A Call for the Use of Display Technology to Support Software Development

Alex Baker, Ping Chen, Christopher Van der Westhuizen and André van der Hoek

Department of Informatics
Donald Bren School of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA

{abaker,pchen,cvanderw,andre}@ics.uci.edu

Abstract. The discipline of software development is ripe for improvement via the use of new display technologies. We feel that by providing software developers with multiple personal displays, and large-scale shared displays, the process of software engineering may be greatly enhanced in more than one way. In this paper, we present several specific approaches to using display technology to facilitate software design and raise numerous related issues for continued research.

1 Introduction

As display technology has improved and large-scale displays have emerged as viable, computer scientists have worked to apply this kind of technology in a variety of situations. Ubiquitous display environments can now be found in support of distance collaboration [6], education [1], entertainment[2], and so on. One class of potential users, software developers themselves, have ironically paid little attention to the technology. While it is now common to find two monitors on a software developer's desk, most development environments do not take advantage of this fact. We believe that, through the use of multiple displays of various sizes, it is possible to change this situation. Specifically, we have begun initial conceptual explorations into how the availability of multiple monitors and wall-size displays may alter the software development, and in particular the software design process and experience.

Two main problems exist in our current approaches to software design that could be improved via the use of new display technology. First of all, it is often difficult for individual software developers to comprehend the vast complexity of the systems that they are working on, especially when the system is designed from multiple points of view. Second, collaborating with other developers on such systems is often problematic, regardless of whether the collaborators are locally or remotely located. Display technologies have the potential to provide support for both these problems.

Below, we discuss both of these issues and present some of the directions into which we believe the research should go. It is important to stress that we understand

that one aspect of this opportunity lies simply in the additional screen real estate that multiple displays can provide. The deeper research, however, comes with the question of how to effectively use that space. To do so, current software development and design environments should be created very differently from their current incarnations – resulting in new kinds of uses: improved collaboration, more efficient processes, and better context sensitivity when it comes to the tasks that software developers perform on a day-to-day basis.

2 Easing context switching for individual developers

Much work has been done to provide software engineers with tools to visualize software systems and processes. One only has to look at the plethora of formalizations and modeling notations available, culminating in the current industry standard of UML as supported by tools such as Rational Rose [10]. While it is common for these models to have multiple views (e.g., in UML one can design class diagrams, interaction diagrams, object diagrams, deployment diagrams, and so on; in software architectures it is common to have a configuration view, an implementation view, a deployment view, and so on [7]), little work has been done to ease the developer's task of using more than one of these views at the same time. The de facto approach has been to force the developer to switch between multiple windows (or worse yet, to switch the content of a single window), each containing its own view. This context switch is burdensome on the developer in subtle and significant ways. It requires the developer to rearrange their hands to issue the command to switch views, and while they are looking at the alternate view, they no longer have access to a view of the previous task (unless they operate with lots of small windows, in which case typically no information is present or usable whatsoever).

A solution that we envision and intend to prototype is to provide each developer with three displays, one on either side of the primary monitor. It is possible to create software that will support these monitors and allow for different diagrams to be assigned to each, enabling the developer to glance at appropriate views of the system while minimizing the disruption of their primary task. Clearly, the tools must now be adapted to this new situation. For instance, it becomes immediately obvious that it would be very helpful to highlight in real time how changes in one view influence other views. Additionally, the tools must provide a user control over which views to display at any time – the creative design process must be flexibly supported.

We also observe that the benefits extend not just to multiple views, but even to entirely different artifacts. For instance, while designing, it may be beneficial for the software development environment to automatically bring up relevant requirements on the auxiliary monitor. The same may happen during coding, but that activity could be supported to an even greater degree by automatically bringing up the architecture of the system and highlighting any deviations that newly written code is introducing – thereby guarding against the problem of architectural drift [8].

3 Encouraging collaboration on software artifacts

Software design is not a singular activity. While individual developers may work on refining parts of a design by themselves, the initial blueprint of a system is almost always designed by a small team of collaborating designers. Often, they will collaborate during the refinement phase as well when individual designers run into problems or are in need of clarification. Currently, this kind of collaboration is usually handled via meetings in which the appropriate members plan aspects of the system on a whiteboard or other shared drawing space. The use of a large-scale display could emulate a whiteboard's functionality, while providing additional features. Interactive whiteboards of this nature have been explored by other researchers such as Winograd [4]. However, these approaches have been largely generalized and have not addressed the specific needs of a software engineering team.

A problem with traditional whiteboards is that transferring the results achieved in the shared space to personal electronic spaces can be difficult. Some research has been performed into addressing this issue in a variety of ways [5, 9]; this problem must be ameliorated if we are to expect frequent informal collaboration to take place. Furthermore, we note that software design is inherently multi-faceted. It is not uncommon to see a design session in which designers move from a high-level architecture to low-level data structures, back to a user interface, and into some algorithmic aspects of a certain component in the architecture. Currently, no tools exist that even approximate appropriately capturing this kind of information in a way suitable for the individual developer to easily continue their work, nor do the tools help in any which way in performing these kinds of explorative sessions – one view is still the common mode (as discussed in the previous section).

Finally, we observe another aspect of collaboration. A traditional problem in tools supporting coordination of distributed developers is where to place the "awareness information" [3]. A significant portion of the development of these systems is therefore tailored to minimizing obtrusiveness while still maintaining some level of informative directions. What if, instead, an entire secondary monitor can be leveraged to display relevant information on all ongoing parallel activities, with summaries of others' work, or even to have a direct, real-time view of others' work. In such cases, the collaboration paradigm would change, the parallel process of software development becomes much more visible, and software engineers are provided with a multi-faceted understanding of complex projects.

4 Conclusions

It is our belief that the discipline of software development is ripe for improvement via emerging display technologies. The use of multiple displays has the potential to ease the understanding of complex documents by individuals, and shared displays are able to assist collaboration. By connecting these different types of displays and allowing for the simple transition of work between them, it may be possible to create a work environment that makes developing software easier than ever before. It is our belief

that a strong first step in investigating this potential would be the development of environments in which displays of this kind are made available. Through experimentation with this technology and the creation of software to take advantage of it, there lies the potential to greatly improve the manner in which software of all kinds is developed.

We acknowledge that our work with this technology is only beginning. Though we have begun to experiment with the use of multiple monitors by a small group of software developers, many issues remain to be addressed. For example, many approaches could be used to move views and diagrams between monitors, to draw different types of diagrams on an electronic whiteboard and to display information about parallel activities. Other issues exist outside of the technology itself, for example privacy concerns may arise from monitors displaying information about others' work. It is our hope that through continued investigation and discussion of display technology's use in this context that our understanding of such issues will continue to grow.

5 Acknowledgements

This effort has been funded by the National Science Foundation under grant numbers CCR-0093489 and IIS-0205724.

References

1. Abowd, G., Brotherton, J., Bhalodia, J.: Classroom 2000: a system for capturing and accessing multimedia classroom experiences. Conference on Human Factors in Computing Systems. ACM Press, New York, NY, USA (1998) 20-21
2. Björk, S., Holopainen, J., Ljungstrand, P., Åkesson, K.: Designing Ubiquitous Computing Games – A Report from a Workshop Exploring Ubiquitous Computing Entertainment. Personal and Ubiquitous Computing, Vol 6, Issue 5-6. Springer-Verlag, London, UK (2002) 443-458
3. De Guzman, E., Yau, M., Gagliano, A., Park, A., Dey, A.: Exploring the design and use of peripheral displays of awareness information. Conference on Human Factors in Computing Systems. ACM Press, New York, NY, USA. (2004) 1247-1250
4. Guimbretière, F., Stone, M., Winograd, T.: Fluid interaction with high-resolution wall-size displays. Proceedings of the 14th annual ACM symposium on User interface software and technology. ACM Press New York, NY, USA. (2001) 21-30.
5. Holmquist, L.E., Sannebald, J., Gaye, L.: Total recall: in-place viewing of captured whiteboard annotations. Conference on Human Factors in Computing Systems. ACM Press, New York, NY, USA. (2003) 980-981
6. Ishii, H., Kobayashi, M., Grudin, J.: Integration of interpersonal space and shared workspace: ClearBoard design and experiments. ACM Transactions on Information Systems, Vol 11, Issue 9. ACM Press, New York, NY, USA (1993) 349–375
7. Kruchten, P.: The 4+1 View Model of Architecture. IEEE Software, Vol 6, Issue 12. IEEE New York (1995) 42-50

8. Perry, D.E., Wolf, A.L.: Foundations for the Study of Software Architectures. SIGSOFT Software Engineering Notes Vol 17, Issue 4. ACM Press New York, NY, USA. (1992) 40-52.
9. Rekimoto, J.: Pick-and-drop: a direct manipulation technique for multiple computer environments. Proceedings of the 10th annual ACM symposium on User interface software and technology. ACM Press, New York, NY, USA. (1997) 31-39
10. Rose Resources. IBM. July 7, 2004 <http://www-136.ibm.com/developerworks/rational/products/rose/>