a team's activities. The system highlights individual team efforts in order to report progress to management, identify potential conflicts, and share insight of the entire project. The success of this approach comes from the overview of the current project efforts; however, the visualization remains largely developer-centric and source code centric. Ripley *et al.* [21] show another instance where a 3d visualization is used to monitor the artifacts and developer's software configuration management. The system `AUGUR` [8] provides a `SeeSoft` [6] view to juxtapose source structure with activities on that structure. The authors note that the system is not designed for managers.

* The author was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

The above systems focus mainly on visualizing data from source control repositories. Source control repositories reflect only a small subset of the information sources that developers must deal with. In addition, the design of these systems assume that developers have the time and attention to devote to these tools without understanding the context of how these activities occur.

Ambient and peripheral visualization systems are an alternative to the highly interactive and exploratory tool paradigm. The term *ambient* visualization has been used to refer to many applications: from glowing ambient orbs [2] casting the stock market's daily movement in green and red hues to plasma displays hung in public spaces streaming weather, event announcements, and traffic information.

A connection needs to be made with the philosophy of ambient visualizations and the problems they contend to address. Humans pursue assorted interests in their lives. Some interests and information exist beyond the range of their sphere of attention. In a world where only limited resources and time are available, ambient conveyance of information connect people to the fringe of their awareness with minimal interruption. The displays accomplish this through the blending of aesthetic qualities with environment and low-key information conveyance.

We discuss the use of ambient displays in general office applications in the next section, followed by an explanation of how these displays can be arranged and operated in a developer's office in Section 3. We describe how information flows throughout the software process and how incorporating content from these sources promote awareness and facilitate information flow in Section 4. Finally, we suggest design guidelines for managing the scheduling and representation of this information in Section 5 and draw conclusion in Section 6.

2 Ambient Displays in the Workplace

The design issues involved in ambient displays that foster information awareness and collaboration in the workplace has received wide attention in the HCI community. Huang and Mynatt [10] have described these efforts as they vary along two dimensions: the target *group size* (pairs, small groups of about 10, and large groups sized 20 to over 100), and the location of the *display space* (personal, collocated, and public). McCarthy *et al.* have described systems as being either *UniCast*, *OutCast*, or *GroupCast* [14]. In UniCast, information is displayed for personal purpose in the office location. OutCast systems can be placed outside the office door to project information such as biographically information, current projects, and in or out of office status. In GroupCast, information is shared among a larger group.

In large groups, general information is broadcast to individual desktops or public displays. The information caters to general awareness of office events, weather, traffic, etc. These systems have difficulty in providing directly relevant content due to the size of the intended audience. Approaches such as GroupCast limit information scope, whereas systems such as THE BUZZ [5] address this problem through personalization of the broadcast information.

Other systems have attempted to tailor display design toward smaller groups. The NOTIFICATION COLLAGE [9] serves as a digital bulletin board allowing posting of general events, photos, interesting news articles, and for-sale ads in the office environment. Huang and Mynatt [10] steer away from general office applications and describe how semi-public displays can support collocated groups by sharing information about personnel presence and absence, work progress, participation in non-work activities, upcoming work events, and solicitation for short-term assistance and long-term collaboration. The MessyBoard [7] also focuses on groups by supporting the sharing of project ideas and notes in a common digital space.

Other systems have catered toward the individual and supporting the performance of their tasks. TASKTRACER [4] enables workers to switch among and maintain different working contexts by tracking documents, email messages, and visited websites under various contexts. Similarly, KIMURA [22] uses a peripheral display to hold montages of artifacts that are associated with different working contexts. The system allows users to switch among and make notes about the contexts.

Although this body of work has demonstrated the efficacy of ambient visualization in the workplace, the content of these systems generally focus on office activities that foster a better workplace environment. Other systems do partially address task-relevant issues such as maintaining working memory, but do not leverage the nature of the work or project domain.

3 The Ambient Development Display

3.1 Display Arrangements

Workplace arrangements vary greatly across different offices and organizations. One developer may have a cubicle with limited wall space or have physical limitations in mounting displays. Other developers may have their own office, whereas some might share an open lab space.

In these assorted configurations, the placement of an external display and the resulting effects warrant interest. For desks placed against the wall, one possible configuration involves a multi-monitor setup of dual or triple screens and a large display placed above the monitors. This configuration allows the developer to work with their main monitors while enabling passive viewing of information on the large display (see Figure 1). In this configuration the question

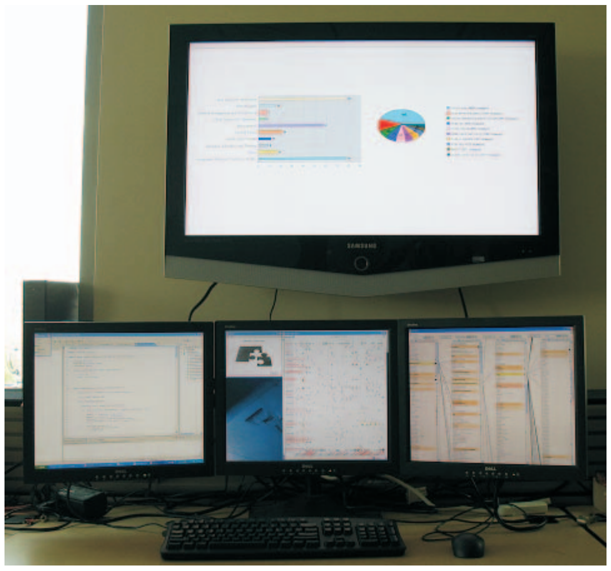

Figure 1. Multi-monitor setup.

arises: is the extra information on the large monitor distracting? In Plaue's study [18] of students performing tasks, it was found that smooth animations did not distract or interrupt the performance of the task. For desks facing outward, placement may be either on the nearest opposite wall or farthest adjacent wall. The distance from the desk requires that information viewed in this configuration cannot be too text or detail-oriented. Some offices share a common atrium, allowing the placement of a display in the common area. This configuration is similar to GroupCast [14] and semi-public displays [10] where a collocated group of people can share one community display.

3.2 Display Operation

The operation of the display can be described in two modes:

disengaged. The display is operating in disengaged mode when the developer is not actively focusing on the display. In this mode, information is passively presented and rotated in an ambient mode.

engaged. The display is operating in an engaged mode if the developer is using the display for an active purpose such as view information in a conference call, display diagrams for group discussions, or serve as a supplemental view area for pixel-intensive visualizations and applications.

The default operation of the display is in disengaged mode. It may be possible to allow some control in the event that interesting information is presented and the developer needs to pause, or rewind the display. In this situation, the developer could use an external input device such as a knob control or a control program to pause, or navigate forward or backward. The display can be *engaged* by a control program or by dragging content onto the display space when using multiple monitors.

4 Information Sources

If a work environment had a display that could give information about the project and workplace, what information could it display? What processes, artifacts, and source of information would be appropriate?

Pressman describes the components that exist in the software development workplace [20]: "Effective software project management focuses on the four P's: people, product, process, and project. The order is not arbitrary." To this we add *place* as the fifth *P*, the environment in which these activities occur. We argue that complexity of software development does not necessarily derive from the individual components but also from coordinating the efforts among distributed resources while operating under the constraints of time and cost.

Figure 2 shows the information flow between the different aspects of software development.

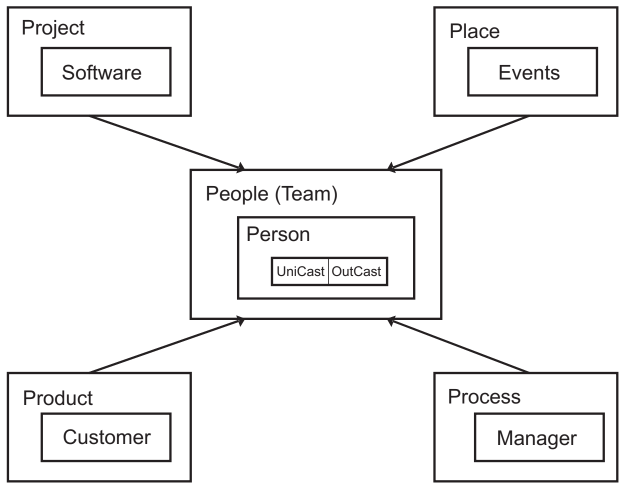

Figure 2. Communication information flow.

The people in the team developing the product must grapple with maintaining a shared vision and goals constructed from various external sources of information. At the personal level, a person must manage their tasks to avoid prospective memory failure and negatively affecting their team members.

The product is the envisioned outcome that addresses some customer's need. It includes the application domain,

Product	Process	Project	People (Internal)	People (External)	Place
user stereotypes	schedule/tasks	build status	class-context	performance fixed	lunch
use cases	budget	test failures	testing checklist	someone review this	meetings
domain diagrams	issues/risks	quality metrics	be nice to	new db protocol	announcements
concept prototypes	deadlines	bad smells	block released	need help	cookies

Table 1. Sample artifacts for different content classes.

the requirements, and the end-user's attributes. The information about the product comes to the development team in form of requirements, user-centered documents, or interface designs.

The process is the collection of methodologies that are employed in solving problems and realizing the product (including code review strategies, test plans, and development techniques). The manager often coordinates these efforts and allocates the task to the team.

The project is what is being built: the software and all the related artifacts used in its construction, (*e.g.* test cases, quality assurance metrics, source code management). Information about the project and its status often flows informally amongst the team: (through emails, and face-to-face communication) whereas information flow to management occurs through solicitation or status meetings. Systems such as `WAR ROOM` [16] have tried to facilitate communication to management but the opposite direction has not been addressed.

In development environments, gaps and disconnect between developers and these information sources commonly occur. Actively seeking this information is hampered by the people's inability to digest information that is outside of their expertise and from the sheer mental burden of managing their personal work flow.

We argue that providing a mechanism for digesting various sources of information from people, product, process, project, and place in a manner that is accessible but non-intrusive will alleviate the following information concerns:

1. Make decisions in the light of constraints and maintain a shared vision.

2. Manage coordination of the developers working on the project.

3. Bridge the gap in communication through feedback.

4. Assist the developer in managing their own personal work flow.

In the next subsections, we visit the different information concerns and describe how information contained in various content classes alleviates these concerns. Table 1 lists a summary of artifacts that contain interesting content for each information component. This section is intended to foster discussion on the space of possible applications that researchers should investigate for ambient software visualization.

4.1 Constraints and Vision

Developers are often shielded from information and decisions that occur in process management and product design. In some sense, being buffered from non-essential information and red-tape allows developers to focus on their work. However, there is an overall benefit if developers are more informed about the constraints they are operating under and if developers share a common vision.

Even the best developers can get carried away sometimes. In the pursuit of solutions, features creep in and code mavericks run-off on tangents. These developers are loosing focus on the goal; often because the goal is not clearly expressed in a tangible manner. One way to assist these wayward spirits is to keep developers in perspective: where are they now in development, how much work is left, how much money has been spent, how much money is left, what are the top priorities and issues? Engineers need to make decisions in the light of constraints, but much of this information is in the managers hands and not easily communicated to developers.

Developers are disconnected from their customers. They will often never meet or interact with a customer. The managers divide and allocate tasks that senior developers or technical leads further distill into manageable chunks for junior developers to digest. By the time developers receive these morsels, they have lost all the original essence, and developers have no sense that ultimately their contribution intends to solve some customer or end-user need. To counter this vision atrophy, visualizations that represent the final product outcome and user environment keep developers focused on the outcome. If the developer is creating some operational flight plan software for pilots, then diagrams of the airplanes, and views from inside the cockpit can be displayed to inspire the developers in how their product fits into the big picture and remind them of what they are working toward.

4.2 Coordination

Despite the efforts to distribute work in independent subtasks, development can be halted when central files are locked, or sequential dependencies in task occur. Furthermore, knowledge and expertise is also distributed throughout the team. Ambient visualization systems are suitable for assisting teams in broadcasting announcements and monitoring project information.

When users *outcast* information, they want to make announcements to the team about their personal contributions (*e.g.* update of a protocol or the fix of some long-standing bug/performance issue), coordinate and share design decisions, ask for help on an issue [10], or request a quick code review. The scope of this information is most appropriate for the team and can be applied even when the ambient display is in a collocated area.

If there are several conditions developers are waiting for, then a system may monitor for changes in these blocking conditions and notify the developer when the condition has changed. A *continuous integration* system is an appropriate solution for monitoring project state. Continuous integration is a process philosophy that incorporates integration and testing throughout development. This practice aims at reducing integration problems by having developers integrate frequently, usually daily. Automation of this process can be achieved with a system that continually builds the latest software and runs tests.

The build philosophy can be taken further; instead of just integration and testing, the continual build cycle can be seen as ensuring continual quality and provide metrics for process management. For example, one continuous integration system might monitor changes to source control and then ensure that it builds, updates auto-generated documentation, update code metrics, and run unit and system tests.

build status. Includes information about the current build: if the build is broken, who recently made changes, etc.

quality status. Has the code/comment ratio drastically gone down for a class? Has the size of code increased too much? Has the check-in frequency changed? What anomalies appear? etc.

test status. Have any unit tests failed? For which modules? Who is the author related to the code?

bad smells. Are there any bad smells present in code such as long method, middle-man, code duplication, etc.?

4.3 Feedback

When information flows in one direction, there is an opportunity to provide feedback in response to the opposite direction. Feedback may occur with product design decisions, process management, and implementation choices of other developers.

For example, managers typically worry about regulating work focus, time, and cost in addition to making risk estimates and assessments. This information is later communicated to upper management. The health of the project benefits from managers having a honest and insightful view into the state of the project regardless how they communicate this to the next level of management. However, in performing risk management, which requires identifying risks and giving estimates of the likelihood and impact of unfortunate events, managers may have difficulty developing real insights into the problem because of their distance from the work. Communicating the manager's assumptions about risk to developers has several benefits. The developers have a role in providing a sanity check on estimations given to upper management and a developer's understanding of project status deepens.

4.4 Personal Task Management

During software development, developers generate ideas of what they would like to do but cannot do at that moment (*e.g. investigate a possible bug*) resulting in failure to recall this insight. Additionally there are many tasks that are either waiting on someone else (blocked at the moment) or that require performing many time consuming actions. Without a structured method that supports keeping track of all these things, mismanagement and failure in performing tasks often arises.

In "Getting Things Done" [1], Allen provides some useful insight into how to take care of business and manage future actions. When confronted with thoughts like, "It would be nice to use this concept when *undo* gets implemented later on", "I need to bring up the database connection issue at the meeting", or "Making this change might break the preference dialog, I should test that" a scheme is needed to help keep track of these thoughts and remind the person of them at the appropriate time. Problems where no actions can take place until a later date are difficult to record and recall in a timely fashion.

One approach to improve the 'like-to' situation is to maintain a *tickler* file. An implementation of a tickler file is as simple as keeping twelve vanilla folders, one for each month, where the person can refer to the file at the beginning of each month and recall all the things they wanted to investigate at that date. This concept can be extended to the software development world where developers record thoughts and set reminders for some time in the future. The reminders are displayed on the ambient display near the appropriate time and trigger those insights when they are more actionable.

The ability to perform actions is strengthened when the appropriate context is made available. By clustering actions around the context in which all the tools and resources are available to perform the action, a person only has to worry about the actions available in a context and can be reminded when that context occurs. A developer may be heavily involved in making changes to the software when a thought occurs, "one of the changes I made may break another feature"; however, the software is not in a runnable state and may not be until a couple of days later. The action 'test feature' cannot occur until the context 'software runs' is present. With an ambient system, the developer records that they want to test that feature in the 'software runs' context. Later on, when the build is stable, the ambient system reminds the developer to test the feature which did indeed break.

There are further ways to define context by target content and interaction mode. For instance, when actively editing a class, any issues related to this class can belong to the 'class context'. Working on different tasks, bug reports, or features could trigger other reminders for these contexts. The other dimension involves the operating mode (either debugging, testing, error message, or wrong input). In an operating mode, if a certain error message reoccurs then the ambient system can complement this event with information from previous instances when this occurred or notes from another team member.

4.5 Other Concerns

The realities of the development environment stipulate that people will need to take lunches, companies have internal events, and details outside of the scope of the project affect the performance of developers. At one company, some people discuss plans for lunch on the 'Lunch Train', an internal message board devoted to lunch plans. Similarly, team members note their lunch engagements if they want to invoke some interest. Several previous research systems have explored using ambient visualizations specifically for office events and non-work related activities. Uses in conveying information such as "in/out of office" status, corporate events, and participation in non-work activities have been deployed. Even in a project-focused application, these uses are warranted in that they situate the ambient device as something both personal and integrated into work, not a big-brother 'telescreen' that pressures employees into constant state of fear of being monitored and regulated.

5 Design Guidelines for Ambient Displays

When researchers and future designers create ambient views for software development environments three essential problems exist: (1) the scheduling and timing of the content, (2) extracting the necessary details from an information source, and (3) choosing the appropriate notification level.

5.1 Scheduling Content

Content can be either event-based or non-event based. The following are design principles in sharing information concerning events.

The law of volatility. Infrequent outliers, a change, or spike in value are more interesting when they occur in contrast to relatively unchanging data. If the build often breaks, then employing high levels of notification to inform the developer that the build has broken will just become annoying. If the unit tests rarely fail but in this case have, then higher levels of notification are justified.

The law of timeliness. Information is more interesting when it has *locality* or co-occurrence with an event. This may be before or after the event. The right time to remind someone about the agenda they wanted to present at a meeting is just before the meeting.

The law of freshness. The value of information decreases in proportion to the underlying data's volatility. The appeal of a topic diminishes with time, especially if the relevance of the facts are deemed unreliable.

Event-based content is scheduled using volatility and timeliness. Non-event content should occur in between the event-based content in a manner that maximizes the freshness of the content.

5.2 Representation

The design choices in representation affect the developer's perception of the system, their understanding of the information, and the success of integration with the developer's work flow. Pousman and Stasko [19] have identified four dimensions prevalent in designing ambient visualization systems:

information capacity. The amount of information displayed with the system.

notification level. The extent of interruption on the user.

representational fidelity. The faithfulness of the visualization in representing the original information.

aesthetic emphasis. The level of artistic design in representation.

Different content classes of information (product, process, project, people, and place), require different considerations in representing a visualization. In Figure 3, we suggest how the design dimensions vary with different content classes.

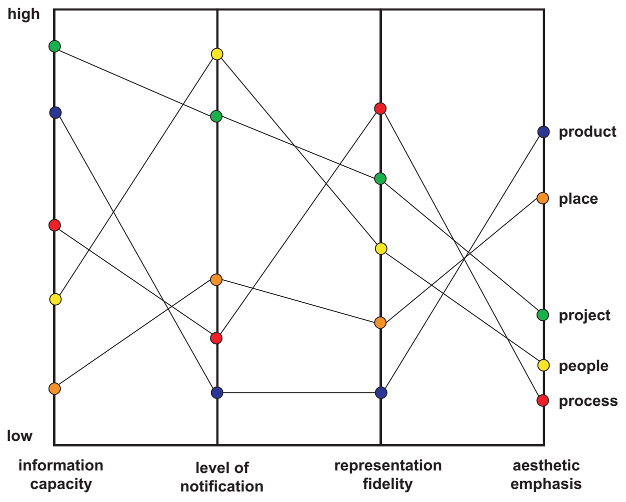

Figure 3. Suggested design requirements for different content classes.

For instance, place content such as a lunch announcement does not require high levels of information capacity; it may only need a few words or text and time information. In contrast, process content may require a higher information capacity. Consider the scenario where a *user stereotype* (a design technique that creates a fictional user type with some background information, preferences, and common activities) is displayed. This information requires higher information capacity than desired on the ambient display system. In this scenario, it is common to abstract the information by adjusting the representational fidelity or artistic expression. By creating user caricatures with icons representing different actions and information needs, developers are reminded of different user types without having to read through the text.

The content classes do not necessarily generalize all the design decisions. The complexity of individual information sources may require a higher-level of representation fidelity. With process content, on one hand scheduling information needs time and task relations to be preserved, whereas a view of budget information may discard certain details as long as the point is driven across.

Source code components present the most interesting challenges in representation. In coordination efforts, the representation of source content may be restricted to file or method names that are under conflict. This has been the approach taken by most software visualization systems. However, in displaying code metrics, potential security violations, or possible code smells, a more detailed code visualization is warranted. Approaches such as light-weight visualizations for code smell inspection [17] provide more context with task-oriented visualizations.

5.3 Notification

With the ambient system, the notification of timely information can be done in such a way that the developer is not disturbed from their current task. Unlike other reminder systems, the display is capable of presenting information without popping up message boxes or balloon notifications. The information is available when the developer actively decides to refer to the display.

But how much further should the system attempt to acquire attention? Researchers in ambient visualizations systems have employed various animation, ticker, acoustic, and lighting techniques for acquiring attention. Information such as contextual reminders may justify higher levels of notification, especially if the display is currently operating in the engaged mode. Ultimately, the spirit of ambient visualization systems suggests that notification should be considered low by default and then configured to acceptable thresholds of tolerance.

6 Conclusion

In this paper we have characterized information flow as part of the software development process. Several systems have addressed a subset of this information flow – only the flow from project state to management and from project efforts to people has been thoroughly studied. We have illustrated several information sources and artifacts that are of interest in this communication model. We present a new paradigm for incorporating visualizations that reflects the complexity of the software development environment. Many software visualizations are heavy-weight and require exploration and direct interaction with users. We advocate light-weight visualizations that highlight insights throughout the software development process that have minimal impact on the developer's work flow. We propose that the use of ambient visualization systems will serve these needs.

Several systems have demonstrated the effectiveness of ambient visualization systems in general office settings. Future work in the context of software development would generalize these results for knowledge-intensive work. Enrichment of this work could develop new content classes and examine the nature of information flows within organizations and teams. Several issues that are unique to software development need to be addressed. For example, the role of personalizing and configuring levels of notifications, representations, content, and content scheduling needs to be further explored.

Acknowledgments. Spencer Rugaber, James Eagan, and our anonymous reviewers gave helpful comments on earlier revisions of this paper.

References

[1] D. Allen. *Getting Things Done: The Art of Stress-Free Productivity*. Penguin, 2002.

[2] Ambient orb (ambient devices). `http://www.ambientdevices.com/cat/orb/orborder.html`, 2007.

[3] M. Czerwinski, E. Horvitz, and S. Wilhite. A diary study of task switching and interruptions. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182, New York, NY, USA, 2004. ACM Press.

[4] A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. Mclaughlin, L. Li, and J. L. Herlocker. Tasktracer: a desktop environment to support multi-tasking knowledge workers. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 75–82, New York, NY, USA, 2005. ACM Press.

[5] J. R. Eagan. Designing interfaces to enrich personalization. In *DIS '06: Proceedings of the 6th ACM conference on Designing Interactive systems*, pages 350–351, New York, NY, USA, 2006. ACM Press.

[6] S. G. Eick, J. L. Steffen, and J. Eric E. Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Trans. Softw. Eng.*, 18(11):957–968, 1992.

[7] A. Fass, J. Forlizzi, and R. Pausch. Messydesk and messyboard: two designs inspired by the goal of improving human memory. In *DIS '02: Proceedings of the conference on Designing interactive systems*, pages 303–311, New York, NY, USA, 2002. ACM Press.

[8] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 387–396, Washington, DC, USA, 2004. IEEE Computer Society.

[9] S. Greenberg and M. Rounding. The notification collage: posting information to public and personal displays. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 514–521, New York, NY, USA, 2001. ACM Press.

[10] E. M. Huang and E. D. Mynatt. Semi-public displays for small, colocated groups. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 49–56, New York, NY, USA, 2003. ACM Press.

[11] H. Krasner, B. Curtis, and N. Iscoe. Communication breakdowns and boundary spanning activities on large programming projects. *Empirical studies of programmers: second workshop*, pages 47–64, 1987.

[12] T. D. Latoza, G. Venolia, and R. Deline. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 492–501, New York, NY, USA, 2006. ACM Press.

[13] G. Mark, V. M. Gonzalez, and J. Harris. No task left behind? Examining the nature of fragmented work. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–330, New York, NY, USA, 2005. ACM Press.

[14] J. F. McCarthy, T. J. Costa, and E. S. Liongosari. Unicast, outcast & groupcast: Three steps toward ubiquitous, peripheral displays. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 332–345, London, UK, 2001. Springer-Verlag.

[15] B. O'Conaill and D. Frohlich. Timespace in the workplace: dealing with interruptions. In *CHI '95: Conference companion on Human factors in computing systems*, pages 262–263, New York, NY, USA, 1995. ACM Press.

[16] C. O'Reilly, D. Bustard, and P. Morrow. The war room command console: shared visualizations for inclusive team coordination. In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 57–65, New York, NY, USA, 2005. ACM Press.

[17] C. Parnin and C. Görg. Lightweight visualizations for inspecting code smells. In *SoftVis '06: Proceedings of the 2006 ACM symposium on Software visualization*, pages 171–172, New York, NY, USA, 2006. ACM Press.

[18] C. Plaue and J. Stasko. Animation in a peripheral display: Distraction, appeal, and information conveyance in varying display configurations. In *GI '07: Proceedings of the 33rd conference on Graphics Interface*, 2007. to appear.

[19] Z. Pousman and J. Stasko. A taxonomy of ambient information systems: four patterns of design. In *AVI '06: Proceedings of the working conference on Advanced visual interfaces*, pages 67–74, New York, NY, USA, 2006. ACM Press.

[20] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2001.

[21] R. Ripley, A. Sarma, and A. van der Hoek. Using visualizations to analyze workspace activity and discern software project evolution. Technical report, University of California, Irvine, 2006.

[22] S. Voida, E. D. Mynatt, B. MacIntyre, and G. M. Corso. Integrating virtual and physical context to support knowledge workers. *IEEE Pervasive Computing*, 1(3):73–79, 2002.