Multimed Tools Appl (2007) 32:139–159
DOI 10.1007/s11042-006-0062-y

Toolkits and interface creativity

Saul Greenberg

Published online: 9 November 2006

Abstract Interface toolkits in ordinary application areas let average programmers rapidly develop software resembling other standard applications. In contrast, toolkits for novel and perhaps unfamiliar application areas enhance the creativity of these programmers. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a 'language' to think about these new interfaces, which in turn allows them to concentrate on creative designs. This is important, for it means that programmers can rapidly generate and test new ideas, replicate and refine ideas, and create demonstrations for others to try. To illustrate this important link between toolkits and creativity, I describe example groupware toolkits we have built and how people have leveraged them to create innovative interfaces.

Keywords Rapid prototyping · Interface toolkits · Creativity · Innovative interfaces · Groupware

1 Introduction

Rightly or wrongly, most software companies expect their programmers to create and design product interfaces. These can range from variations of existing interfaces to traditional systems, to novel interfaces for new products based on innovative concepts. Yet relying on programmer creativity can be a problem for two reasons. First, most programmers are trained in algorithmic construction rather than creative design. For example, while creative disciplines such as arts and industrial design use the design studio as the central pedagogy to train students in creativity and critique, these studios are extremely rare in computer science programs. Second, while computers are arguably the most flexible medium ever produced, the tools available to programmers—the libraries and interface toolkits—inhibit truly inventive designs. If programmers want to do something different, they often have to work around system limitations and do their programming at a

S. Greenberg (✉)
Department of Computer Science, University of Calgary, Calgary, AB T2N 1N4, Canada
e-mail: saul@cpsc.ucalgary.ca

very low-level. Because this type of programming is costly, it often goes undone or it becomes a poorly functioning hack. Because programmers lack appropriate training and only have conservative tools, most of the designs they produce are rather mundane copies and uninteresting variations of interfaces that have been around for decades. The consequence is that innovative interface design has been stifled.[1]

In this paper, I argue that good toolkit design[2] can enhance programmer creativity. The premise is that if we give everyday programmers good tools and building blocks to innovative areas, then these tools will become a language that influences their creative thoughts [30]. Simple ideas become simple for them to do, innovative concepts become possible, and designs will evolve as a consequence.

I will support this premise through a case study of real time groupware, i.e., where people interact in more or less real time through technology. First, I will explain why groupware has evolved surprisingly slowly as a product because simple ideas were just too hard for average programmers to replicate and vary. I then illustrate how the introduction of several toolkits profoundly changed the ability of students within our laboratory to rapidly prototype and evolve groupware design. Along the way, I will reflect on the role of tools and creativity in how science technology develops over time.

2 Why groupware has not yet fulfilled its potential

2.1 Comparing desktop productivity tools, hypertext and groupware

The first true vision and implementation of real time groupware happened at the Fall Joint Computer Conference in 1968, where Douglas Engelbart demonstrated many important concepts including terminal-sharing, multiple pointers and turn-taking over shared displays, and audio/video conferencing [7]. This tour-de-force was far ahead of its time, and it was not until 15 years had passed that a few other researchers began replicating and extending Engelbart's ideas, most notably Sarin's [23] and Foster's [8] PhD theses. Shortly afterwards, the field of Computer Supported Cooperative Work (CSCW) formed (late 1980s). A veritable explosion of research followed, and CSCW is now considered a relatively mature discipline. In spite of this history and the many research advances made since 1968, groupware has *not* made many inroads into the everyday world. Why is this the case?

To put groupware's failure into perspective, we can compare it to the advances made in two other now mature research disciplines, both also introduced and first implemented by Engelbart in 1968: desktop productivity tools and hypertext (Table 1). We have seen desktop productivity tools take off in the early 1980s with early word processor products such as the Xerox Star and Apple's MacWrite. These heralded the new genre of desktop publishing for the masses, which in turn gave life to the new industry of personal computing. Hypertext, originally popularized by the Apple Hypercard system, exploded into the mainstream with the introduction of the World Wide Web. Both areas have many mature applications and products, have undergone massive deployment to a very broad end-user audience, and are considered a significant success (Table 1, left side). There is no

[1] This article is based on a keynote plenary talk presented at the CRIWG '2003 9th International Workshop on Groupware and expands on the summary included in the proceedings [11].

[2] I use the term *toolkit* fairly liberally, where it can include SDKs, APIs, widget sets, interface builders, component libraries, development environments, and so on. The defining feature is that the toolkit should encapsulate interface design concepts in a way that makes it easier for the programmer to build those designs.

Table 1 A comparison of desktop productivity tools, hypertext and real-time groupware

Desktop productivity/hypertext	Real time groupware
All are mature research disciplines	
Mature applications	Prototypes and early products
Many products	Few products
Massive deployment	Poor deployment
Commercial success	Risky venture
Broad audience	Early adopters and people with great needs
Society has changed	Little effect except for Instant Messaging

question that society has changed as a consequence. Yet when we look at real time groupware, little has happened since Engelbart. Instant Messaging is likely the *only* conferencing system that has had a significant impact, yet it is little more than text-chatting augmented with a simplistic presence indicator. While we are now seeing such systems augmented with video and other facilities, they tend to be unreliable, unimaginative and awkward. Other groupware systems are few and far between, are mostly prototypes and early products, have quite poor deployment (mostly to early adopters or to those with great needs), and are considered a risky business venture. Excepting Instant Messaging, they have had little effect on society (Table 1, right side). This failure of groupware is quite surprising, for groupware's potential to eliminate distance barriers and to augment group work would seem far more compelling to society and likely to succeed than desktop productivity tools and hypertext.

2.2 The role of toolkits in groupware failure

While there are many reasons that we can ascribe to the success of desktop publishing and hypertext vs. the failures of groupware, let us consider these areas from a programmer's point of view. Graphical user interface toolkits, which have been around since the 1970s, significantly eased a programmer's task of creating desktop applications. These toolkits often include widgets or other encapsulated behaviors that let a programmer simply drop in a well-developed component into an application. As a consequence, there are thousands, if not millions, of productivity applications—some commercial, some built for fun, some as student learning exercises. Similarly, in the 1980s Apple's Hypercard let amateurs with barely any programming ability rapidly author interesting hypertext stacks. Teachers, for example, widely used Hypercards to develop self-directed learning modules for students as well as animated simulations to bring learning concepts to life. The World Wide Web took this to another level: the simplicity of HTML, its robustness to errors (i.e., by ignoring incorrect HTML commands and syntax vs. crashing or stopping the program), and the wide availability of a good graphical web browser meant that people with minimal computer experience could immediately author their own web site after learning just a handful of HTML commands. The authoring threshold was reduced even further with the introduction of 'what you see is what you get' web page editors. What is important is that these tools let everyday people, rather than only advanced programmers, develop quite sophisticated hypertext systems. This has reached the point where these kinds of tools are now taught and used in elementary schools as part of a basic computer literacy program.

Yet when we look at groupware, programming tools are back in the dark ages. Groupware development in non-research settings requires a highly trained programmer adept at writing low-level code. The programmer's task often includes implementing a network protocol atop of sockets, dealing with multimedia capture and marshalling (e.g., audio and video), writing various compression/decompression modules for information transmission, worrying about distributed systems issues such as concurrency control, developing a session management protocol so participants can create, join and leave conferences, and creating some kind of persistent data store so that information is retained between sessions. This list goes on and on.

I argue that the key technical problem behind groupware failure is that average programmers (and end-users as developers) do not have sufficient tools to design, prototype and iterate real time groupware. Current commercially available development tools are far too low-level. This has several serious implications:

– Most programmers eschew groupware development because it is too hard to do.
– Those who do decide to develop groupware place most of their creative efforts into low-level implementation concerns.
– Resulting designs are often fairly minimal ones, with little attention paid to necessary design nuances (even ones well known in the CSCW literature) simply because so-called 'advanced features' are too hard to implement.

The consequence of inadequate development tools is that—excepting graduate students taking advanced courses in CSCW or researchers within the relatively small specialist CSCW community—there has been relatively little evolutionary development and dissemination of groupware systems.

2.3 Groupware considered within the BRETAM model of technology development

To further explain why this lack of tools is an important bottleneck in groupware, we can situate groupware's evolution in Gaine's BRETAM phenomenological model of how science technology develops over time (Fig. 1) [9]. The model states that technology-oriented research usually begins with an insightful and creative *breakthrough* (the B in

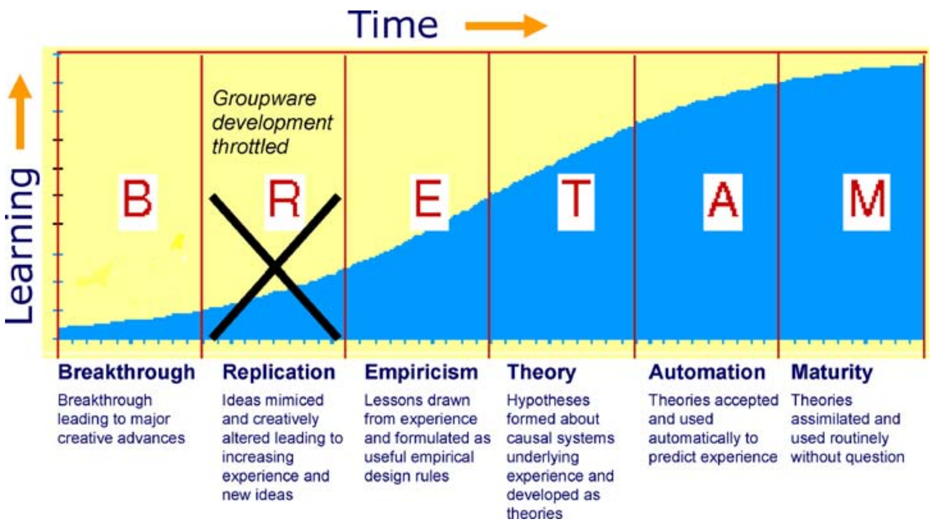

Time
Learning
Groupware development throttled
B
R
E
T
A
M
Breakthrough
Breakthrough leading to major creative advances
Replication
Ideas mimiced and creatively altered leading to increasing experience and new ideas
Empiricism
Lessons drawn from experience and formulated as useful empirical design rules
Theory
Hypotheses formed about causal systems underlying experience and developed as theories
Automation
Theories accepted and used automatically to predict experience
Maturity
Theories assimilated and used routinely without question

Fig. 1 BRETAM model of the development of science technology. Modified from [9]

BRETAM) that leads to a major new way of thinking about that technology. Engelbart's vision of desktop productivity, hypertext, and groupware are all excellent examples of breakthroughs that caused people to rethink the role of the computer from number-cruncher to a machine that augments human intellect [7]. *Replication* occurs when others mimic the ideas, either by re-implementing them or (as is more common) by creatively altering the original idea in both small and large ways. Replication is research, where the community gains increasing understanding and experience in the core factors (including human factors) behind the technology, which in turn suggests new ideas that they can apply to it. This naturally leads to *empiricism*, where the lessons drawn from experience are formulated and codified as useful empirical design rules. These include textual rules (e.g., the myriad of guidelines produced in the earlier days of human computer interaction) and tools that encapsulate ideas that work well from experience (e.g., interface building blocks such as widgets). As more experience is gained, *theories* are developed that hypothesis about the causal reasoning behind the guidelines. Over time, *automation* occurs as these theories are accepted, with *maturity* of the science technology reached when the theories are used routinely and without question [9].

The big catch is that because groupware is hard to build, we have essentially throttled its development at the replication stage. This has minimized product invention and innovation as well as hands-on experience to all but the CSCW research community and a few well-resourced developers. Thus the necessary creativity leading to product evolution was stifled.

Another way of saying this is that all design disciplines recognize the importance of creative media and media tools in how the 'average' designer thinks. Echoing the Sapir-Whorf Hypothesis in linguistics [30], which states that language influences how we think and behave, the media tools becomes a language that influences creative thoughts, that indicates design directions, and that lets them concentrate on their design. Groupware has, in general, failed to give people this creative media.

2.4 Groupware toolkits for the everyday programmer

The solution is that we as a community must develop groupware toolkits appropriate for everyday programmers, where they can use it to develop their own creative ideas. By appropriate, I mean that a good groupware toolkit should:

– Be embedded within a familiar platform and language in common use so that people can leverage their existing knowledge and skills.
– Remove low-level implementation burdens common to all groupware platforms (e.g., simplified access to communications, data sharing, concurrency control, session management).
– Minimize housekeeping and other non-essential tasks (e.g., hiding of details, or automating tasks that would otherwise have to be coded).
– Encapsulate successful design concepts known by the research community into programmable objects so they can be included with little implementation effort.
– Present itself through a concise API that encourages how people should think about groupware.
– Make simple things achievable in a few lines of code, and complex things possible.

I believe that effective groupware toolkits not only make it possible for others to develop groupware, but also enhance their creativity. If we remove low-level implementation burdens and supply appropriate building blocks, we provide people a language to think about groupware [30], which in turn allows them to concentrate on replicating and varying designs in creative ways, thus overcoming the replication bottleneck in the BRETAM model [9].

While some may question this premise as overly simplistic, we should recognize that toolkits in other domains have a proven record of enhancing creativity for the general programming community. We already mentioned GUI toolkits for desktop productivity applications. For example, Visual Basic supplies a large set of interface components (widgets), an interface builder for laying them out on the display, and a relatively easy to learn programming language. Because GUI toolkits encourage programmers to think in terms of widgets, programmers have created a plethora of applications that 'glue' these components together in interesting ways [22]. Oddly enough, 'serious' programmers often snub VB, likely because it is perceived as too simple a language! Within the hypertext domain, Macromedia's Flash encourages both programmers and non-programmers to think in terms of scripted animations. Because Flash makes it easy to do, we now see a proliferation of many quite amazing animations on the Web.

One final point: the creativity I am talking about is not the 'big C' creativity usually associated with breakthrough ideas, or the result of people explicitly learning exercises or formal steps to promote their own creativity (e.g., De Bono's lateral thinking [5] or Shneiderman's Genex [24]). Rather, it is 'small c' creativity where people naturally develop their own ideas through copying and varying the ideas of others in interesting and unorthodox ways, and through testing their ideas via prototypes and learning from their experiences of what works and what does not. In essence, this is the type of creativity often described as part of iterative interface development. Surprisingly, while there is a large literature on how to extract user requirements that form the heart of initial interface prototypes, and on methodologies that test systems for uncovering interface bugs, there is little written on how one creates the basic prototype from the original requirements, or how one actually goes about discovering solutions to interface bugs. This is where 'small c' creativity is mostly seen.

2.5 A common pattern: how toolkits afforded an explosion of rapid designs

To illustrate the link between groupware toolkits and creativity, I will provide in the remainder of this paper several examples of groupware toolkits we have built and how students—both graduates and undergraduates—have leveraged these tools in their own creative work. Before doing so, I want to explain a recurring pattern that emerged over the years in our group's investigation of the human and technical factors of groupware, and how recognizing this pattern has led to our appreciating the value of good tools. The pattern is illustrated in Fig. 2 and described below.

(a) *Human factors perspective*. Our initial goals in our groupware projects are typically oriented towards human factors. Essentially, we wanted to understand how people interact together when using a particular style of yet-to-be developed groupware application. We would then generalize this understanding to inform other groupware designs (Fig. 2a).
(b) *Initial prototype*. Next, we would set about building the first version of the groupware application. This typically involved huge effort as measured by lines of code, time, learning, failed attempts, debugging, and so on. In spite of this effort, the result was often a fragile and rudimentary system (Fig. 2b).

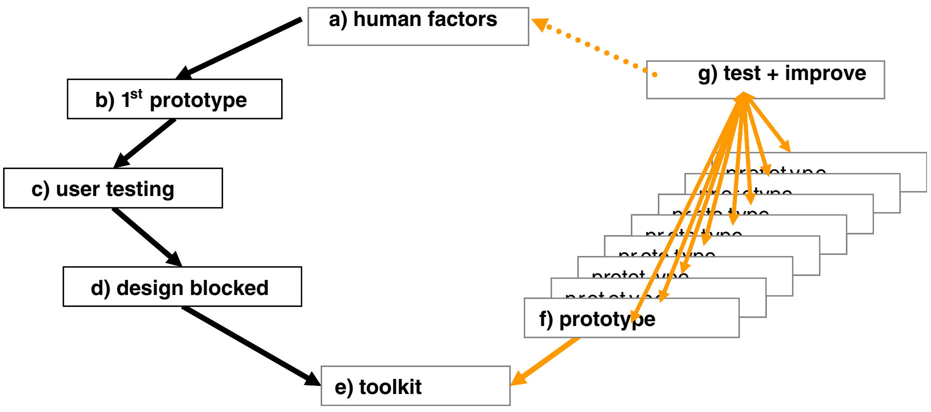


Fig. 2 A recurring pattern: how toolkits promote rapid prototype designs

(c) *Prototype testing*. We would then have people try out this prototype. Because it is an early design, we often saw major usability problems that required fixing (Fig. 2c).

(d) *Design blocked for iterative prototyping*. To fix these usability problems, we would then iteratively redesign the prototype. Yet this often proved impractical to do. The prototype code was often too complex to change, or the system itself was too fragile. Redesigning from scratch, while possible, was onerous due to the time involved (Fig. 2d).

(e) *Retrenchment: building a groupware toolkit*. We would then realize that—in the long run—iterative prototyping would be far easier if we took the time to build a robust toolkit. Thus we would set ourselves a new technically oriented goal, where we would delve into the challenges of understanding and building this toolkit and its accompanying run-time architecture. This often meant that we had to defer work on our main human factors goal (Fig. 2e).

(f) *The payoff: rapid prototyping*. After building the toolkit, we would release it to our internal community. There would then be an explosion of activity. Those with core interests in the human factors challenges would rapidly develop and test a variety of groupware interaction techniques and applications. Those with interests lying elsewhere would often create innovative groupware applications as a side project just to satisfy their own curiosity (Fig. 2f).

(g) *Testing, improvement and dissemination*. Because we would develop the toolkit and applications side by side, we would bring well-tested good application ideas back into the toolkit as building blocks that could be trivially included in other applications. Examples included common architectural features, widgets, interface components, and interaction techniques. Of course, both prototype testing and our experiences in rapid prototyping fed back into our understanding of the basic human factors behind groupware design (Fig. 2g), thus achieving our original project goal.

In the remainder of this paper, I will briefly summarize the experiences my research group and others have had with several toolkits that we developed for three groupware domains: real time distributed groupware, single display groupware (SDG), and physical user interfaces. These experiences serve as case studies that show how toolkits helped promote creativity in rapid prototyping and in idea replication and variation, thus breaking out of the replication bottleneck in the BRETAM model.

3 Toolkits for real time distributed groupware

My first foray into groupware echoed the above pattern. In 1989, I and my students decided to build a simple drawing application for distributed participants designed around John Tang's human factors observations of how small design teams draw together [26]. We were primarily interested in understanding group interaction, where we wanted to develop effective groupware interaction techniques and generalize our understanding as design requirements (i.e., step a in Fig. 2).

The result was GroupSketch [18], a minimalist multi-user sketching program written over several months by student Ralph Bohnet (step b in Fig. 2). While simple in functionality, the actual program was quite complex in its underlying 'plumbing'. Its more than 5,000 lines of code had to deal with many things: setting up the basic communication architecture and protocol for data exchange, creating a session manager that would let people join an existing conference, managing an event stream that handled simultaneous local and remote user actions, creating labeled telepointers for each participant, creating the actual drawing surface and actions, handling graphical problems associated with rapid updates when people drew simultaneously, and so on. All this had to work efficiently so that the participants would see no noticeable lag, and this required quite a bit of time and experimentation to get right. Shortly after, student Roseman built GroupDraw [18], an object-based drawing system. As with GroupSketch, the majority of the GroupDraw programming effort went into developing the underlying architecture and worrying about performance issues vs. designing the actual group-drawing interface.

Both systems worked well enough to do rudimentary user testing and give us insights into what we wanted to do next (step c in Fig. 2). However, the programming design shortcuts we had taken in constructing the underlying plumbing code meant that these programs were just too large and too finicky to extend. Yet redoing the plumbing from scratch was far too costly to do for a single application. While we had good design ideas, we were blocked from pursuing them in an effective and efficient manner (step d in Fig. 2).

Consequently, we turned our efforts into developing GroupKit, a toolkit for building distributed real time groupware applications [16] (step e in Fig. 2). Our experiences with GroupSketch and GroupDraw helped us identify elements common to real time distributed groupware applications, and our GroupKit design would provide these elements to the programmers.

- A run-time architecture automatically managed processes, their interconnections, and communications; thus programmers did not have to do any process or communications management. This came for free without any further programming.
- Session managers let end-users create, join, leave and manage meetings. A selection of session managers came as pre-packaged interfaces (one is visible on the top right of Fig. 3b), and the programmer could use these 'out of the box.' However, the programmer could craft their own session manager if they wished.
- Network connectivity and distributed data sharing was trivialized. A small set of groupware programming abstractions let a programmer manage all distributed interactions. Through an RPC-like mechanism, the programmer could easily broadcast

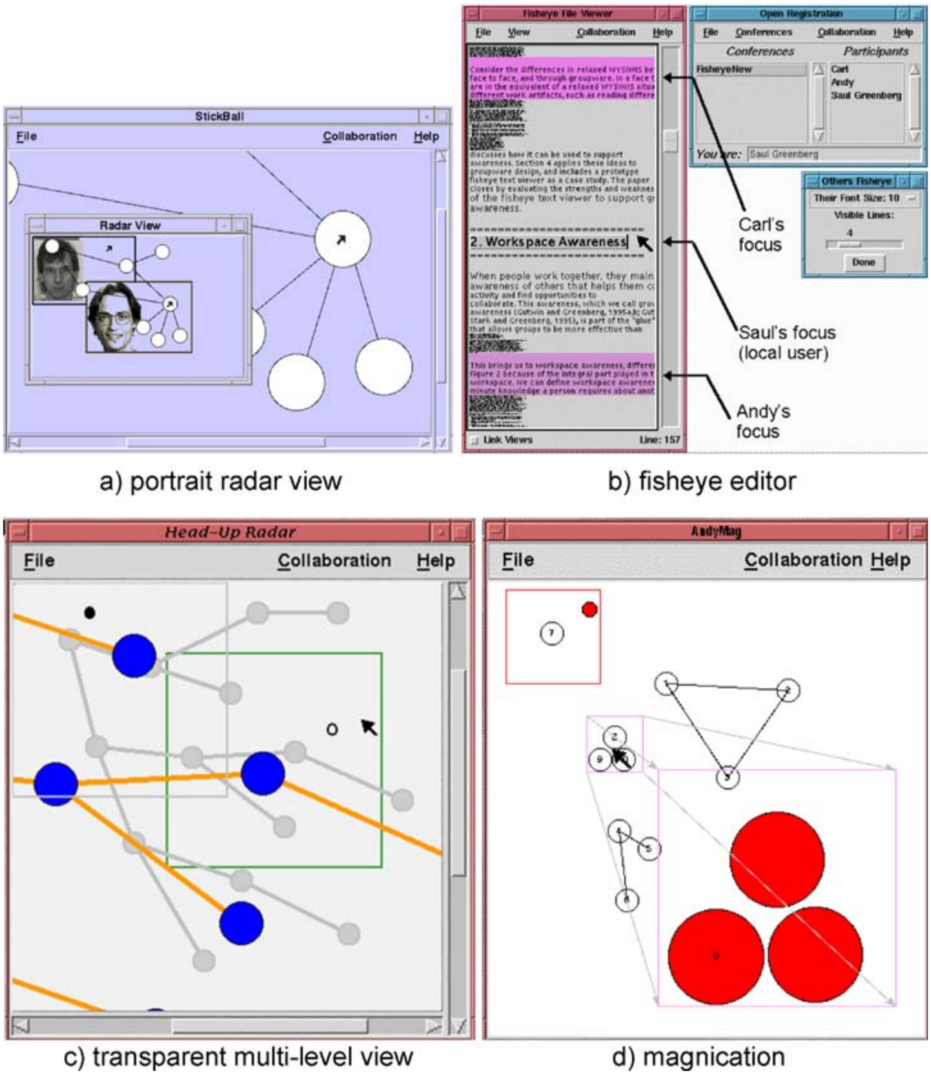
interaction events to selected participants. Alternatively, the programmer could manage interaction via a shared data model: programmers would then think about groupware as a distributed model-view-controller system. Local user actions would change data in the shared model, and remote processes would detect these and use the altered data to generate the view.

– Finally, groupware widgets were included that let programmers add generic groupware constructs of value to conference participants. Our first widgets were telepointers, which a programmer could add with a single line of code. Later widgets included awareness widgets such multi-user scrollbars and radar views.

GroupKit considerably simplified groupware development e.g., using GroupKit we reimplemented GroupSketch and GroupDraw in a few hours using very little code. Other simple groupware tools were similarly rapid to build: a brainstorming tool in 74 lines, a graphical concept map editor in 213 lines, a file-sharing system in 51 lines, and a text-chat system in 80 lines of code. What was more important is that we could now explore various design ideas through rapid prototyping (step f in Fig. 2). For example, our group created a flurry of systems illustrating different methods for supporting awareness within a visual workspace, sometimes turning around several different design ideas in a single day.

Figure 3 shows four of the many example systems illustrating how people within a group could maintain awareness of others' actions [20, 21, 14]. Figure 3a illustrates the portrait radar view, where a miniature overview of the workspace (the inset) shows what others can see within it (the view rectangles), where individual cursors are located, and participant identity via a transparent image embedded within the view rectangle. Figure 3b shows a radically different approach to awareness: while the text contained by this groupware text editor is shrunk to fit the display, multiple fisheye lenses (one for each user) disproportionably magnifies the region around where they are working. Thus a person sees where they and others are working, and can also see the details of their work. Figure 3c illustrates a transparent multi-level view: the view in the foreground is the detailed view where the local person does the work, while the entire background contains an overview of the entire work area showing where all are working. Figure 3d applies an offset magnification effect to groupware: As with Fig. 3b, the main window fits the entire work surface within it as an overview, but the local user can see details of their own work through a magnification lens moved over this surface. What was important is that because we could now test these and other ideas for awareness support, we could quickly determine which ones were worth pursuing and which were not, and how we could improve upon our ideas (step g in Fig. 2). Eventually, the empirical knowledge gained through this creative evolution of prototypes as well as user testing of them let us form a theoretical framework that generalizes and codifies awareness properties in real time groupware work surfaces [20] (full circle: back to step a in Fig. 2).

While this was going on, we were also rapidly prototyping quite novel end-user applications. Figure 4 illustrates three of these. GroupWeb (Fig. 4a) is arguably the first groupware web browser ever created [10]. It allows distributed people to jointly navigate the web, to attach group annotations to a page, and to scroll through the page via a multi-user scrollbar in either a tightly coupled or loosely coupled manner. It also contains telepointers, where the pointer remains over the correct part of the web page even though each page may be formatted quite differently to fit into individual participants' web browsers. TeamRooms in Fig. 4 is a very sophisticated system that was eventually commercialized as TeamWave Workplace [17]. It let people create and enter rooms full of persistent groupware artifacts. As one person entered a room, one is immediately connected



Consider the differences in relaxed WYSIWIS be
face to face, and through groupware. In a face t
are in the equivalent of a relaxed WYSIWIS situa
different work artifacts, such as reading differe

discusses how it can be used to support
awareness. Section 4 applies these ideas to
groupware design, and includes a prototype
fisheye text viewer as a case study. The paper
closes by evaluating the strengths and weaknes
of the fisheye text viewer to support gr
awareness.

2. Workspace Awareness

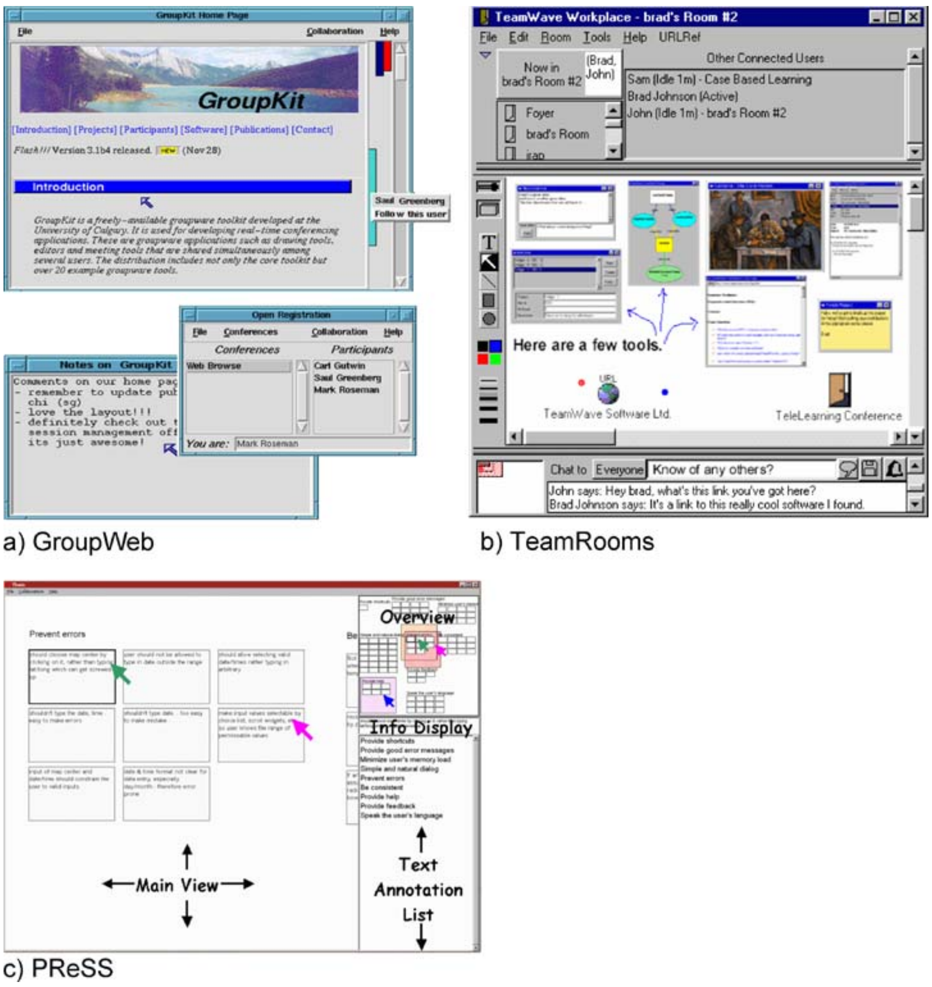
When people work together, they main
awareness of others that helps them c
activity and find opportunities to
collaborate. This awareness, which we call grou
awareness (Gutwin and Greenberg, 1995a,b; Gut
Stark and Greenberg, 1995), is part of the "glue"
that allows groups to be more effective than

This brings us to workspace awareness, differe
figure 2 because of the integral part played in t
workspace. We can define workspace awarenes
minute knowledge a person requires about anot

Fig. 3 Four ideas about awareness. All ideas show where others are working

to all others in the same room. Figure 4c illustrates PReSS, a tool that lets distributed usability engineers transform heuristic evaluation results into problem reports that can be passed onto developers [4]. Of course, these and other sophisticated groupware applications required effort to develop, but because GroupKit gave the basic groupware infrastructure for free, the developers could concentrate on the application design. They could also apply the experiences and human factors gained through the many creative prototypes directly to these products. For example, all three systems in Fig. 4 contain multiple cursors, as well as awareness mechanisms (the multiple scroll bars showing people's relative positions in the document in Fig. 4a, and the radar overviews in Fig. 4b (top right) and Fig. 4c (bottom left).

As well, all this information fed back into GroupKit's evolution: facilities were improved and new ones added as we learnt from our prototyping experiences (the arrows



Flash!!! Version 3.1b4 released. NEW (Nov 28)

GroupKit is a freely–available groupware toolkit developed at the University of Calgary. It is used for developing real–time conferencing applications. These are groupware applications such as drawing tools, editors and meeting tools that are shared simultaneously among several users. The distribution includes not only the core toolkit but over 20 example groupware tools.

```
Comments on our home pag
- remember to update pul
  chi (sg)
- love the layout!!!
- definitely check out t
  session management off
  its just awesome!
```

Sam (Idle 1m) - Case Based Learning
Brad Johnson (Active)
John (Idle 1m) - brad's Room #2

Know of any others?

John says: Hey brad, what's this link you've got here?
Brad Johnson says: It's a link to this really cool software I found.

Provide shortcuts
Provide good error messages
Minimize user's memory load
Simple and natural dialog
Prevent errors
Be consistent
Provide help
Provide feedback
Speak the user's language

Fig. 4 Three major groupware applications developed with GroupKit

joining step f and e in Fig. 2). Similarly, it let us iterate over new toolkits. While GroupKit was very useful for prototyping real time distributed graphical user interfaces, it did not handle multimedia. Consequently, we built a new toolkit called the Collabrary [3] that would let us rapidly prototype multimedia groupware. It provides extremely easy access and manipulation of multimedia information. For example, discovering a video camera and acquiring an image takes two line of Collabrary code. It also provides a straightforward API that lets people distribute this multimedia information between groupware program instances through a shared data model. Similar to GroupKit, students began creating multimedia groupware because it was easy to do. Figure 5, for example, shows two working prototypes developed by Michael Boyle, where each shows a recent history of a remote person's presence. The interface on the left displays presence as an activity graph over time atop the video image. The other interface overlays periodically captured video frames as transparent layers, thus allowing people to get a sense of how one moved around the space. Both programs are less than a page of code, and took a short time to write (e.g.,

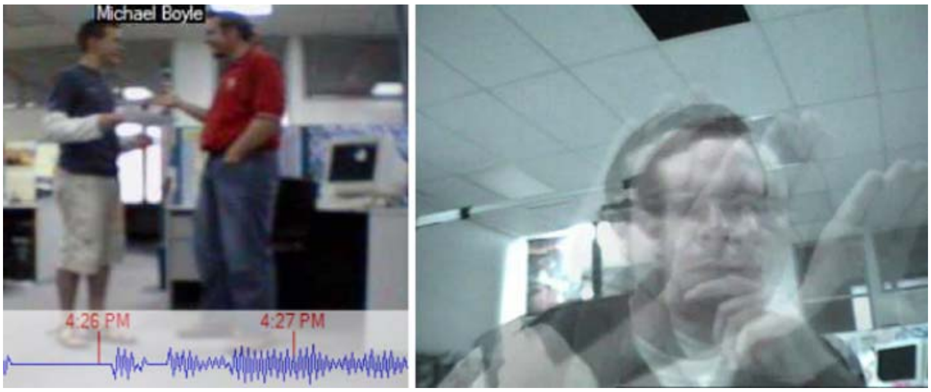


Fig. 5 Two example explorations into multimedia presence history (by M. Boyle)

minutes to hours). As with GroupKit, we also developed fully functional sophisticated applications using the Collabrary, e.g., Michael Rounding's Notification Collage implements a public space, where colleagues can post multimedia information elements onto a real time collaborative surface that all can see [19].

In summary, this case study of real time groupware dramatically illustrates how toolkits fostered creativity within our group. Because people could think about their designs rather than low-level plumbing, the toolkits engendered a culture of rapid prototyping, of idea creation and exploration, and of testing and iterative redesign. We were able to replicate and vary groupware ideas easily, and thus could move to other stages of the BRETAM model as we developed both theory and practice in groupware design.

4 Toolkits for single display groupware

Our second case study concerns interfaces for collocated work. Researchers in Computer Supported Cooperative Work (CSCW) are now paying considerable attention to the design of single display groupware (SDG) i.e., applications that support the work of co-located groups over a physically shared display [2, 25]. Our own work in SDG began with an investigation of transparent menus as an interaction technique that would minimize how people working close together would interfere with each other [31]. Fortunately, Bederson and Hourcade [1] had developed the MID toolkit that lets one access multiple mice from Microsoft Windows'98. While it was still difficult to develop SDG, their toolkit made our own development a reasonable prospect.

The problem was that MID did not work with later versions of Windows, and again we hit the design blockage illustrated in Fig. 2d. Consequently, we decided to re-implement and significantly extend some of the ideas in MID in our own SDGToolkit [28]. SDGToolkit automatically captures and manages multiple mice and keyboards (as does MID), and it also presents them to the programmer as uniquely identified input events relative to either the whole screen or a particular window. Unlike MID, it transparently provides multiple cursors, one for each mouse. To handle orientation issues for tabletop displays (e.g., people seated across from one another), programmers can specify a participant's seating angle, which automatically rotates the cursor and translates input coordinates so the mouse behaves correctly. Finally, SDGToolkit provides an SDG-aware widget class layer that significantly eases how programmers create novel graphical components that recognize and respond to multiple inputs.

With SDGToolkit, simple things are simple. For example, Fig. 6a illustrates a simple drawing application designed by Edward Tse for a square tabletop with four seated people, one per side. Cursors and text labels are oriented appropriately, and the person's mouse behaves correctly given their orientation. It is written in 20 lines of code.

Another example illustrates how students Nicole Stavness and Edward Tse reimplemented Xerox PARC's ToolGlass interaction technique as an SDG widget (Fig. 6b). Each user has two mice. Using the first mouse in with their non-dominant hand, each moves his/her toolglass around. With their other hand and mouse, they click through the lens to choose a color. Their programming effort to manage and identify multiple input devices and to package it up as an SDG widget was relatively small; instead most efforts went into the creative aspects of the ToolGlass graphics. The third example in Fig. 6c shows how undergraduate student Rob Diaz is exploring SDG-equivalents of conventional widgets. Each widget remembers the state of each person. The slider shows all peoples' settings (the thin lines), while one or more people can simultaneously adjust their settings to a new value. SDG checkboxes and radio buttons fill the region by color and area to represent how people have toggled it; as one or more people move over it, a drop-down list shows the state of those people.

More recently, we received a DiamondTouch surface from MERL, which detects multiple simultaneous touches by multiple people and that reports them to a programmer through a basic SDK. We created the DiamondTouch toolkit that wraps this SDK and adds extra capabilities to it, considerably simplifying how people program multi-user/multi-touch applications [6]. Similar to our SDGToolkit, the toolkit identifies multiple inputs on a per-user basis. It generates events that reports when people tap or double-tap the surface, the bounding box surrounding one person's multiple touches, and a set of vectors reporting the signal strength of a person's touches.

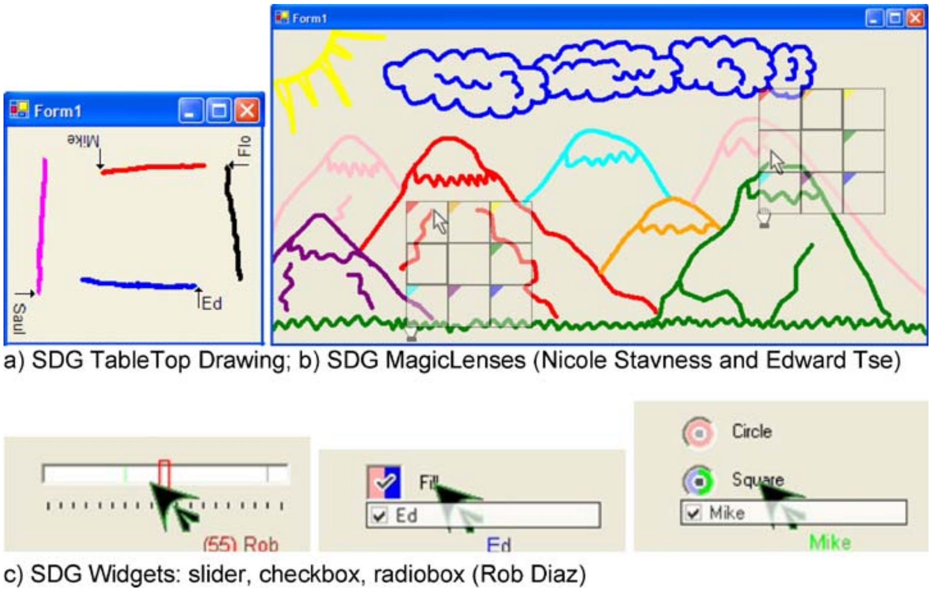

Form1
Mike
Flo
Saul
Ed
Form1
a) SDG TableTop Drawing; b) SDG MagicLenses (Nicole Stavness and Edward Tse)
(55) Rob
Fill
Ed
Ed
Circle
Square
Mike
Mike
c) SDG Widgets: slider, checkbox, radiobox (Rob Diaz)

Fig. 6 Simple applications and widgets developed in the SDG toolkit

To illustrate, Fig. 7 shows SquiggleDraw, a paint program written in approximately 10 min in about 15 lines of code. SquiggleDraw has two interesting properties.

- A person adjusts line thickness on the fly. One draws by changing the bounding region of the drawing with two fingers. One draws thin lines by holding their thumb and forefinger close together, and progressively thicker lines by spreading their fingers apart.
- Up to four people can draw simultaneously, with each person's lines appearing in a different color.

Because of the availability of both the SDGToolkit and the DiamondTouch Toolkit, many other students in our laboratory are now working on single display groupware. Some are 'dabbling' for their own curiosity, but are producing fairly interesting systems. Others are concentrating on quite serious research projects, and are rapidly implementing ideas for hypothesis testing. For example, student Tony Tang combined both the SDGToolkit and the Collabrary to create a tabletop application that handles both co-located and distance-separated participants [27]. Similarly, other students have used the toolkits to develop test environments for evaluating hypothesis of how people use space within SDG (e.g., [29]).

In summary, while our involvement in SDG is fairly recent, the availability of both toolkits meant that we could quickly move into the rapid prototyping stage (Fig. 2f). In turn this gave us insights into hypotheses of SDG use as well as a means to test these hypotheses over actual SDG systems (Fig. 2g, a).

5 Toolkits for physical user interfaces

Our third and final case study moves to quite a different aspect of groupware design that includes physical user interfaces.

In the last few years, researchers have developed groupware designs that include physical user interfaces augmented by computing power. These typically involve ambient displays for showing awareness information, or collaborative physical devices that are controlled by multiple (perhaps distributed) people. While this is an exciting new area, everyday programmers face considerable hurdles if they wish to create even simple physical user interfaces. Most lack the necessary hardware training. Those willing to learn find themselves spending most of their time building and debugging circuit boards, firmware and low-level wire protocols rather than on their physical user interface designs.

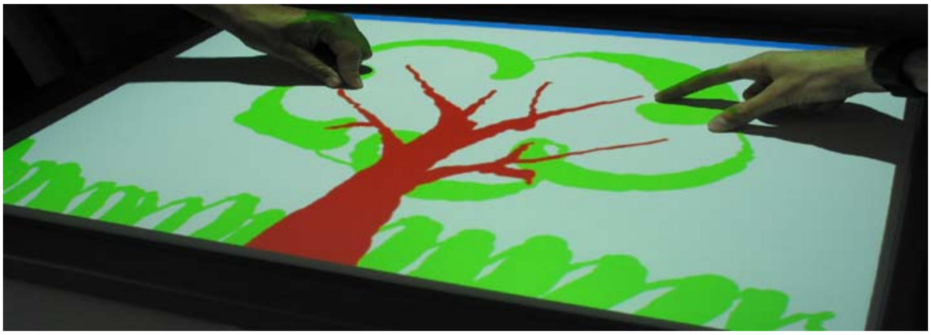


Fig. 7 Two people using SquiggleDraw

The problem is that we have not provided programmers with adequate building blocks for rapidly prototyping physical user interfaces. This leaves them in a position similar to early GUI researchers who had to build their widgets from scratch, or to early graphics researchers who had to build their 3D environments by brute force. Given this onerous situation, it is no wonder that most research on physical user interfaces come from top researchers at major university and industrial research laboratories.

Our own efforts in physical user interface design for groupware quickly ran into the design bottleneck in Fig. 2. In 1999 Japanese visitor Hideaki Kuzuoka and I built the Active Hydra system, an always-on video-based media space augmented by physical devices [15]. As Fig. 8 (left) illustrates, the Active Hydra embodies a single remote person by showing a video and audio connection to that person within a device, where the communication channel is opened as a function of proximity. If people are close to the device, he/she can see and hear the remote person in full fidelity. As one moves away, audio is disabled; even further away, and the video goes into a 'glimpse' mode giving only a partial view into the remote site. A figurine immediately in front of the Active Hydra also shows the availability of the remote person by the direction it is facing: it faces forward when the remote person is present, and away when one is absent. The other figurine lets the local person explitely control their privacy by how they position it relative to the media space. The system was built from scratch, and took several months to do: Kuzuoka mainly handled the hardware design, while I handled the software aspects. When Kuzuoka left, his hardware expertise went with him. Our attempts to modify the Active Hydra were somewhat disastrous, leading to its current state as shown on the right of Fig. 8. Clearly, the design bottleneck was a serious issue for us.

As with our other case studies our solution was to develop a toolkit, in this case for the rapid development of physical widgets, or *phidgets* [13]. Our approach was to provide programmers with pre-packaged hardware devices that can be 'dropped into' software applications. This familiar programming paradigm is directly analogous to how graphical user interface (GUI) widgets are programmed. For example, if a programmer wants to build an interface that uses a servo motor, she would just drop in a widget and/or component that

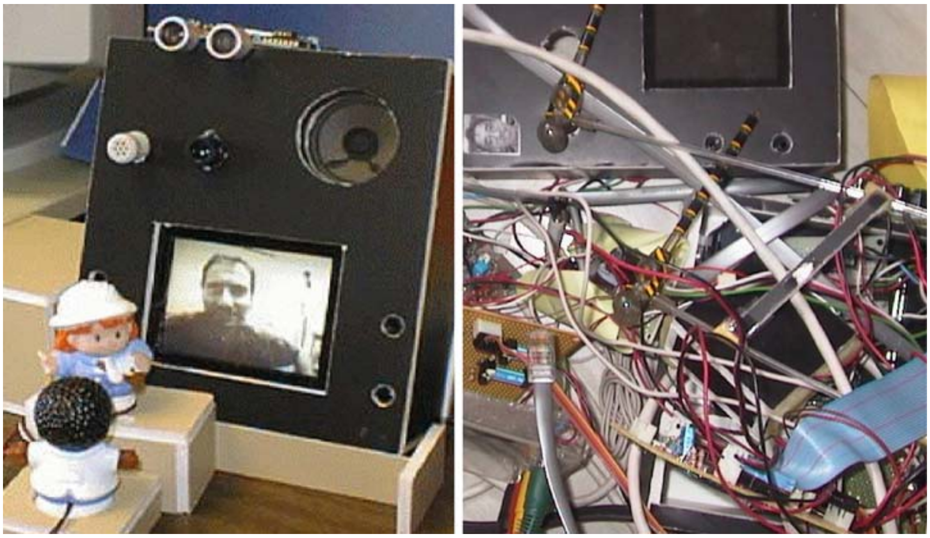


Fig. 8 Before and after view of the active hydra unit

corresponds to the motor. She could rotate the motor to a specific angle either by directly using the graphical control or through a simple API e.g., ServoMotor.Position=90. Similarly, input devices such as sensors, switches, and RFID tags report changes to their state as simple input events, and programmers have the ability to show these states by dropping a widget into the graphical interface.

I gave phidgets to undergraduate students with no hardware expertise to see what they could do with them. These typically took the form of a short two to three week assignment. The results were remarkable. While some students replicated examples of physical user interfaces reported by other researchers, most produced their own innovative designs [12, 13].

A few example of the many groupware systems they built are illustrated in Fig. 9. Figure 9a is a notification device, where the blooming state of an artificial flower shows the state of another person's interest in communication. The next two are physical instantiations

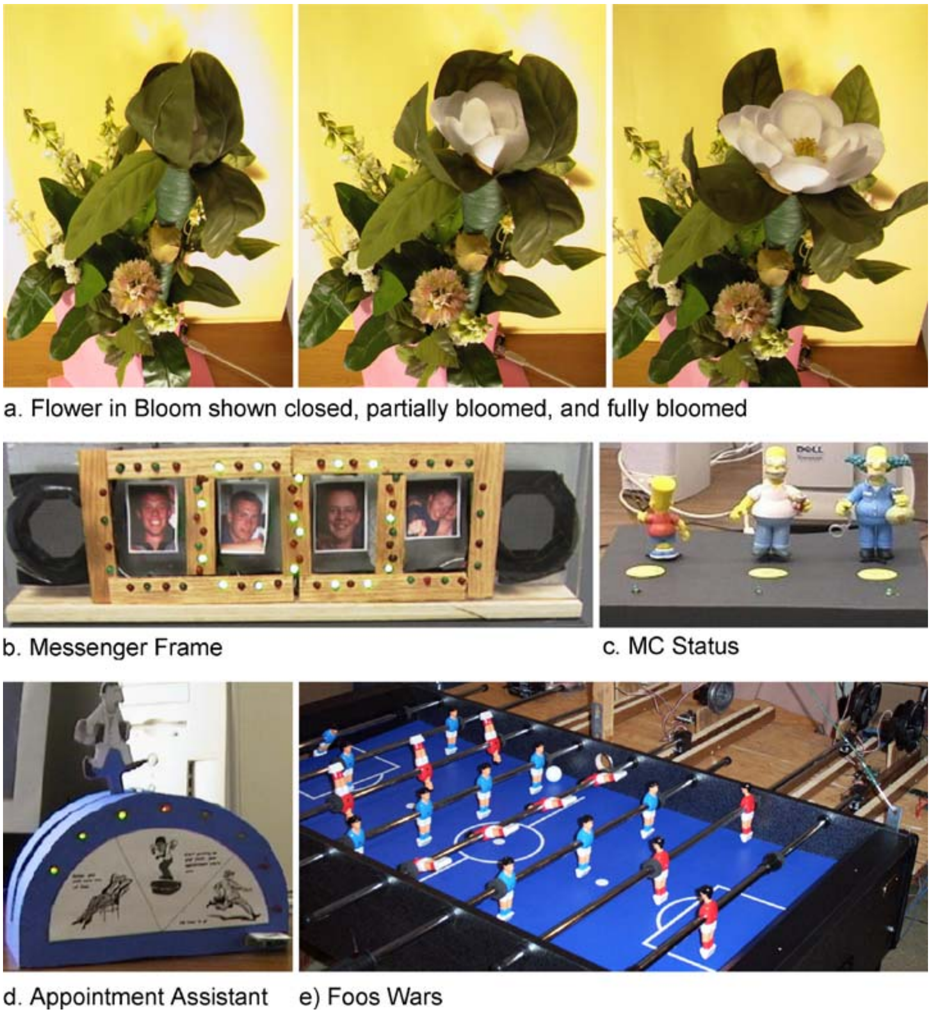

a. Flower in Bloom shown closed, partially bloomed, and fully bloomed
b. Messenger Frame
c. MC Status
d. Appointment Assistant
e) Foos Wars

Fig. 9 Example undergraduate projects on physical user interfaces for groupware

of MSN Instant Messenger. In Messenger Frame by Mike Hornby-Smith (Fig. 9b), a contact's photo is lit up and a sound cue generated as that contact appears online or changes their activity status. One sends a message directly to that contact by touching his photo. In MC Status by Christian Leith (Fig. 9c), contacts are represented by figurines. Offline figurines face the wall, and online figurines face forward. Touching the area in front of the figurine initiates a message. Appointment Assistant by Zaid Alibhai (Fig. 9d) is an ambient appointment reminder display that interacts with a user's on-line calendar to remind them of upcoming appointments. As an appointment approaches, the figure on the top of the display moves along the scale and LEDs light up to further indicate the time remaining before the next appointment. FoosWars by Mike Larke and Mike Clark (Fig. 9e) is a soccer table that involves distributed as well as local players. One person plays on the physical table while the other plays over the web. The remote player has a live aerial view of the table captured via a web camera located above the table, and directly manipulates his or her players through use of physical sliders. Descriptions of other example student phidget projects related to groupware are found in [12], while many videos illustrating how they work are available at http://grouplab.cpsc.ucalgary.ca/phidgets/gallery.

In summary, our work with physical user interfaces for groupware is perhaps the most dramatic example of how toolkits afford creativity. Our dearth of expertise in hardware meant that we were almost totally blocked from continuing our work in this area. Yet the availability of the phidgets toolkit meant that we could create quite interesting and sophisticated physical interfaces in spite of our ignorance. Even average programmers, as represented by our undergraduate students, were creating novel interfaces in a very short amount of time, many of which are now used as exemplars in research publications.

6 Closing thoughts

6.1 BRETAM and toolkits

Using groupware as a case study, we now see how its development parallels Gaines' BRETAM phenomenological model of developments in science technology [9]. As described earlier, the model states that technology-oriented research usually begins with an insightful and creative breakthrough, followed by many (often painful) replications and variations of the idea. Empiricism then occurs when people draw lessons from their experiences and formalize them as useful generalizations. This continues to theory, automation and maturity. I argued that the lack of widely deployed and generally accepted groupware toolkits has, until recently, throttled the replication stage of BRETAM because creative development and innovation of groupware proved rare: even simple ideas were too hard to implement. This resulted in poor evolution of research and creative product ideas.

We also see how the construction of various toolkits within our laboratory afforded creative research in this area. Toolkits make it easier for researchers to create new breakthroughs through rapid prototyping of many new ideas. They let others replicate and evolve ideas reported in the literature. They also let researchers move more easily into empiricism by making it easy to create different versions of testable systems. Groupware evolved as a consequence. To summarize:

(a) *Breakthroughs*. One cannot build a toolkit to create a breakthrough. However, each breakthrough can suggest a new area of toolkit design.

(b) *Replication*. Toolkits naturally support replication. They ease the scientist's task of replicating and varying not only their own design ideas, but those of others.
(c) *Empiricism*. As the toolkit matures, it codifies empirical knowledge as usable design constructs that embodies a sort of 'design rule.' That is, the toolkit produces constructs that have been shown by prior experience to have value.
(d) *Theory*. The toolkit as a whole suggests a genre of development, which itself becomes a 'design theory' of products that can be produced by it. In essence, the hypothesis proposed by the toolkit is that the building blocks of the toolkit suffice to create effective applications within that design genre.
(e) *Automation and maturity*. As the toolkit becomes known and regularly used, it becomes an embodiment of an accepted design theory. The resulting class of applications are somewhat predictable and used without question. We see this now with the current genre of GUI toolkits.

Unfortunately, the current state of groupware is still rather sad. Outside of research institutes, there are still no good commercial tools for developing groupware. Within research institutes, toolkits are excellent at aiding replication and codifying ideas as empirical constructs, but few have pushed matters to the 'T' stage of BRETAM. This is mostly because they are not well disseminated beyond a narrow community. Our own experiences reflect this, and are but an indication of what could be rather than what is. Still, we can learn from these experiences by asking 'How can toolkits be made and disseminated to promote creativity?' This is answered below.

6.2 Designing and disseminating toolkits for creativity

While this paper has used groupware as a case study, the arguments linking toolkits to creativity is applicable to any innovative area within interface design. For example, the lack of commercial tools means that only experts work within gesture-based interfaces, attentive interfaces, ubiquitous computing, and even information visualization. It is somewhat sad that interface builders and toolkits supplied by most commercial development environments offer little more than those offered in the mid-1980s: buttons, listboxes, scrollbars, dialog boxes, and so on.

As researchers, we promote creativity in these innovate areas through several means, listed below and contrasted to our own efforts.

1. *Take the lessons learnt from 'one-off' system design and package them as reusable components within a toolkit*. Too often people write software that illustrates great ideas, but that does not give others the means to replicate them. If initial software design is done with re-use in mind, then it becomes reasonable and natural to package the good ideas as a toolkit. This is what we did with all our toolkits. Because we had tools, others started using them and in turn their feedback helped the toolkit evolve. Having any toolkit, even if it is perhaps overly simplistic or imperfect, is far better than nothing.
2. *Create toolkits, components, and open software with a clean API that package our good ideas in a way that others can use them*. This software should encapsulate good ideas in an easy to understand abstraction, should make simple things simple to do, and should make hard things possible to implement. This API is not just good computer science, but it creates the language that people will use to think about design. Our own toolkits did this: we found that people were designing and conversing in terms of the

toolkit language, which gave them creative power. With Groupkit, they talked about shared distributed data and groupware widgets. With SDG Toolkit, they talked about manipulating input from multiple mice and keyboards. With Phidgets, they talked about how they could assemble servo motors, sensors, RFID tag readers and other hardware primitives into design products.

3. *Make them easy to learn*. Learning a new toolkit should leverage what people already know. Tools should be usable from popular programming languages and environments. People should not have to learn a new language just to use the toolkit. For example, Java and/or the major languages included in Visual Studio are reasonable target platforms. Similarly, concepts should be presented in paradigms already familiar to those programs. Examples include encapsulating behaviors as widgets, or as objects with properties and event callbacks. Our own toolkits were easy to learn by our community because we intentionally embedded them within well-known languages and programming paradigms. Because we were in a university, our language choices were dictated by what our students knew, and by what we could teach in a lecture or two. We implemented Groupkit, in Tcl/Tk because Tcl/Tk was easy to teach and had a powerful graphical user interface. As a result, Groupkit appealed to both our students and to the larger Tcl/Tk community. Our later systems, developed in Visual Studio languages, had an even broader appeal to students because they were all familiar with Microsoft languages and its development environments.
4. *Disseminate these tools within our community.* This means not only making software available, but making sure they are very easy to download, that they are well-documented (e.g., API documentation and getting started tutorials), and that they include many examples to help one quickly get going. We found that people needed to be ‘convinced’ that the toolkit was of value. By giving them the opportunity to quickly install the toolkit, and write their own very simple example via the tutorial, the personal cost of testing the toolkit for suitability was low. This meant that our potential audience had the opportunity to convince themselves of the toolkit’s value. By including reasonable documentation of the API, they could then quickly create their own examples.
5. *Recognize toolkit creation as an academic contribution*. Currently, toolkit development is rarely rewarded in the major interface conferences, for toolkits are typically perceived as software that just package already known ideas. As a research community, we have to recognize that a toolkit is a significant software engineering contribution in its own right. Toolkits deserve publication, which in turn would encourage further research and development in toolkit design. While we have successfully published many of our own toolkits, we recognize it is an uphill battle. We have spoken to many talented and inspired toolkit developers, and most are quite frustrated with the community’s view of toolkits as an engineering project (and thus not worthy of publication) instead of a first class academic contribution.
6. *Encourage the inclusion of these tools within mainstream development tools*. This is probably the hardest to achieve for it is largely a political and organizational issue. Still, many researchers are in a position of influence with industry, and should at the very least let industry people know about the existence of these tools. Similarly, students may eventually work for companies that develop these tools, or they may be involved in relevant open-source projects. They can serve as ambassadors.

Fulfilling the above steps is not easy. In my own laboratory, we have created a culture that advocates steps 1 through 4. As a consequence, toolkit creation and sharing is a frequent activity. For step 5, we have largely (but not completely) circumvented the biases associated with publishing toolkit papers mentioned by emphasizing their research contribution as well as their utility. However, we have not been as successful in step 6. Our software is freely available, is frequently downloaded by others, and interoperates well with standard development environments. Yet we have not been able to get them included in mainstream product releases. The only exception is our Phidget work, where the hardware and software are available through Phidgets, Inc., a spin-off company (www.phidgets.com). While all this takes extra work, it pays off immensely when programmers become creative in their interface designs.

Acknowledgements I am thankful for the many wonderfully talented students who produced the many interface ideas illustrated in this paper, as well as the many other ideas that I had to leave out due to lack of room. My graduate students and colleagues were instrumental in creating a culture promoting toolkit design and use: Mark Roseman, Carl Gutwin, Michael Boyle, Chester Fitchett and several others were key players. Many organizations have funded aspects of this work over the years in different ways: Microsoft Research, Smart Technologies, MERL, Intel, and of course the National Science and Engineering Research Council of Canada (NSERC).

References

1. Bederson B, Hourcade J (1999) Architecture and implementation of a Java package for Multiple Input Devices (MID). In: HCIL Technical Report no. 9908
2. Bier B, Freeman S (1991) MMM: a user interface architecture for shared editors on a single screen. In: Proc ACM UIST, pp 79–86
3. Boyle M, Greenberg S (2005) Rapidly prototyping multimedia groupware. In: Proc 11th Int'l Conference on Distributed Multimedia Systems (DMS'05), Knowledge Systems Institute, IL, USA
4. Cox D, Greenberg S (2000) Supporting collaborative interpretation in distributed groupware. In: Proc ACM CSCW, pp 289–298
5. De Bono E (1973) Lateral thinking: creativity step by step. Harper Colophon, New York, NY
6. Diaz-Marino RA, Tse E, Greenberg S (2003) Programming for multiple touches and multiple users: a toolkit for the DiamondTouch hardware. In: Companion Proc ACM UIST
7. Engelbart D, English W (1968) Research center for augmenting human intellect. In: Proceedings fall joint computing conference. AFIPS, Montvale, NJ, pp 395–410
8. Foster G (1986) Collaborative systems and multi-user interfaces. PhD thesis (UCB/CSD 87/326). Computer Science Division (EECS), University of California, Berkeley, USA
9. Gaines B (1999) Modeling and forecasting the information sciences. Inf Sci 57/58:13–22
10. Greenberg S (1997) Collaborative interfaces for the web. In: Forsythe C, Grose E, Ratner J (eds) Human factors and web development, Chapter 18. LEA, Mahwah, NJ, pp 241–254
11. Greenberg S (2003) Enhancing creativity with groupware toolkits. Invited keynote talk. In: Proceedings of the CRIWG '2003 9th international workshop on groupware (Sept 28–Oct 2, Autrans, France), LNCS vol. 2806, 1–9, Springer, Berlin Heidelberg New York
12. Greenberg S (2004) Collaborative physical user interfaces. In: Okada K, Hoshi T, Inoue T (eds) Communication and collaboration support systems (advanced information technology series) information processing society of Japan
13. Greenberg S, Fitchett C (2001) Phidgets: easy development of physical interfaces through physical widgets. In: Proc ACM UIST, pp 209–218
14. Greenberg S, Gutwin C, Cockburn A (1996) Using distortion-oriented displays to support workspace awareness. In: Sasse A, Cunningham R, Winder R (eds) People and computers XI (Proc HCI'96). Springer, Berlin Heidelberg New York, pp 299–314
15. Greenberg S, Kuzuoka H (2000) Using digital but physical surrogates to mediate awareness, communication and privacy in media spaces. Personal Technologies 4(1), January, Elsevier
16. Greenberg S, Roseman M (1999) Groupware toolkits for synchronous work. In: Beaudouin-Lafon M (ed) Computer-supported cooperative work (trends in software 7). Wiley, New York, pp 135–168
17. Greenberg S, Roseman M (2003) Using a room metaphor to ease transitions in groupware. In: Ackerman

M, Pipek V, Wulf V (eds) Sharing expertise: beyond knowledge management. MIT, Cambridge, MA, pp 203–256, January
18. Greenberg S, Roseman M, Webster D, Bohnet R (1992) Human and technical factors of distributed group drawing tools. Interact Comput 4(1):364–392
19. Greenberg S, Rounding M (2001) The notification collage: posting information to public and personal displays. In: Proc ACM CHI, pp 515–521
20. Gutwin C, Greenberg S (2004) The importance of awareness for team cognition in distributed collaboration. In: Salas E, Fiore SM (eds) Team cognition: understanding the factors that drive process and performance. APA, Washington, pp 177–201
21. Gutwin C, Greenberg S, Roseman M (1996) Workspace awareness in real-time distributed groupware: framework, widgets and evaluation. In: Sasse A, Cunningham R, Winder R (eds) People and computers XI (Proc HCI'96). Springer, Berlin Heidelberg New York, pp 281–298
22. Myers B (1995) State of the art in user interface software tools. In: Baecker R, Grudin J, Buxton W, Greenberg S (eds) Readings in human computer interaction: towards the year 2000. Morgan Kaufmann, pp 323–343
23. Sarin S (1984) Interactive on-line conferences, PhD thesis. MIT/LCS/TR330. MIT, USA
24. Shneiderman B (2000) Creating creativity: user interfaces for supporting innovation. ACM Trans Comput-Hum Interact 7(1):114–138 (March)
25. Stewart J, Bederson B, Druin A (1999) Single display groupware: a model for co-present collaboration. Proc ACM CHI, pp 286–293
26. Tang J (1991) Findings from observational studies of collaborative work. Int J Man-Mach Stud 34(2):143–160, Academic Press
27. Tang A, Boyle M, Greenberg S (2003) Display and presence disparity in mixed presence groupware. In: Proc fifth australasian user interface conference, volume 28 in the CRPIT conferences in research and practice in information technology series, (Dunedin, NZ January). Australian Computer Society Inc., pp 73–82
28. Tse E, Greenberg S (2004) Rapidly prototyping single display groupware through the SDGToolkit. In: Proc fifth australasian user interface conference, volume 28 in the CRPIT conferences in research and practice in information technology series, (Dunedin, NZ January). Australian Computer Society Inc. pp 101–110
29. Tse E, Histon J, Scott S, Greenberg S (2004) Avoiding interference: how people use spatial separation and partitioning in SDG workspaces. In: Proc ACM CSCW, pp 252–261
30. Whorf BL (1956) Language, thought and reality. In: JB Carroll (ed). MIT, Cambridge, MA
31. Zanella A, Greenberg S (2001) Reducing interference in single display groupware through transparency. In: Proc ECSCW. Kluwer, Dordrecht, The Netherlands

Saul Greenberg is a Full Professor in the Department of Computer Science at the University of Calgary. The work by Saul and his talented students typifies the cross-discipline aspects of human computer interaction. He and his crew are well known for their development of: toolkits enabling rapid prototyping of groupware and physical user interfaces; innovative and seminal system designs based on observations of social phenomenon; articulation of design-oriented social science theories, and refinement of evaluation methods. He was inducted into the ACM CHI Academy in 2005.