Design Strategy and Software Design Effectiveness[1]

Antony Tang and Hans van Vliet[2]

Software design strategy is about organizing design activities during the course of design. The organization of design activities, be it well-planned or ad-hoc, reflects a designer's approach to creating a design. It is common that a software designer thinks about a list of design issues and tasks that need to be addressed and just do them in an ad-hoc way. However, the planning and execution of design activities matters to the effectiveness of a design. In this article, we describe our observations of how designers work and the different design strategies that can be used to improve design effectiveness.

1. Design Activities

There are two important types of design activities. First, *Design Exploration* is about how designers plan to design. When people deal with highly complex problems, planning helps to identify the scope up-front, and identify areas where risks exist [1]. A designer may choose to use different approaches such as a breadth-first strategy to explore the broad scope of top-level design issues, or a depth-first strategy to explore a key design issues. The choice of a strategy reflects where the risk of a design is, whether enough is known about the scope or the depth of a problem.

Second, *Design Problem Solving* is about developing a design through the exploration of the problem space and the solution space. Some designers use a solution-driven strategy, and some use a problem-solution coevolution strategy. The choice of a strategy should reflect how familiar a designer is with the type and interrelationships of the problems.

During software design, *design exploration* and *problem solving* activities can be carried out in different combinations and time order. Figure 1 shows the iterative application of these activities. For instance, a designer may decide whether to explore the broad design space or to explore a particular design area in-depth. When a designer moves to the actual design, he may choose to use a problem-driven or a solution-driven approach. Designers can switch between these activities during design. However, the way strategic design activities are intermixed can influence the outcome and the effectiveness of reaching a final design. When a chosen approach does not fit the problem situation, such negatively impacts the design outcome. For instance, in designing a nationwide e-government system, it does not make sense to start with a depth-first approach because many high-level issues need to be considered upfront. Designers must understand the scope of their problems, plan which design activities are important and must be tackled earlier.

[1] This article is an excerpt from an article published in IEEE Software (51-55, Jan-Feb 2012) by A. Tang and H. van Vliet, Design Strategy and Software Design Effectiveness.

[2] Antony Tang is Associate Professor, Swinburne University of Technology, email:atang@swin.edu.au. Hans van Vliet is Professor, VU University Amsterdam, email: hans@cs.vu.nl

2. Design Teams In Action

In this study, we describe the design strategies employed by two design teams of two designers each. Team M (Amberpoint team) and Team A (Adobe team) were asked to design a traffic simulator.[3] Both Team M and Team A were given the same design problem but they used different design exploration strategies.

There are two design exploration strategies, breadth-first and depth-first, and the strategy can change during a design session. If designers adopt a breadth-first strategy, they investigate the broad goals and the design scope of a system, and make decisions on what to design first based on the relative priorities of the different issues. Knowing the broad goals up-front in certain circumstances allow designers to assess design risks. On the other hand, some designers choose to design a part of a system in depth from the beginning, they investigate a specific design area in details. The timing and choice of a design exploration strategy is a decision a designer always makes, even if unconsciously. A design exploration strategy can change in the middle of a design. For instance, one may switch to a depth-first strategy to study an important issue that can be a show-stopper.

Team M used a breadth-first strategy at the early stage of design. Later, they took a depth-first exploration strategy to investigate the details of their design. Team A, on the other hand, spent only 1.5 minutes upfront on investigating the scope of the problem. They used a depth-first exploration approach.

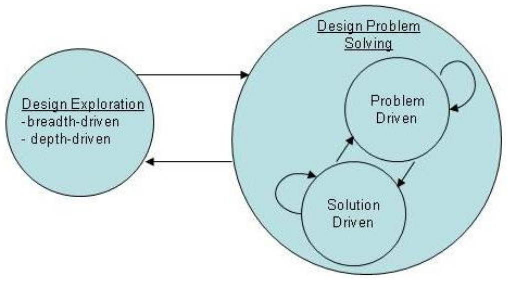

Figure 1 – Exercising Strategic Design Activities

Designing software in an unknown domain requires an understanding of the design problems. The goals of a software design are typically specified in high-level. During design, new and implicit goals are discovered. They could be undesirable problems that need to be avoided, or new sub-goals that need to be achieved. All of these problems need to be identified and solutions to be found. Dorst and Cross [2] show that the discovery and evolution of solutions are based on problem identification, and they suggest that a solution can evolve with a problem. This is called problem-solution coevolution (PS-coevolution).

When observing Teams A and M, we noticed that they also used different problem solving strategies. Team A used a solution-driven strategy, they already made solution statements such as "I am thinking in terms of MVC…" in the 5th minute. Team M largely followed a PS-coevolution strategy, they made problem statements such as "what are the effects of …".

[3] This study is based on an NSF-sponsored workshop that took place at the University of California, Irvine, in February 2010.

Throughout the design session, they spent as much time exploring the problems as proposing solutions, alternating between raising design issues and proposing solutions. The questioning approach they used appeared to have given Team M new insights, and the solutions that followed addressed traffic initiation into the simulation model.

In the end, Team M identified many more problem statements than Team A. As a percentage of the total number of statements made, Team A made 13.4% of problem statements compared to Team M's 31.9%. Basically, Team M formulated their design problems thoroughly during solution creation. This has facilitated a focused design discourse that serves to reduce design context switching [3].

3. Design Strategies

The design teams M and A used Design Exploration and Design Problem Solving implicitly without consciously discussing how these strategies should be applied. We suggest that using a combination of these two types of strategy in the right context can help design effectiveness. In the design exploration dimension, designers can choose a breadth-first versus a depth-first strategy to scope which part of the design space needs to be explored first; in the problem solving dimension, designers can choose a problem-driven versus a solution-driven strategy. This gives rise to four archetypical strategies (Table 1).

		Problem Solving Strategies	
Design Exploration Strategies		**Problem-driven**	**Solution-driven**
	Breadth-first	Scoping/questioning [*Context: high complexity/low experience*]	Scoping/solving [*Context: high complexity/high experience*]
	Depth-first	No scoping/questioning [*Context: low complexity/low experience*]	No scoping/solving [*Context: low complexity/high experience*]

Table 1 – Software Design Strategies

The conscious selection of a strategy should be guided by the complexity level of a design and the experience of the designers. Complexity is about the size and the difficulty of a system. Experience is about a designer's familiarity with the specific problems and the solutions of a system.

These strategies can be characterized as follows:

- **Scoping/questioning**: This strategy is applicable to green-field and complex systems in which designers have no prior experience. The goals and scope of a design need to be explored first to understand broad design issues. This helps designers to appreciate high-level relationships between design issues, and to plan and prioritize design activities. When the search for a solution begins, new problems are identified and there would be frequent backtracking to design problems as the solution evolves. This fits a high complexity and low design experience (of the problem domain) situation. This is the initial strategy chosen by Team M. At a later stage, they switched to the no scoping/questioning strategy for certain parts of the problem.

- **Scoping/solving**: This strategy is applicable to using packaged or known solutions where the application domains and the problems are well known. A solution is readily available. Designers start with exploring and setting the scope to ensure that no major surprises exist in deploying a solution. If the target system falls within the known capabilities of the solution, then there are few design problems that need to be explored. This fits a high complexity and high experience situation.

- **No scoping/questioning**: This strategy is applicable to design in which designers are familiar with the overall scope but certain aspects of a design are unknown to designers. Technical and algorithmic problems such as security implementation are examples. While searching for a solution, regular backtracking to explore the problem space is required.

- **No scoping/solving**: This should be used when the design problem and the solution is well-known. This fits a low complexity and high experience situation. But this strategy was followed by Team A throughout the whole session though they were unfamiliar with the problem and the solution.

Selecting the right combination of strategies given the nature of a particular problem type at the right moment is important in positioning how best to design, but selecting a wrong strategy can give rise to design problems. We show that under a high complexity and low experience situation, the misuse of a strategy can make design less effective.

In our study, we found that Team A spent 1% of their total time in initial design exploration. They also used depth-first solution-driven strategies. As they did not have a broad scope of design issues from the beginning, they could not prioritize what problems are more important. When they developed their solutions, these issues arose and that made them switch design topic. This switching back and forth caused them to revisit the same design topic again. They context-switched their design topics in 1 out of 4 discussions. As the design problems grew more complicated overtime, it made them revisit the same problem over and over. Team A had a ratio of problem to solution statements of 1 in 3, indicating their strong preference for a solution-driven strategy. Such a strategy led Team A to prematurely anchor on a solution. When different problems started to appear and the anchor did not shift, the problems gradually became more difficult to tackle because the initial solution was not as suitable as it first appeared.

On the other hand, Team M used a breadth-first strategy that helped them understand the design scope. Team M did not rehash the same issues as often as Team A. Team M used a PS-coevolution strategy. Their ratio of problem to solution statements is almost 1 to 1, showing that they explored the problem space more thoroughly. Due to the combination of these strategies, Team M context-switched design topic less frequently, 1 in 7.7 discussions, making their discussions more effective. When Team M investigated a focused set of problems, they took a depth-first strategy. This is effective when the issues in that area are well understood, and the designers do not have to branch out to tackle side issues.

Traffic simulation and the design issues are new and complex to the designers. Under these circumstances, the scoping/questioning strategy is more effective because in-depth design that follows a systematic problem exploration helps to minimize context switching. On the other hand, a no-scoping/solving strategy potentially postpones problem identification till later. A solution anchoring effect appeared to encourage Team A to stay with their solution. When designers anchor on a chosen solution prematurely, they usually are reluctant to abandon the original solution choice and backtrack to a better solution [4].

4. Lessons Learned

The four quadrants in Table 1 show four design strategies and the contexts under which they can be used effectively. The choice of a design strategy depends on the complexity of a design problem and the designers' experience with a design problem. The complexity of a design problem is relative to both the difficulties of the problem and the designers' experience in the relevant application and technical domains. A designer may have worked for many years but can be relatively inexperienced with a new domain and technology. A design problem that appears complex to one designer may be familiar to another designer. Mismatches between the contexts and the strategies can harm design effectiveness and quality. If we consider Team A and Team M in the design of a traffic simulator, the design is complex and none of the designers have prior experience on this subject. A choice of scoping/questioning strategy is more helpful, which Team M took. Team A followed a no-scoping/solving strategy that was less effective.

If designers are experienced in a certain application domain, and a packaged solution is available, then using a questioning strategy to revisit solved problems would be a waste of time. Instead, designers should use a scoping/solving strategy to design. As new information and design issues are discovered, a seemingly simple design may become more complex than first anticipated. Newly discovered design issues may impact the design strategy. Designers need to choose a suitable strategy conscientiously and adopt new strategies as the context changes.

3. References

[1] D. Dörner, *The Logic Of Failure: Recognizing And Avoiding Error In Complex Situations*: Basic Books, 1996.

[2] K. Dorst and N. Cross, "Creativity in the design space: co-evolution of problem-solution," *Design Studies,* vol. 22, pp. 425-437, 2001.

[3] A. Tang, A. Aleti, J. Burge, and H. van Vliet, "What makes software design effective?," *Design Studies,* vol. 31, pp. 614-640, 2010.

[4] A. Tang, M. H. Tran, J. Han, and H. van Vliet, "Design Reasoning Improves Software Design Quality," in *Proceedings of the Quality of Software-Architectures (QoSA 2008)*, 2008, pp. 28-42.