On the Timing of Presenting Situational Talkback in Support of Reflective Practitioners

Yunwen Ye[1,2]

[1]Center for LifeLong Learning and Design
Department of Computer Science
University of Colorado, Boulder, CO80309-0430
+1 303 492 3547

[2]SRA Key Technology Laboratory
3-12 Yotsuya, Shinjuku
Tokyo 160-0004, Japan
+1 81 3 3347 9361

yunwen@cs.colorado.edu

INTRODUCTION

Design is an activity of making plans to inform the process of making, in order to create artifacts we want to surround us [5]or change the current situation to our satisfaction [9]. Since the primary goal is to inform making, certain representations are bound to be produced during the process of designing. Design could be logically regarded as the repetition of making the representation (action), and interpreting the partially finished representation (reflection) in terms of its fitness with the ideal image in the mind of the designer. The interpretation leads to further actions that either change the current representations for better fit or add more representations for better approximation of the ultimate design goal.

If we view the design process as a goal-oriented cognitive process in which the goal itself is sometimes not well defined, each action during the design process could be viewed as a decision-making one which is chosen from possible candidates after deliberation. The decision-making process, is dynamically determined by the knowledge that the designer has in his or her mind and the information presented in the workspace in which the designer is placed. In other words, the design process is a continuous dialogue between human minds and the interim representations [8] Designers act to make representations, reflect upon the information presented by the representations, and act further after reflection.

Because the information contained in the workspace and the representation is an important resource for reflection, the important research questions for providing support for reflective practitioners are: What information should be presented? How should such information be presented? And when should such information be presented? This position paper describes some of my research efforts in understanding the timing of presenting information in support of reflective designer.

THE TIMING OF PRESENTING INFORMATION

Design is a knowledge intensive activity and result from the design activity is a knowledge artifact that embodies the knowledge of the designer. The knowledge that comes from the designer could be acquired by the designer through three different phases: before, in and after the action.

The knowledge that has been acquired by the designer *before* he or she starts design is the result of his or her professional education and experience. In other words, the knowledge is the result of previous learning or interpretation of information presented to the designer long before the design starts. The context in which learning takes place is different from the context in which the learned knowledge is applied.

The knowledge acquired *after* the action is finished is a process of learning from previous experience, from the feedback information of the action. To support reflective designers to reflect on their finished action, feedback information is presented when the action for which the information is provided has been finished. Feedback can create a situational backtalk of the action by pointing out a potential breakdown the designer has not known or noticed, or can augment the situational backtalk to help designers reflect better on the action just completed. Feedback can serve two roles. First, it creates a learning opportunity for designers to improve work performance. For example, the ACTIVIST system [4] teaches users the corresponding key shortcut to replace a series of complex keystrokes used in their previous action in a text editor. Second, if the previous problematic action can be undone or modified, it helps users reach a better solution, such as the on-the-fly spell-checking mechanism in many word-processing systems.

The knowledge acquired *in* the action is a process of expanding the knowing-in-action. For each design situation, there is a period of time called *action-present* in which the designer remains in the "same situation." This is a period of time that the designer has made up his or her plan of action but has not executed the necessary operations to change the situation. Information presented in the action-present period could be immediately acquired by the designer and applied to change the designer's original design plan. Information presented in this period of time is feedforward [9] information because it can make designers change the course of action or assist designers in accomplishing the action [10].

These three forms of knowledge acquisition could be unified in terms of the temporal relationship between the timing of the information being presented and the timing of its application. In the first case, information is presented

without any advanced knowledge of its potential context. In the second case, information is presented when the context is still remembered or can be easily reconstructed by the designers. In the third case, the information is presented right into its application context.

OPPORTUNITIES FOR SUPPORTING REFLECTION-IN-ACTION IN SOFTWARE DEVELOPMENT

Software development involves many design activities. Software developers engage in many reflections-in-action. However, few systems have been developed to provide explicit support for reflective software developers. I want to explore the opportunities of supporting reflection-in-action by applying the framework described in the previous section.

The goal of software development is to create computational representations for problems to be solved. Since the representations are computational already, some might think that software should be the best domain for augmenting situational talkback to support reflection-in-action. Apparently, this is not the case. Except for a few exceptions [3, 7], very little software engineering research literature has brought up this issue. However, a careful analysis of certain efforts in software engineering leads to the conclusion that the research community has, all the time, been trying to do that implicitly.

Brooks claimed that the invention of interaction programming environment is the most significant development in advancing our capability of software construction [2]. If we analyze this observation from the perspective of reflection-in-action, we can see it is simply because the representational feedback information is presented to the programmer much closer to its original context, in terms of timing. It is well accepted that interpretative programming language, though less efficient performance-wise, is easier to use and easier to learn because programmers can try their programs immediately without going through the save-file and compilation phase.

Code review is a process in which project team come together to review the code written by one member to find bugs in the code or provide feedback to improve the code. The representational talkback comes from the social environment. Due to the difficulty of organizing reviewing team and coordinating review process, program review is often an expensive process and cannot be done right after the code is written or at the needs of the software developer. Computationally networked socio-technical environment provides a new opportunity of coordinating such social process of providing representation talkback. For example, one of the success factors in Open Source Software development is that "given enough eyeballs, all bugs are shallow [6]." Support for reflective software developers means not only providing computational mechanisms that augmenting or presenting timely the situational talkback, but also facilitating the timely presentation of socially situational talkback.

Rapid prototyping, especially research on executable specification, is yet another effort. There is a long separation between the phase in requirement acquisition (problem framing) and the development of executable code (problem solution) that can provide situational talkback. To make the initial specification executable is an effort of shorten the time separation so that modification could be made earlier and easier.

Extreme programming and agile development methodology [1] has pushed this a little further by breaking down the long separation between the framing of problems and the solution of problems. Functionality is added incrementally as a result of incorporating the feedback from the users. Pair programming is an effort of providing immediate socially representational talkback.

There are many tools developed for analyzing ripple effects or test coverage or slicing. All those tools are supposed to provide feedback information on certain software development actions. For example, ripple effect analysis is meant to identify the range of code that is affected by code or design modification. However, the cycle of making modification and getting the feedback is too long and too cumbersome because the analysis is only available when the modification is made final. Software developers can get better tool support if the situational talkback of the modification is presented immediately after the modification is made. If the feedback is presented even before the code modification is finally committed, software developers can execute reflection-in-action better.

The above list is definitely limited, but serves as a starting point of thinking how to support reflective software developers with better development methodology and tools that presents representational backtalk in a timely fashion.

CONCLUSION

This position paper describes my research efforts in reframing some research problems in software development from the perspective of the theory of reflection-in-action, and in exploring the opportunities of supporting reflective software developers. Most of the thinking remains theoretical and will be evaluated further in the near future with system implementation and empirical studies.

REFERENCES

1. Beck, K. Extreme Programming Explained. Addison-Wesley: Reading, MA, 2000.
2. Brooks, F.P.J. The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary edition. Addison-Wesley: Reading, MA, 1995.
3. Fischer, G. Domain-Oriented Design Environments. Automated Software Engineering, 1994. 1(2):177-203.
4. Fischer, G., Lemke, A.C., and Schwab, T. Knowledge-Based Help Systems, in Human Factors in Computing Systems (CHI'85): San Francisco, CA, 1985, 161-167.

5. Habraken, N.J. The Appearance of the Form: Four Essays on the Position Designing Takes Between People and Things. Awater Press: Cambridge, MA, 1985.
6. Raymond, E.S. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly: Sebastopol, CA, 2001.
7. Robbins, J.E., and Redmiles, D.F. Software Architecture Critics in the Argo Design Environment. Knowledge-Based Systems, 1998. 11:47-60.
8. Schön, D.A. The Reflective Practitioner: How Professionals Think in Action. Basic Books: New York, 1983.
9. Simon, H.A. The Sciences of the Artificial, Third edition. The MIT Press: Cambridge, MA, 1996.
10. Ye, Y., and Fischer, G. Supporting Reuse by Delivering Task-Relevant and Personalized Information, in Proceedings of 2002 International Conference on Software Engineering (ICSE'02) (Orlando, FL., 2002), 513-523.