# Design Principles for Tools to Support Creative Thinking

Mitchel Resnick, MIT Media Lab
Brad Myers, Carnegie Mellon Univ.
Kumiyo Nakakoji, Univ. of Tokyo, Japan
Ben Shneiderman, Univ. of Maryland
Randy Pausch, Carnegie Mellon Univ.
Ted Selker, MIT Media Lab
Mike Eisenberg, Univ. Of Colorado

Revised October 30, 2005

## Introduction

We have developed a set of "design principles" to guide the development of new creativity support tools – that is, tools that enable people to express themselves creatively and to develop as creative thinkers. Our goal is to develop improved software and user interfaces that empower users to be not only more productive, but more innovative. Potential users of these interfaces include software and other engineers, diverse scientists, product and graphic designers, architects, educators, students, and many others. Enhanced interfaces could enable more effective searching of intellectual resources, improved collaboration among teams, and more rapid discovery processes. These advanced interfaces should also provide potent support in hypothesis formation, speedier evaluation of alternatives, improved understanding through visualization, and better dissemination of results. For creative endeavors that require composition of novel artifacts (e.g., computer programs, scientific papers, engineering diagrams, symphonies, artwork), enhanced interfaces could facilitate exploration of alternatives, prevent unproductive choices, and enable easy backtracking.

Some of these design principles have appeared previously [Myers 2000][Shneiderman 2000][Resnick 2005][Yamamoto 2005][Hewett 2005][Selker 2005]. These principles have emerged through collaborations with a large number of colleagues, in the development of many different creativity support tools, both for children and adults. Some of the principles are also relevant to tools for creating software in general, often called "User Interface Software Tools," but targeting tools specifically for creativity highlights new perspectives and requirements.

In our analysis, we focus especially on "composition tools" -- that is, computational systems and environments that people can use to generate, modify, interact and play with, and/or share both logical and/or physical representations. A creative composition process is not a routine production process that can be prescribed, and what tools and representations people use strongly affect their courses of actions and thought processes [de la Rocha, 1985][Zhang, 1997][Shirouzu et al. 2002].

While it is difficult to study "creativity" itself, we can study the process by which creative people and teams work, and embody their best practices in tools that can aid others in emulating those processes. Examples include the IDEO design team "brainstormer" process has been nicely documented and used by many other organizations [Kelly 2001] and the widely-used TRIZ method for systematic innovation [Mann 2002]. Strategies for studying creativity support tools are discussed in this report's section titled "Creativity Support Tool Evaluation Methods and Metrics."

## 1. Support Exploration

An important requirement for creativity is to be able to try out many different alternatives. Almost by definition, creative work means that the final design is not necessarily known at the outset, so users must be encouraged to explore the space [Fischer 1994]. This has a number of implications on the tools. Exploratory programming has been promoted for a long time [Sheil 1983], but most tools still focus on projects where the outcome is well-known in advance. In the terms of Green and Petre [Green 1996], we want systems with "low viscosity" – that make it easy to change all aspects of the design.

First, it must be very easy to try things out, and then backtrack when unsuccessful. This means that the tools must be trustworthy so that users are comfortable trying things. For example, a very good Undo capability is required in the tools. Implementing Undo can be quite difficult however [Myers 1996], so many research systems leave it off. The rich histories that are required to support undo can also be useful for creating programming-by-demonstration interfaces [Myers 1992], where users teach the system how to automate repetitive tasks by giving examples. Previewing mechanisms [Terry 2002] and set-based operations [Terry 2004] have also been proposed and tested to support such processes.

A second requirement is that the tools be "self-revealing" so that it is clear to users what can be done. If the flexibility is not apparent, it will not be used. This is especially important as users are learning the tools. The tools must also be facile and unencumbering, so that expert users can try out the different alternatives very quickly. Finally, tools must be pleasurable and fun to use. When people are stressed or concentrating too much effort on how to use the tools themselves, then they will have less cognitive resources left over for use on finding creative solutions to their tasks.

Spreadsheets are famous for giving people an ability to compare results in what-if scenarios [Brown 1987], by enabling the user to easily separate what will stay fixed (the formulas) from what should vary (the values to be explored). In the user interface realm, tools such as Flash and Visual Basic are popular because they allow prototypes to be created, evaluated and modified quickly.

Another way to support exploration is to make it very fast to "sketch" out different alternatives at the early stages of design. Professional user interface designers will often try out dozens of ideas by drawing on paper or a whiteboard, before starting to write code

for a real implementation. The goal is to allow a partial effort to get a partial result quickly. Tools to facilitate this kind of sketchy exploration include Silk [Landay 1995] and Denim [Lin 2002], which specifically focus on sketching screens and storyboards of interactive behaviors.

Supporting exploration requires functionality made available through careful interaction design. We view a computational tool as something that provides materials with which users interact to create a situation that "talks back to the users" [Schoen 1983][Nakakoji 2000a]. Tools for fostering, not obstructing, creativity need to be designed around the understanding of what representations users need to interact with [Yamamoto 2005].

The interaction design of a tool influences a user's cognitive process. By interaction design, we mean to determine the representations and operations of an application system [Yamamoto 2005]. Systems for supporting creative processes need to enable users not only to compose artifacts, but also to think of what to compose as artifacts [Nakakoji 2005]. Historically, existing tools and application systems have been mostly used to digitally compose artifacts. Examples are word-processing software, image-processing software, or spreadsheet applications. Elaborated 3D CAD systems are found effective in helping architects to compose solutions but obstructive to their creative exploration [Lawson 1994].

## 2. Low Threshold, High Ceiling, and Wide Walls

Effective tool designs should make it easy for novices to get started (low threshold) but also possible for experts to work on increasingly sophisticated projects (high ceiling) [Myers 2000]. The low threshold means that the interface should not be intimidating, and should give users immediate confidence that they can succeed. The high ceiling means that the tools are powerful and can create sophisticated, complete solutions. Too often tools that enable creative thinking may be quite hard to learn (they don't have a low threshold). Instead, they focus on providing numerous powerful features so that experts can assemble results quickly.

Now, we add a third goal: *wide walls*. That is, creativity support tools should support and suggest a wide range of explorations. By not including predefined widgets, the Flash tool encourages designers to explore many different ways to control the interaction, instead of just using buttons and scroll bars. Carnegie Mellon's Alice tool has enabled the creation of a wide variety of three-dimensional stories, games, and interactive Virtual Reality experiences [Conway 2000], and the Disney/Carnegie Mellon Panda3d System (panda3d.org) has allowed theme park, online, and classroom content creation. When kids use MIT's Programmable LEGO Bricks, for instance, they can create anything from a robotic creature to a "smart" house to an interactive sculpture to a musical instrument [Resnick 1993][Resnick 1996]. We want users to work on projects that grow out of their own interests and passions – which means that the creativity support tools need to support a wide range of different types of projects.

When evaluating the use of creativity support tools, we consider diversity of outcomes as an indicator of success. If the creations are all similar to one another, we feel that something has gone wrong. And if, after finishing one project, users feel that they are "done" with the tool, again we feel as if we have failed. Creativity support tools should define a space to explore, not a collection of specific activities. And our hope is that users will continually surprise themselves (and surprise us too) as they explore the space of possibilities. As an example, it was a surprise that kids would use MIT's Programmable LEGO Bricks to measure their speed on rollerblades, or to create a machine for polishing and buffing their fingernails [Resnick 2000].

The problem with systems that aim for a low threshold is that they usually are quite limited in what they can do, so users are either constrained, or else need to find "work-arounds" to achieve what they want. Tools with high ceilings tend to require significant training and effort to learn how to use. And wide walls means that there are very general primitives that users must learn how to combine.

One strategy to try to achieve all three is to explicitly include elements and features that can be used in many different ways. The design challenge is to be specific enough so that users can quickly understand how to use the features (low threshold), but general enough so that users can continue to find new ways to use them (wide walls). The tool should help users learn how to use the features, for example with mouse-overs, tool-tips, and a variety of examples, so users can make the transition necessary to understand the variety of possible uses.

## 3. Support Many Paths and Many Styles

When MIT researchers were testing an early version of the computer-controlled LEGO technology, they tested prototypes in a fourth-grade classroom where students wanted to build an amusement park. One group of students decided to create a merry-go-round. They carefully drew up plans, built the mechanisms, and then wrote a program to make the ride spin round-and-round whenever someone pressed a touch sensor. Within a couple hours, their merry-go-round was working. Another group of students decided to build a Ferris wheel. But before the ride was working, they put it aside and started building a refreshment stand next to the Ferris wheel. The developers were concerned: the refreshment stand did not have any motor or sensors or programming. They worried that the students would miss out on some of the powerful ideas underlying the LEGO/Logo activity. But they didn't interfere. After finishing the refreshment stand, the group built a wall around the amusement park, created a parking lot, and added lots of little LEGO people walking into the park. Then, finally, they went back and finished their Ferris wheel.

These two groups represent two very different styles of playing, designing, and thinking. Turkle and Papert [Turkle 1990] have described these styles as "hard" (the first group) and "soft" (the second). The hard and soft approaches, they explain, "are each characterized by a cluster of attributes. Some involve organization of work (the hards

prefer abstract thinking and systematic planning; the softs prefer a negotiational approach and concrete forms of reasoning); other attributes concern the kind of relationship that the subject forms with computational objects. Hard mastery is characterized by a distanced stance, soft mastery by a closeness to objects."

Similarly, faculty at CMU through a decade of working with creative people at Walt Disney Imagineering, Electronic Arts, and in CMU's Entertainment Technology Center (ETC) (see www.etc.cmu.edu) have identified similarities and differences between "left brain" (logical, analytical) and "right brain" (holistic, intuitive) thinkers. You might think that people who do Art and Graphic Design would be the "soft" or "right brain" group (as people who can draw), versus people who do Science and Engineering (as people who can do math). But in fact, people who focus on art and science often have more in common with each other, as people who focus on finding the "truth", compared to Graphic Design and Engineering, who are people who try to solve problems, often within constraints such as budget, time, and client demands.

In many math and science classrooms, the hard approach is privileged, viewed as superior to the soft approach. Turkle and Papert argue for an "epistemological pluralism" that recognizes the soft approach as different, not inferior. We should take a similar stance in the design of new creativity support tools, putting a high priority on supporting learners of all different styles and approaches. We should pay special attention to make sure that technologies and activities are accessible and appealing to the softs; since math and science activities have traditionally been biased in favor of the hards, we want to work affirmatively to close the gap.

## 4. Support Collaboration

An important implication of this diversity is the need to provide support for *collaboration* in the tools. In all of our projects, in schools, and in the "real world," most creative work is done in teams. At CMU, the ETC specifically focuses on creating teams that include people with various strengths. But diversity of talent will appear in all teams. It is important that the creativity support tools allow each person to contribute using their own talent. For example, the Building Virtual Worlds course at Carnegie Mellon (etc.cmu.edu/curriculum/bvw.html) requires creating 3D interactive virtual worlds, which combine art work, sound design, script writing, and programming. The tools allow team members to work on their own parts in parallel, but more work is needed on supporting the integration and iteration which results from these kinds of activities.

With the advent of the Internet, another form of "collaboration" has become prevalent: finding good material from others by using search tools like Google. Creativity support tools should foster a community of users to share their creations, and the tricks and techniques they have discovered for using the tools. Research shows that when confronted by a challenge, experienced and novice creators alike will go to Google to see if they can find the answer. And with professional tools, often they will find a host of examples, discussion and documentation posted by other users. Some research tools have

built-in techniques to help with posting creations (e.g., Agentsheets.com [Repenning 2004], Alice.org [Conway 2000]), and current research is looking at how other people's examples can help people learn new systems [Stylos 2005]. Commercial tools such as Spotfire (http://www.spotfire.com) provide numerous strategies for sending results by email and posting results to a website with an automatically generated chat window to promote discussion. Social and psychological factors such as trust and appropriation play an important role in support of collaborative creativity [Nakakoji 2000b][Shneiderman 2000].

The NSF is already funding a number of "collaboratories" to help bring scientists working in the same field or on related problems together. Examples include The Gene Ontology project (http://www.geneontology.org/), the Protein Data Bank (http://www.rcsb.org/pdb/), the Collaboratory for Research on Electronic Work (http://www.crew.umich.edu/), etc.

## 5. Support Open Interchange

The creative process will not usually be supported by a *single* tool, but rather will require that the user orchestrate a variety of tools each of which supports part of the task. Creativity support tools should seamlessly interoperate with other tools. This includes the ability to easily import and export data from conventional tools such as spreadsheets, word processors and data analysis tools, and also with other creativity support tools. This requires that the data formats in the files be open and well-defined. Fortunately, the increasing pervasiveness of XML and projects such as the Gene Ontology (http://www.geneontology.org/) (which provides a controlled vocabulary to describe gene and gene product attributes in any organism) show promise in this direction.

Another form of openness allows extensibility of the tools themselves. Professional tools increasingly provide a "plug-in" architecture, or an "open data model" [Myers 1998] to support extensibility. This has long been available for artistic tools like PhotoShop to allow capable creative people to define their own operations that work on the shared data types. Another example is the Eclipse tool (www.eclipse.org), which is not just a Java programming environment, but actually is a kernel into which many pieces can be plugged together to create a wide variety of environments to support different activities. The COM interfaces for Microsoft Office tools provide similar extensibility, although these are usually used to make business operations more efficient, rather than more creative. The professional suite of tools from companies such as AutoDesk (http://www.autodesk.com) and Adobe (http://www.adobe.com), are largely designed to facilitate taking the results from one tool into another.

Integration of operations across tools could allow smoother coordination across windows and better integration of tools. We have proposed operations such as getting an English definition, a French translation, or a medical dictionary report just by clicking on a word [Shneiderman 2000], and a "smart" clipboard that allows easy transformation of

structured information among many applications just using the familiar copy-and-paste operation [Stylos 2004].

## 6. Make It As Simple As Possible - and Maybe Even Simpler

In some ways, this design principle seems obvious. Who wants needless complication? But there is no doubt that technology-based products have become more and more complex. One reason is "creeping featurism": advances in technology make it possible to add new features, so each new generation of products has more and more features. This trend is reinforced by the belief among marketing professionals that it's quite hard to sell a product as "simpler," but much easier to sell it as "containing more features."

A related problem is getting a simpler design/tool past the "gatekeeper experts." Particularly in technical fields like Computer Science, a great deal of the self-image of the practitioners has historically been linked to their ability to master complex tools themselves. Later, simpler (and better designed) tools are often denigrated as "toys" – for example, while many would find this hard to believe, the introduction of the computer mouse was widely derided at the time by "serious" computer scientists.

We yearn for a return to the clean usability of the Macintosh of the 1980s. We see a role for complexity: we make use of ever-more complex technologies, and we want to help users accomplish complex tasks. But we want the user experience to be simple. We try to develop systems that offer the simplest ways to do the most complex things.

We have found that reducing the number of features can actually improve the user experience. What initially seems like a constraint or limitation can, in fact, foster new forms of creativity. In the mid-1990s, for example, MIT researchers had developed a Programmable LEGO Brick that was roughly the size of a child's juice box. It could control four motors and receive inputs from six sensors. For a sponsor event at the Media Lab, some interactive decorations for the tables were needed. All of the capabilities of the Programmable Brick were not necessary, so a smaller, scaled-down version was quickly developed, roughly the size of a matchbox car. It could control only two motors with inputs from only two sensors. This was expected to be a short-lived project, since they "knew" that most users would want more motors and more sensors. But once the scaled-down version (called a Cricket) was available, people kept finding more and more creative applications for it, in spite of (or perhaps because of?) its apparent limitations. Even though the original Programmable Brick was better for certain projects, the simplicity of the Cricket won out.

## 7. Choose Black Boxes Carefully

In designing creativity support tools, one of the most important decisions is the choice of the "primitive elements" that users will manipulate. This choice determines, to a large

extent, what ideas users can explore with the tool – and what ideas remain hidden from view.

When kids build robots with MIT's Programmable Bricks, for instance, they learn about mechanisms and gearing, and they learn about feedback and control. But they generally don't learn about the inner workings of motors. The motor remains a black box. If you wanted to help kids learn how motors work, you should design a construction kit with lower-level building blocks, so that kids could build their own motors.

Similarly, the choice of the basic "building blocks" in a programming language determines what kids are likely learn as they use the language. When kids put together Logo commands like **forward** and **right** into instructions like **repeat 4 [forward 50 right 90]** (to make a square) or **repeat 360 [forward 1 right 1]** (to make a circle), they gain a better understanding of programming concepts, and it has been argued that they also learn important mathematical and geometric concepts [Papert 1980]. But the primitive command **forward** is still a black box. Each time the turtle moves, the computer must calculate new x and y positions from the original x and y positions using trigonometric calculations. These calculations are hidden from the user. If the goal of the construction kit were to help kids learn these types of trigonometric calculations, then the turtle would be a bad black box. But by hiding these calculations inside a black box, the turtle frees the user to experiment and explore other mathematical and geometric ideas.

All language and toolkit designers face the same challenge. This is closely related to our point above about floor versus ceiling – the higher-level the primitives, the easier they are to use, but the less they can do. But even at the same level of capability, there are easier and more difficult ways to present the same functions. The designers of Alice rejected using conventional matrix representations for graphical transformations because it is not understood by the target audience [Conway 2000]. In the HANDS language, animation is built-in, so characters can be made to move simply by setting a speed and direction, rather requiring calculations of how far to move with each elapsed time interval [Pane 2002]. Similarly, for the language for programming the Cricket, the original design used the conventional Red, Green, Blue parameters to **setcolor.** This representation provides great power and flexibility, but kids found it difficult to map inputs from a single sensor into meaningful color values (for example, red for low values, blue/purple for high values). So a simpler **setcolor** command was added with just a single input that ranges from 0 to 100 (with 0 at the red end of the spectrum, 100 at the blue end), making it easy for kids to map sensor inputs to colors along the spectrum.. In these three examples, the goal was to enable the users to achieve a certain effect, not to teach the underlying principles (linear algebra, physics of motion, or the red-green-blue composition of light), so the simplifications were appropriate.

## 8. Invent Things That You Would Want To Use Yourself

At first blush, this design principle might seem incredibly egocentric. And, indeed, there is a danger of over-generalizing from your own personal tastes and interests. But we have

found that we do a much better job as designers of creativity support tools when we ourselves really enjoy using the tools that we are building.

We feel that this approach is, ultimately, more respectful to users of the technology. Why should we impose on users systems that we don't enjoy using ourselves? For example, we are generally skeptical of educational software that, in an effort to encourage students to reflect on their actions, requires that they annotate each action. We wouldn't want to do that with the software that we use, so why should we require it of students? Software engineering tools that require commenting, rather than encourage it, are similarly disliked by users.

There is an additional, perhaps less obvious, reason why we try to invent creativity support tools that we enjoy using ourselves. Creativity support tools can not succeed in a vacuum: they work best within communities where people share their expertise and experiences with one another (see this report section titled "Creativity and Distributed Intelligence" for further discussions of social creativity). When students use creativity support tools, for example, they require support from teachers, parents, and mentors. In developing creativity support tools for students, we need to build not only new technologies, but also communities of people who can help students learn with those new technologies. And we have found that it is easiest to build those communities if everyone involved (adults as well as students) enjoy using the technologies. In New York, for example, groups of MIT alumni have been volunteering their time to help kids at Computer Clubhouses (after-school centers for youth from low-income communities) learn to use the Programmable Bricks. The MIT alumni are motivated, in part, by a desire to help youth in low-income communities. But there is no doubt that they are also motivated by their own desire to build robots.

**9. Balance user suggestions, with observation and participatory processes**

Most successful designers seek to understand their users, in order to design products well-matched to the needs of their users. They invest considerable time observing and interviewing users, talking with focus groups, asking users for suggestions and feedback on features, and inviting users to participate in design processes. There are dangers to too little or too much involvement of the users, so thoughtful and balanced approaches are needed ([Druin, 2002] [Nielsen 1993] [Shneiderman 2004]).

Some researchers worry that users may ask for impractical or infeasible features. In other cases, users ask for only incremental changes, not aware of the possibilities of radical change. With early versions of Logo software in the 1980s, users often suggested new ways for the turtle to draw – but they never suggested the addition of paint tools.

Another concern is that users may ask for more flexibility than is needed or desirable. Often, designs with well-chosen parameters are more successful than designs with fully-adjustable parameters. We are all in favor of giving control to users – but only where control will really make a difference in their experiences. Sometimes users may request

numerous “local” or specific features without sufficiently recognizing “global” design imperatives, resulting in a “kitchen sink” otucome (as in “throw in everything but the kitchen sink”).

One alternative to asking users to suggest features is to observe users interacting with prototypes, and infer what they want (and don’t want) from their actions. Often, their actions speak louder than their words. It is usually apparent when users get frustrated, even if they don’t articulate their frustration. When we observe users repeatedly making the same “mistake” with a prototype, we sometimes are able to revise the software so that it behaves in the way that users had expected. In early versions of the Alice system, for example, users repeatedly made syntax errors and repeatedly asked for the system to “be smarter” at understanding their typing errors; instead, the system’s designers realized that having to deal with syntax at all was problematic, and built a completely drag-and-drop user interface where it was no longer possible to form a syntactically invalid program.

Other design teams have emphasized participatory methods that engage representative users actively over long periods of time as members of the team [Muller 2002]. These user representatives may have to invest substantial effort to learn more about design constraints and possibilities. Evidence is strong that projects that include users in the design process result in greater acceptance by the broader user community. The greater acceptance may be due to the insights and more accurate data from users, as well as the ego investment and sympathy generated by having user representatives as part of the design team.

**10. Iterate, Iterate - Then Iterate Again**

Another standard principle of user interface design that we would like to re-emphasize for creativity support tools is the importance of iterative design using prototypes. In designing creativity support tools, we put a high priority on “tinkerability” – we want to encourage users to mess with the materials, to try out multiple alternatives, to shift directions in the middle of the process, to take things apart and create new versions.

Just as we want users to iterate their designs, we apply the same principle to ourselves. In developing new technologies, we have found that we never get things quite right on the first try. We are constantly critiquing, adjusting, modifying, and revising. The ability to develop rapid prototypes is critically important in this process. We find that storyboards are not enough; we want functioning prototypes. Initial prototypes don’t need to work perfectly, just well enough for us (and our users) to play with, to experiment with, to talk about.

One thing most system buildings and designers do not have a strong enough appreciation of is the concept of “iterate just enough to do the next test.” It is crucial to be able to quickly

- observe users with a given iteration of a system
- synthesize design changes as a result of that feedback

- implement (and functionally test) those changes to the tool

and then repeat that process. In a perfect world, one could go around this loop once every day or week.

In his book *Serious Play*, Michael Schrage argues that prototypes are especially helpful as conversation starters, to catalyze discussions among designers and potential users [Schrage 1999]. We agree. We find that our best conversations (and our best ideas) happen when we start to play with new prototypes – and observe users playing with the prototypes. Almost as soon as we start to play with (and talk about) one prototype, we start to think about building the next. This process requires both the right tools (to support rapid development of new prototypes) and the right mindset (to be willing to throw out a prototype soon after creating it).

## 11. Design for Designers

By creating you become creative.

In designing new creativity support tools, it is important to design for designers – that is, design tools that enable others to design, create, and invent things [Papert 1980][Resnick 2002] (see also this report's section titled "Creativity Support Tools for *and* by the New Media Arts Community" for a further discussion).

The traditional LEGO construction kit is a model for what we are trying to achieve with new creativity support tools. With traditional LEGO kits, users are provided with a simple set of parts with which they can design and create a diverse collection of constructions. LEGO kits certainly enable users to express themselves creatively, but new computationally-based creativity support tools go further, enabling users to create not only static, structural artifacts but also dynamic, interactive artifacts: music, video, animations, interfaces. Software-based creativity support tools have an added advantage in that they (and their resulting products) can be distributed widely at low cost.

The analogy with LEGO kits also suggests an important counter-example. In recent years, a growing number of LEGO kits highlight a specific construction (such as a Star Wars spaceship or a Harry Potter castle), with many specialized pieces. Although it is possible to use these kits to create a variety of constructions, many kids build the model suggested on the package, or perhaps slight variants, and nothing more. This is analogous to using a paint-by-numbers kit. These kits clearly encourage "hands-on activity," but they are less effective (compared with traditional LEGO kits) at fostering creative thinking. Our goal is to develop technologies that not only engage users in composing artifacts, but also encourage (and support) them to explore the ideas underlying their constructions.

Another example as a model for what our research tries to achieve is paper and pencil as a tool used by creative practitioners, such as architectural designers. Hand-drawn

sketches and diagrams have been found essential for architects' creative reflection [Arnheim 1969][Lawson 1994]. Not only the drawn diagrams, but the process of drawing helps designers engaging in reflection-in-action [Schoen 1983]. Tools have been designed, developed and tested to support such sketching processes in several domains, including architectural design [Gross 1996], software interface and Web page design [Landay 2001], and industrial design [Hoeben 2005]. Fundamental aspects of sketching for creative thinking have been identified and applied in non-diagramic domains, such as writing and movie-compositions, by using two-dimensional spatial positioning as a representation [Yamamoto 2005]

Writing software is a creative activity, and the authors of this report are programmers who try to be creative, so we would like tools that help us creatively write software. This is somewhat recursive; since we want creativity support software-writing tools that help us create creativity support tools for other tasks as well. The tools we create for ourselves should therefore support all of the guidelines discussed above. Therefore, we should follow good software engineering practices so that the tools themselves are easily modified. This has been the focus of much of the research work in creating flexible user interface software tools [Ousterhout 1994][Myers 1997][Bederson 2004].

## 12. Evaluation of Tools

One important issue with the design of creativity support tools is how they can be evaluated (see this report's section titled "Creativity Support Tool Evaluation Methods and Metrics"). How do you know if a tool is being helpful? Human-computer interaction professionals are used to measuring the effectiveness and efficiency of tools, but how do you measure if it supports creativity? As discussed above, tools that are *not* effective and efficient will probably hinder creativity, but it isn't clear that the reverse will hold. To try to measure creativity, the Silk system designers evaluated many different properties, including the number of different designs produced, the variability of the components used, the variety of questions about the designs from collaborators, etc. [Landay 1996], but these still do not really get at the *quality* of the solution. It is still an open question how to measure the extent to which a tool fosters creative thinking. While the rigor of controlled studies makes them the traditional method of scientific research, longitudinal studies with active users for weeks or months seem a valid method to gain deep insights about what is helpful (and why) to creative individuals [Seo 2005].

We have no delusions that evaluating tools is an easy task, but we also believe that the potential impact of improved tools would be enormous in amplifying and inspiring creativity.

## References

[Arnheim 1969] R. Arnheim, *Visual Thinking*, University of California Press, Berkeley, 1969.

[Bederson 2004] B. B. Bederson, J. Grosjean and J. Meyer. "Toolkit Design for Interactive Structured Graphics," *IEEE Transactions on Software Engineering.* 2004. **30**(8). pp. 535-546.
[Brown 1987] P.S. Brown and J.D. Gould. "An Experimental Study of People Creating Spreadsheets," *ACM Transactions on Office Information Systems.* July, 1987, 1987. **5**(3). pp. 258-272.
[Conway 2000] Matthew Conway, *et. al.* "Alice: Lessons Learned from Building a 3D System For Novices," *Proceedings CHI'2000: Human Factors in Computing Systems,* alice. The Hague, The Netherlands, Apr 1-6, 2000. pp. 486-493. http://www.alice.org.
[Druin 2002] Allison Druin "The Role of Children in the Design of New Technology," *Behaviour and Information Technology*, 21(1) 2002. pp. 1-25.
[Fischer 1994] G. Fischer, K. Nakakoji, Amplifying Designers' Creativity with Domain-Oriented Design Environments, *Artificial Intelligence and Creativity*, Part V, T. Dartnall (Ed.), Kluwer Academic Publishers, The Netherlands, 1994, pp. 343-364.
[Green 1996] T.R.G. Green and M. Petre. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework," *Journal of Visual Languages and Computing.* 1996. **7**(2). pp. 131-174.
[Gross 1996] M.D. Gross, E.Y.L. Do, Ambiguous Intentions: A Paper-like Interface for Creative Design, *Proceedings of Symposium on User Interface Software and Technology* (UIST '96), ACM, 1996. pp. 183-192.
[Hewett 2005] Thomas Hewett, Informing The Design of Computer-Based Environments to Support Creativity, *International Journal of Human-Computer Studies* 63, 4-5, Special Issue on Computer Support for Creativity, E. Edmonds, L. Candy (Eds.), 2005. pp. 383-409.
[Hoeben 2005] Hoeben, A., Stappers, P.J., Direct Talkback in Computer Supported Tools for the Conceptual Stage of Design, *Knowledge-Based Systems Journal*, 2005 (to appear).
[Kelley 2001] T. Kelley, J. Littman, *The Art of Innovation: Lessons in Creativity from IDEO, America's Leading Design Firm*, Currency, New York, NY., 2001.
[Landay 1995] James Landay and Brad A. Myers. "Interactive Sketching for the Early Stages of User Interface Design," *Human Factors in Computing Systems*, Proceedings SIGCHI'95. Denver, CO, May, 1995. pp. 43-50.
[Landay 1996] James A. Landay. *Interactive Sketching for the Early Stages of User Interface Design*. Computer Science Department, Carnegie Mellon University. 1996. Ph.D. Thesis. CMU-HCII-96-105.
[Landay 2001] J.A. Landay, B.A. Myers, Sketching Interfaces: Toward More Human Interface Design, *IEEE Computer*, 34, 3, 2001. pp. 56-64.
[Lawson 1994] B. Lawson, *Design in Mind*, Architectural Press, MA, 1994.
[Lin 2002] James Lin, Michael Thomsen and James A. Landay. "A Visual Language for Sketching Large and Complex Interactive Designs," *ACM CHI'2002 Conference Proceedings: Human Factors in Computing Systems,* Minn, MN, April 20-25, 2002. pp. 307-314.
[Mann 2002] Darrell Mann, *Hands-On Systematic Innovation*, CREAX Press, Belgium, 2002.

[Muller 2002] Michael Muller, Participatory design, In Jacko, Julie and Sears, Andrew (Editors), *Handbook of Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale, NJ. pp. 1051-1068.
[Myers 1996] Brad A Myers and David Kosbie. "Reusable Hierarchical Command Objects," *Proceedings CHI'96: Human Factors in Computing Systems,* Vancouver, BC, Canada, April 14-18, 1996. pp. 260-267.
[Myers 1992] Brad A. Myers. "Demonstrational Interfaces: A Step Beyond Direct Manipulation," *IEEE Computer.* August, 1992. **25**(8). pp. 61-73.
[Myers 1997] Brad A. Myers, *et. al.* "The Amulet Environment: New Models for Effective User Interface Software Development," *IEEE Transactions on Software Engineering.* 1997. **23**(6). pp. 347-365.
[Myers 1998] Brad A. Myers. *The Case for an Open Data Model*. Carnegie Mellon University, School of Computer Science Technical Report. CMU-CS-98-153. August, 1998. http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-153.pdf.
[Myers 2000] Brad A. Myers, Scott E. Hudson and Randy Pausch. "Past, Present and Future of User Interface Software Tools," *ACM Transactions on Computer Human Interaction.* March, 2000. **7**(1). pp. 3-28.
[Nakakoji 2000a] K. Nakakoji, Y. Yamamoto, B.N. Reeves, S. Takada, Two-Dimensional Positioning as a Means for Reflection in Design, *Design of Interactive Systems* (DIS'2000), ACM, New York, NY, 2000. pp. 145-154.
[Nakakoji 2000b] K. Nakakoji, M. Ohira, Y. Yamamoto, Computational Support for Collective Creativity, *Knowledge-Based Systems Journal*, 13, 7-8, December, 2000. pp.451-458.
[Nakakoji 2005] K. Nakakoji, Y. Yamamoto, M. Akaishi, K. Hori, Interaction Design for Scholarly Writing: Hypermedia Structures as a Means for Creative Knowledge Work, *New Review of Hypermedia and Multimedia*, Special Issue on Scholarly Hypermedia, S. Buckingham Shum (Ed.), Taylor & Francis Group Ltd, Oxon, UK, Vol.11, No.1, 2005. pp.39-67, June.
[Nielsen 1993] Jakob Nielsen. *Usability Engineering*. Boston, Academic Press. 1993.
[Ousterhout 1994] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley. 1994.
[de la Rocha 1985] O.L. de la Rocha, The Reorganization of Arithmetic Practice in the Kitchen, *Anthropology and Education Quarterly*, 16, 3, 1985. pp. 193-198.
[Pane 2002] John F. Pane and Brad A. Myers. "The Impact of Human-Centered Features on the Usability of a Programming System for Children," *Human Factors in Computing Systems,* Extended Abstracts for CHI'2002. Minneapolis, MN, Apr 1-6, 2002. pp. 684-685. http://www-2.cs.cmu.edu/~pane/research.html.
[Papert 1980] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. New York, Basic Books. 1980.
[Repenning 2004] Alex Repenning and A. Ioannidou. "Agent-Based End-User Development," *Communications of the ACM.* 2004. **47**(9). pp. 43-46.
[Resnick 1993] M. Resnick. "Behavior Construction Kits," *Communications of the ACM.* July, 1993. **36**(7). pp. 64-71.
[Resnick 1996] M. Resnick, F. Martin, R. Sargent and B. Silverman. "Programmable Bricks: Toys to Think With," *IBM Systems Journal.* 1996. **35**(3-4). pp. 443-452.

[Resnick 2000] M. Resnick, R. Berg and M. Eisenberg. “Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation,” *Journal of the Learning Sciences.* 2000. **9**(1). pp. 7-30.
[Resnick 2002] M. Resnick. “Rethinking Learning in the Digital Age.” *The Global Information Technology Report: Readiness for the Networked World.* G. Kirkman, Ed. 2002: Oxford University Press.
[Resnick 2005] M. Resnick and B. Silverman. “Some Reflections on Designing Construction Kits for Kids,” *Proceedings of Interaction Design and Children conference,* Boulder, CO, 2005.
[Schrage 1999] Michael Schrage. *Serious Play: How the World's Best Companies Simulate to Innovate*. Harvard Business School Press. 1999.
[Selker 2005] Ted Selker, Fostering Motivation and Creativity for Computer Users, *International Journal of Human-Computer Studies* 63, 4-5, Special Issue on Computer Support for Creativity, E. Edmonds, L. Candy (Eds.), 2005. pp. 410-421.
[Seo 2005] Jinwook Seo and Ben Shneiderman, Knowledge Discovery in High Dimensional Data: Case Studies and a User Survey for an Information Visualization Tool, In press 2005.
[Sheil 1983] Beau Sheil. “Environments for Exploratory Programming,” *Datamation.* February, 1983.
[Shneiderman 2000] B. Shneiderman. “Creating creativity: User interfaces for supporting innovation,” *ACM TOCHI.* 2000. **7**(1). pp. 114-138.
[Shneiderman 2004] Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Reading, MA, Addison Wesley. 2004.
[Schoen 1983] D.A. Schoen, *The Reflective Practitioner: How Professionals Think in Action*, Basic Books, New York, 1983.
[Shirouzu 2002] H. Shirouzu, N. Miyake, H. Masukawa, Cognitively Active Externalization for Situated Reflection, *Cognitive Science*, 26, 4, 2002. pp. 469-501.
[Stylos 2004] Jeffrey Stylos, Brad A. Myers and Andrew Faulring. “Citrine: Providing Intelligent Copy-and-Paste,” *ACM Symposium on User Interface Software and Technology, UIST'04,* Santa Fe, NM, October 24-27, 2004. pp. 185-188.
[Stylos 2005] Jeffrey Stylos and Brad A. Myers. *Using Examples to Help Programmers Learn APIs*. 2005. Submitted for Publication.
[Terry 2002] M. Terry, E. Mynatt, Side Views: Persistent, On-Demand Previews for Open-Ended Tasks, *Proceedings of UIST 2002*, ACM Press, 2002. pp. 71-80.
[Terry 2004] M. Terry, E. Mynatt, K. Nakakoji, Y. Yamamoto, Variation in Element and Action: Supporting Simultaneous Development of Alternative Solutions, *Proceedings of CHI2004*, ACM Press, 2004. pp. 711-718.
[Turkle 1990] S. Turkle and S. Papert. “Epistemological Pluralism,” *Signs.* 1990. **16**(1).
[Yamamoto 2005] Y. Yamamoto, K. Nakakoji, Interaction Design of Tools for Fostering Creativity in the Early Stages of Information Design, *International Journal of Human-Computer Studies* 63, 4-5, Special Issue on Computer Support for Creativity, E. Edmonds, L. Candy (Eds.), 2005. pp. 513-535.
[Zhang 1997] J. Zhang, The Nature of External Representations in Problem Solving, *Cognitive Science*, 21, 2, 1997. pp. 179-217.