How a Freeform Spatial Interface Supports Simple Problem Solving Tasks

Eser Kandogan, Juho Kim[+], Thomas P. Moran, Pablo Pedemonte*

IBM Research - Almaden
650 Harry Rd.
San Jose, CA 95120
{eser,tpmoran}@us.ibm.com

[+]MIT CSAIL
32 Vassar St.
Cambridge, MA 02139
juhokim@mit.edu

*IBM Argentina
Ing. Butty 275 – C1001 AFA
Buenos Aires, Argentina
ppedemon@ar.ibm.com

ABSTRACT
We developed DataBoard, a freeform spatial interface, to support users in simple problem solving tasks. To develop a deeper understanding of the role of space and the tradeoffs between freeform and structured interaction styles in problem solving tasks, we conducted a controlled user study comparing the DataBoard with a spreadsheet and analyzed video data in detail. Beyond improvements in task performance and memory recall, our observations reveal that freeform interfaces can support users in a variety of ways: representing problems flexibly, developing strategies, executing strategies incrementally, tracking problem state easily, reducing mental computation, and verifying solutions perceptually. The spreadsheet also had advantages, and we discuss the tradeoffs.

Author Keywords
Spatial Interfaces, Freeform Interaction, Problem Solving, Spreadsheets.

ACM Classification Keywords
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

General Terms
Human Factors

INTRODUCTION
Everyday we need to solve simple problems and make practical decisions at home and at work, from making the wedding invitation list, to arranging schedule for a customer visit, and distributing work items in a group. Often such decisions involve a number of items to be organized or categorized to further the work and deal with constraints, for example, assigning software defects to developers in a balanced way considering the priority and difficulty of the defects, among other factors.

Spreadsheets are often used for such problems to enable people to lay out information in a table to help them make decisions (often collaboratively). Using a spatial representation, such as a table, engages people's natural spatial and perceptual skills to simplify choice, perception, and mental computation [12]. Spreadsheets allow users to arrange data in rows and columns, sort, and perform calculations. Pen and paper or sticky notes on whiteboards are also commonly used to solve small problems on the spot [4]. Freeform interfaces mimic paper or board in the sense that they aim to provide a fluid natural interaction space to arrange information flexibly without the constraint of a table, as in spreadsheets. Freeform graphical, multi-modal, and tangible interfaces have been introduced to support users in tasks that require flexibility such as note-taking, brain-storming, and collaborative design, with varying degrees of costs and benefits [1,3,7,8]. Both freeform and structured spatial interfaces have their upsides as well as downsides. The tradeoff between freeform and structure is the main theme throughout this paper.

We are developing DataBoard to occupy a potential sweet-spot in the spectrum of spatial interfaces, specifically geared towards simple problem solving tasks. Users can jot down information related to their tasks in a freeform manner anywhere in the space in a simple point-click-type style. Yet, they can also arrange information into lists, piles, and tables and flexibly move items between them. For example, when making a wedding invitation list, one might begin by jotting down names of people, quickly organize them into lists of the brides' and grooms' side, personal or work friends, and local or remote people, and reorganize them to satisfy desired constraints flexibly.

As we continued to consider different use cases and add new features to DataBoard, a fundamental question kept coming: How, if at all, does freeform interaction help users, especially in problem solving tasks? To directly investigate that question, we conducted a controlled study to compare a very basic version of DataBoard as a freeform interface

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
Copyright 2011 ACM 978-1-4503-0267-8/11/05...$5.00.

(with the ability to snap objects into lists) with a spreadsheet interface as a structured interface.

Our aim was not just to compare two different interfaces in terms of task performance, but to develop a deeper understanding of the role of space, spatial interaction, and the features of freeform and structured interfaces in problem solving tasks. We carried out a detailed analysis of video data collected during the experiment. Our results show that a freeform spatial interface enables better task performance and memory recall than a tabular interface. From our detailed observations, we argue that freeform interaction enabled participants to (1) flexibly represent attributes of the problem in the freeform space in order to better develop strategies; (2) execute their strategies incrementally, easily tracking state as they proceeded; and (3) easily verify and tweak their solutions.

In this paper we briefly describe DataBoard, present our study method and findings, describe our observations in depth, and discuss how freeform interaction supports problem solving.

RELATED WORK

Spatial Interfaces

GUIs are inherently spatial interfaces, and they vary on the spectrum of freeform vs. structured interactions. Desktop systems, for example, offer some level of freeform interaction in that users can move objects anywhere on the desktop. On the other hand, modern spreadsheets such as Microsoft Excel™ provide a tabular layout and allow users to organize, transform, and visualize data only within a table.

Some tasks, particularly those creative in nature, are potentially better suited for freeform interactions, such as brainstorming [7], mind-mapping [17], planning, design [4], note taking [8], collaborating in the workplace [18], and organizing photos [11]. Freeform interactions could foster the creative process by allowing the users to rapidly externalize thoughts without the constraints imposed on the use of space and transform them informally into useful organizations [1,2, 22]. For example, Microsoft OneNote™, a planning and note taking application, allows users to create several types of information and place them anywhere on the page. In OneNote users can also create lists or tables, and place them anywhere on the page, if they need to provide some structure to content created. Unlike DataBoard, however, desired structure needs to be predetermined, in that users start by creating a list, and adding information to it. Whereas in DataBoard users can simply start creating pieces of information and combine them into lists and tables as they work, and flexibly move them between lists. Another way to structure information is using page-level templates (as in Microsoft PowerPoint™), while still giving the user flexibility to organize information.

Space and Cognition

The distributed cognition literature emphasizes the importance of space and external representations in problem solving [23,24], wherein space can be intelligently utilized to generate and arrange external representations to aid problem solving by simplifying choice, utilizing perception, and minimizing internal computation [12]. Because most problems are difficult to solve only with internal (mental) representations, people often externalize information to distribute the cognitive overhead. In doing so, the nature of the representation used in problem solving can dramatically affect how people think about the problems and how long they take to solve them [23]. There is also a variety of work on how people code, use, and manipulate spatial information [16]. Manipulation can be epistemic, in that it does not directly improve the current state to be closer to the goal state, in contrast to pragmatic steps taken to solve a problem directly [13].

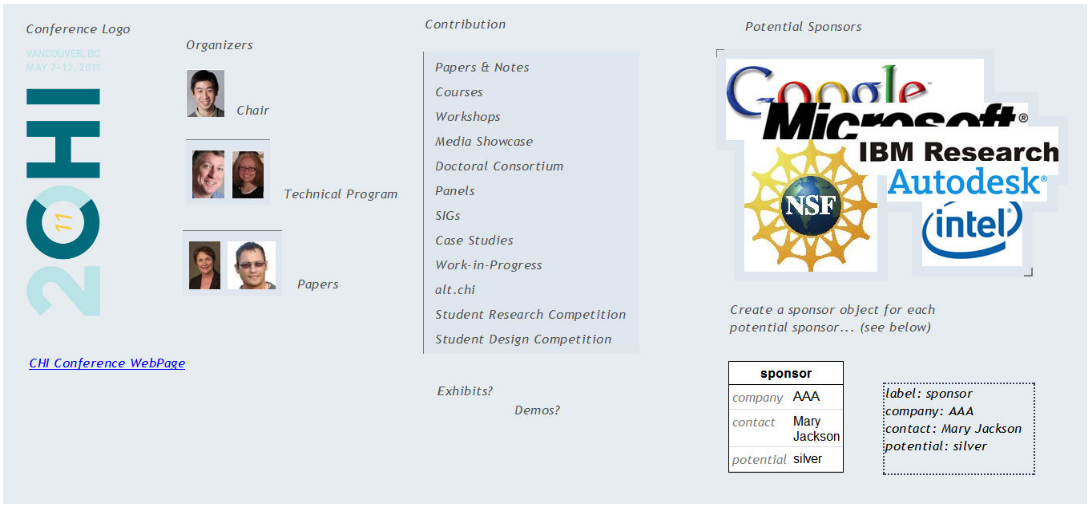
Use of space in regards to recall is also well-studied [9,10,21]. Studies of office workers to locate items in their offices identified different strategies people use for filing and retrieving information [15]. Other studies suggest that relying on location information alone for recall may be insufficient [10], especially if organization was not based on some logical structure [14]. However, user involvement in organizing the space helps recall [19].

DATABOARD

We designed DataBoard, a web-based freeform spatial interface, to help users in everyday problem solving tasks. The motivating example was management of software defects in our group. We found that the web-based defect tracking system we have been using with its rigid tabular interface was severely limiting for reviewing, categorizing, prioritizing, and distributing defects to team members during our weekly meetings.

The central idea of DataBoard is to enable users to organize and augment semantically-meaningful “domain objects” (as in [18]) such as defects in software development. Domain data is imported from a specialized application and laid out on DataBoard, where the user can interact within a freeform space, creating organizing structures (such as lists) and modifying underlying data as a result of such spatial interactions; and finally exporting the updated objects back to the application. For example, in our group meeting, we can import defects into a list on the board and then use the freeform space to “play” with them to represent their importance and urgency and discuss how they relate to each other. Then, at the end of the meeting we prioritize and assign them to the members of the team.

Using DataBoard users can create simple text, images, links, person objects by simply pointing anywhere on the layout, clicking and typing text, URLs, email addresses, which could be used to automatically create objects of different types (Figure 1). For example, an email address would be automatically mapped to a person using company address



sponsor	
company	AAA
contact	Mary Jackson
potential	silver

label: sponsor
company: AAA
contact: Mary Jackson
potential: silver

Figure 1. DataBoard supports users to create simple text, images, links, person, and user-defined objects on a two-dimensional layout flexibly through simple point-click and type. Users can move objects to snap to or overlap each other to organize them into lists and piles, respectively.

book and transformed into a Person object. Users can also create arbitrary user-defined objects with a simple syntax of name-value pairs. Finally, users can organize these objects into lists and piles, if they wish to. For the purposes of this paper, we will not go into much detail of DataBoard, but rather focus on the study and observations on the use of space and spatial interaction in problem solving.

STUDY

The promise of freeform spatial interfaces, like DataBoard's, was quite intriguing and seemed to apply to a lot of other everyday small-scale problem solving tasks. Thus, we wanted to develop a deeper understanding of how the use of space and freeform interaction contributed to our improved problem solving. To that end we designed a controlled study to see if there are performance improvements over tabular interfaces, and we especially wanted to pinpoint what aspects of the freeform spatial interaction contributed to that effect by carefully observing people behavior in problem solving tasks.

Hypotheses

In this experimental study we tested three hypotheses:

H1. Task Performance. Freeform spatial interaction will lead to shorter task performance times in problem-solving tasks over tabular spreadsheet based interfaces.

H2. Solution Quality. Freeform spatial interaction will lead to better solution quality in problem-solving tasks over tabular spreadsheet based interfaces.

H3. Memory Recall. Freeform spatial interaction will lead to better recall in problem-solving tasks over tabular spreadsheet based interfaces.

Design

A repeated measures within-subjects design was used in the experiment. The independent variables were Interface (Spreadsheet, DataBoard) and Task Type (Balancing, Scheduling). The dependent variables were Task Performance Time, Solution Quality, and Memory Recall Errors.

Each participant was brought in one at a time, and performed four tasks, two balancing and two scheduling tasks, one on DataBoard and one on Google Spreadsheet. The balancing task was to distribute a set of objects into three groups based on two attributes of the objects such that the number of objects in each group would be equal with attribute values balanced across groups. For example, given 18 software defects, the task was to assign these defects to three developers such that each would get an equal number of defects, where the distribution of defect priorities and the sum of difficulties would be balanced across developers. In the DataBoard case, participants started with an initial set of defects and they were tasked with moving these defects into three groups with a header of the developer's name (Figure 2). In the Spreadsheet, likewise participants started with an initial set of defects and they were tasked with assigning them to developers by filling in the developer column with A, B, or C (Figure 8).

The scheduling task was to order a set of objects in two lists based on dependencies between objects and another attribute of the objects. For example, given 8 defects, with different dependencies and priorities, the task was to assign these defects to two developers such that dependencies were observed and preference was given to higher priority defects. As such, unlike the balancing task, the order of the objects in the scheduling task was important. In the

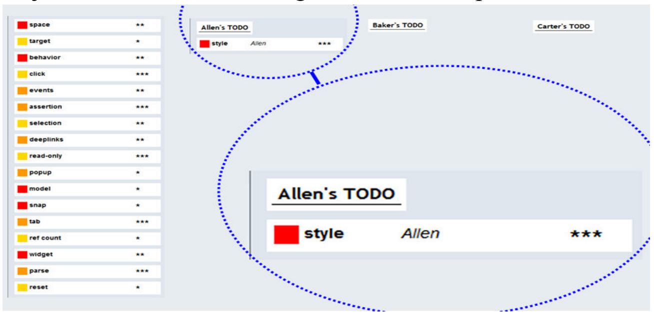


Figure 2. Balancing Task in DataBoard for Defect domain. Participants assigned defects to three developers by moving them from the original list on the left into one of the lists for a developer (enlarged in blue dotted lines)

DataBoard case, participants started with an initial set of defects and they were tasked with moving these defects into two lists with a header of the developers' name. In the Spreadsheet, likewise participants were tasked with assigning defects to developers by filling in the developer column with A or B and determining the order by typing 1, 2, 3, or 4.

To reduce an order effect in performance for the second interface, a parallel task domain was used on the second interface. For example, if the first domain was the software defects, the second interface used the class domain, where in the balancing tasks participants balanced the class load for three professors based on class difficulty and level (PhD, MS, and BS). The scheduling task in the class domain was about registering for classes based on prerequisites and class levels. Each task domain had the same number of objects (e.g. defect vs. class) and same number of groups/lists to distribute to (e.g. three professors, three developers) and tasks were isomorphic in terms of complexity.

The order of interfaces and tasks were counter-balanced. For example, participant S1 did balancing task first with Spreadsheet and then with DataBoard and scheduling task first with DataBoard and then with Spreadsheet. Participant S2 had the tasks the other way around. Participant S3 had the interfaces the other way around. The order of task domain was also counter-balanced.

Task Performance was measured with a stop-watch, started and stopped with the explicit command of the participant. The Solution Quality was assessed by experimenters and measured by the minimum number of steps to reach to a solution that satisfied all constraints of the task (e.g. balance of defect distribution based on priority and difficulty). Memory Recall tasks were given immediately after each session and inquired about the state of the participant's solution, such as the number of Ph.D. level classes for professor A. Different tasks were given at the end of each session to limit potential preparation. Memory Recall was measured by the absolute difference of the participants answer with the correct answer, ranged from 0 to 3, 0 being the perfect match. Participants who couldn't answer the memory recall task were given 3, which was the maximum possible distance from the correct answer. Memory Recall tasks were added later on as an afterthought thus only 12 participants were able to perform these tasks.

Procedure

Before starting the experiment, participants were briefed about the study and procedures to be used. They were assured that the experiment was measuring the effectiveness of the interface conditions not their own abilities to solve problems. With their permission, all experiment sessions were recorded on video in HD format to facilitate in-depth study of the strategies and actions. All sessions took place in a quiet study room.

Participants filled out a pre-questionnaire on their experience with spreadsheets, software development, and bug tracking systems. Afterwards, participants took a 3-minute spatial ability test, which contained Cards Rotation Tests [6]. Next, participants received a five-minute training on both interfaces, and reviewed features like sort and column/row move in Spreadsheet, and object selection, move, and list interaction in DataBoard. Participants then performed a training task on both interfaces to have hands-on experience using the interfaces in problem solving tasks.

Each experiment session consisted of four tasks, where participants performed two balancing and two scheduling tasks using the Spreadsheet and DataBoard interfaces. Before each session participants were reminded to think aloud, which they did to varying degrees. After reading the instructions for a particular session, participants were asked about their strategy in order to help them think in advance about the problem in order to minimize the effects of variances in strategy planning on task performance. It also helped to ensure that the problem was well-understood.

After each session participants were immediately asked to perform a quick memory recall task, followed by a subjective evaluation of the task difficulty and interface appropriateness for the current session. Upon completion of all the experiment sessions, participants filled an online post-questionnaire, where they answered questions to discuss features of Spreadsheet and DataBoard which supported or hindered their task performance, compare task difficulties in both interfaces, and whether and why they used external aids such as paper.

Participants

18 volunteers (7 female, 11 male) participated in the experiment. Participants ranged in ages from mid 20s to mid 40s, had an average skill level of 3.5 (1: novice, 5: expert) in spreadsheet use, 3.6 in software development, and 2.6 in bug tracking system use. Participants were a mix of college students and professionals with backgrounds on several fields of science, including computer science, physics, and social sciences. Participants received a lunch coupon in exchange for their help in the experiment.

Apparatus

Both conditions of the experiment were conducted on a web browser, displayed on a 20" LCD flat panel display in full-screen mode. Participants were given the option to use a mouse, a trackpoint, or a touchpad as the pointing device and pen and paper to use in their tasks.

RESULTS

Task Performance

The mean task performance times for the balancing task were 323.4 sec. (SD=206.6) with Spreadsheet and 231.5 sec (SD=143.6) with DataBoard, and for the scheduling task, 198.5 sec. (SD=81.8) with Spreadsheet and 142 sec. (SD=104.5) with DataBoard (Figure 3).

Repeated measures analysis of variance showed a main effect of Interface ($F_{Interface}(1,32) = 8.77$, $p < 0.01$) thus suggesting H1 is supported. No impact of interface order was observed on task performance ($F_{Interface*InterfaceOrder}(1,32) = 0.12$, $p > 0.5$). ANOVA on spatial ability incorporated as a covariant did not yield a significant difference ($F_{Interface*SpatialAbility}(1,31) = 0.17$, $p > 0.5$).

Solution Quality
Mean solution quality (as measured by minimum number of steps to satisfy all problem constraints from the participant's final solution) for the balancing task were 0.61 (SD=0.92) with Spreadsheet and 0.33 (SD=0.77) with DataBoard, and for the scheduling task, 0.94 (SD=1.06) with Spreadsheet and 0.72 (SD=1.07) with DataBoard. Repeated measures analysis of variance showed no main effect of Interface ($F_{Interface}(1,34) = 1.35$, $p > 0.2$) thus suggesting H2 is not supported.

Memory Recall
Mean memory recall errors for the balancing task were 0.83 (SD=0.83) with Spreadsheet and 0.42 (SD=0.9) with DataBoard, and for the scheduling task, 1.5 (SD=1.24) with Spreadsheet and 0.67 (SD=0.98) with DataBoard (Figure 3). Repeated measures analysis of variance showed a main effect of Interface ($F_{Interface}(1,22) = 4.84$, $p < 0.05$) thus suggesting H3 is supported. Note memory recall tasks were only conducted on 12 participants.

Subjective Evaluation Ratings
Subjects rated task difficulty for each interface after each session on a scale of 1 to 5 (1: very hard, 5: very easy). Mean ratings of the balancing task were 2.56 (SD=1.38) with Spreadsheet, and 3.44 (SD=0.98) with DataBoard, and of the scheduling task, 3.0 (SD=0.77) with Spreadsheet and 3.72 (SD=0.67) with DataBoard, with a main effect of the Interface (Kruskal-Wallis (KW), $p < 0.05$).

Subjects rated appropriateness of the interface, on a scale of 1 to 5, (1: not really, 5: very much). Mean ratings of the balancing task were 2.33 (SD=1.24) with Spreadsheet and 4.14 (SD=0.83) with DataBoard, and of the scheduling task, 2.28 (SD=0.82) with Spreadsheet and 3.83 (SD=0.98) with DataBoard, with a main effect of Interface (KW, $p < 0.01$).

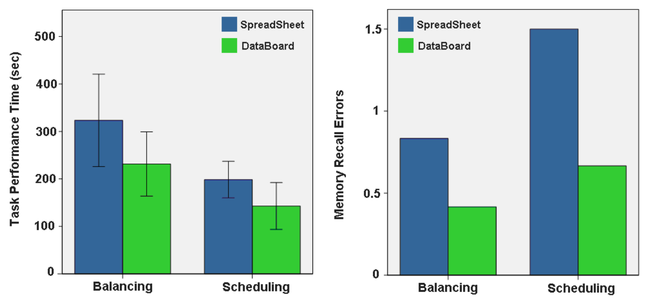


Figure 3. Task Performance Times and Memory Recall Error by Task and Interface. (Error Bars shown for +/- 2 SE)

Post Questionnaire
When asked about what helped or hindered their task performance on Spreadsheet, 10 out of 18 participants mentioned sort, particularly for categorizing information visually for easy verification. Despite availability of sorting, five participants mentioned that they had trouble mapping their tasks onto the tabular format, particularly for balancing tasks when all categories where in a single column (as opposed to three for example for balancing task, one for each developer). This was also an issue for scheduling tasks, as one participant said, not being able to map dependencies on the tabular format, "It was hard to keep in mind dependencies." One participant mentioned that even with sort he needed a way to more clearly separate rows. Four participants mentioned difficulty of keeping track of their state, and found it difficult to use expressions even as simple as sum and count, like one participant who put it as "Calculations might have helped, if I had known the right spreadsheet magic...."

Regarding DataBoard, 6 participants felt that lack of sorting hindered their task performance, while 3 mentioned ability to do calculations would have been nice for DataBoard. On the other hand, participants overwhelmingly (11) liked the visual nature of DataBoard in problem solving and suggested it helped them in their tasks by allowing them organize data in 2D, particularly to represent problem states, as one participant resembled it to a "parking lot" where you could solve parts of the problem in different places. Visual layout was also considered helpful for "visually balancing by just looking" thus reducing the need for calculation. Most participants felt that spatial interaction helped them "more effectively tune their solution". Three participants mentioned problems with spatial interaction particularly that the list interaction mechanisms could have been better designed and undo was difficult.

Participants were divided evenly regarding which task was more difficult, 8 suggested balancing task, 8 scheduling tasks, and 2 suggested about the same. On the other hand participants overwhelmingly said that Spreadsheet (12) was more difficult than DataBoard (3), while remaining said they were about equal.

Six participants used paper as an aid to help them solve problems such as calculating sums of difficulties, drawing dependency graphs, and remembering problem states at specific points, but 4 said they didn't but they should have used paper for similar reasons.

In concluding remarks, participants felt that DataBoard was really useful in their problem solving tasks, particularly for "manually solving simple problems." One participant found that "DataBoard was a lot more useful than I thought it would be," and another said "[It] opened my eyes to a new kind of dealing with data." Scalability was brought up as a specific concern with DataBoard, as one participant said: "I wonder if DataBoard would scale as much as the Spreadsheet when we get to many characteristics." The scalability of DataBoard remains to be tested, as our focus currently was simple everyday problems with a small number of objects (less than 50). Many subjects found tabular layout of spreadsheets to be limiting despite the availability of the column/row move feature in Google Spreadsheets – in fact that is why we chose Google Spreadsheet over other spreadsheet applications. This would make the interactions comparable to DataBoard both in terms of treating objects as a unit as well as in terms of interaction style. Even though participants were shown how to move rows and columns during training, many of the participants except one did not move rows and columns in the Spreadsheet condition. We believe that the visual affordances of the spreadsheet were so strong that a column/row move action was contrary to what users expected in a strict grid representation. One participant said "even though I knew I could move rows around, it never occurred to me to do so during the tasks."

VIDEO ANALYSIS

To develop a more in-depth understanding of the findings from the controlled experiment we analyzed video captured during the experiments. One of the researchers coded all the video from the experiment sessions, where events were identified and time-stamped. No additional coding by another researcher was conducted as coding was not subject to interpretation but rather focused on Next, itime-stamping events, such as typing in a number on a column in the spreadsheet or moving an object under a column in the DataBoard case.

Coding

Aside from obvious events such as, START, TALK, END, we coded action events as <OBJECT> <COLUMN> {<POSITION>}, when a participant moved (or entered text to that effect in Spreadsheet) an object to a column, optionally at a specific position, and SORT, when participant sorted (in the Spreadsheet), and NOTE to code external action such as using paper.

Additionally, we divided each session into three sequential phases: Prepare, Execute, and Verify & Recover. Prepare phase covered all actions such as sorting and moving objects in free space and lasted until the first actual assignment was made, followed by Execute which lasted until all objects were assigned, at which point we considered an initial complete solution was reached, and lastly a Verify and Recover phase in which participants tried to improve upon their initial solution. Note that this doesn't mean that participants did not perform, for example, verification in the Execute phase; it was merely our definition of gross phases of problem solving.

Observations

We identified three different strategies in our observations: 1) *Incremental* (with Trial & Error), 2) *Heuristic* (with Tweaking), and 3) *Decompose and Recompose*. Incremental strategies sought to solve the problem by reducing the scale of the problem linearly at each step. Heuristic approaches employed a general scheme to get to an initial solution very quickly and focused on fixing the initial solution thereafter. The Decompose and Recompose strategy divided the problem into smaller, more manageable parts and then merged them to achieve a solution. Below we describe our observations for both balancing and scheduling tasks and describe specific instances of each of these strategies in the experiment. The participants reported below were selected based on the differences in task performance times, in one way or another, and based on the commonality of the strategy and actions.

Balancing Task

We report our observations of S2 and S15, who performed faster with DataBoard (DB), and S4, who performed slightly faster with Spreadsheet (SS).

Decompose and Recompose + Incremental: S2. In the DB case, S2 spent considerable time planning and organizing defects into groups based on priority and difficulty (Figure 4). Then, she started distributing defects one row at a time with defects having the same difficulty and priority. She compensated differences in difficulty on subsequent rows when possible. Finally, she counted defects based on priority and difficulty across developers.

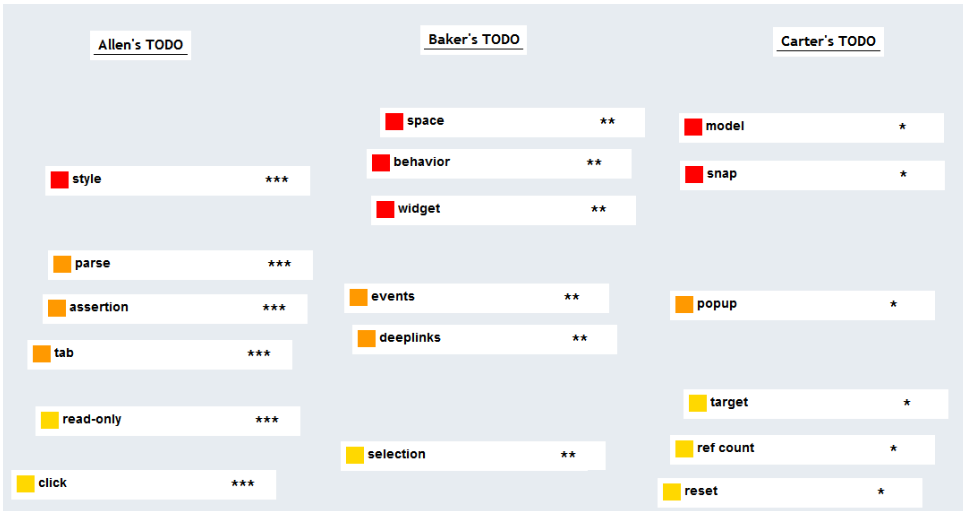
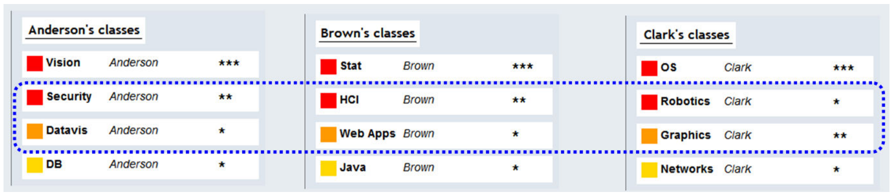


Figure 4. S2 spent considerable time decomposing the problem space by organizing defects into groups based on priority and difficulty levels.



Anderson's classes		
🟥 Vision	Anderson	***
🟥 Security	Anderson	**
🟧 Datavis	Anderson	*
🟨 DB	Anderson	*

Brown's classes		
🟥 Stat	Brown	***
🟥 HCI	Brown	**
🟧 Web Apps	Brown	*
🟨 Java	Brown	*

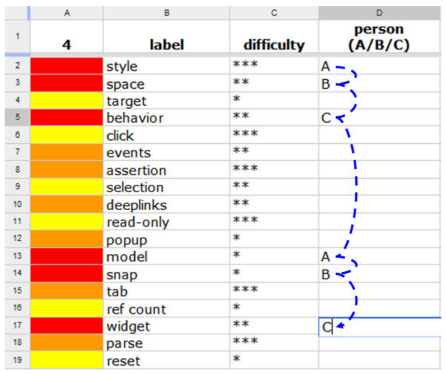
Clark's classes		
🟥 OS	Clark	***
🟥 Robotics	Clark	*
🟧 Graphics	Clark	**
🟨 Networks	Clark	*

Figure 5. S15 trying to maintain balance on subsequent rows by compensating difficulties (indicated by *s) on the second row 2, 2, 1 with difficulty levels on third row 1, 1, 2.

Next, in the SS case, S2 started by sorting first by class-level and then by difficulty, which gave a good overview of the distribution of classes. She then started distributing classes of same level and difficulty but had trouble balancing once the obvious ones (equal level and difficulty) were assigned. To help in her task, she created tables to keep track of the distribution by level and difficulty, but spent a lot of time doing this as she had trouble maintaining the table. She gave up on the table and tried sort by class level, which gave a reasonable view of the solution state. After a few more assignments she was done.

Analysis. In both cases S2 employed a combination of Decompose and Recompose and Incremental strategies. In the DB case S2 decomposed the problem by arranging defects into groups based on priority level and difficulty and thereafter executed fairly optimally in an incremental fashion. S2 was able to proceed carefully on each row, and arrived to an initial solution which was the correct solution. Tackling the problem one row at a time helped S2 execute with awareness of the problem state on each row and verify easily at the end. In the SS case, she decomposed the problem easily by leveraging sort and began an incremental strategy. Once the obvious assignments were made, S2 started to have a hard time keeping tally of assignments. Creating summary table didn't help, and she lost quite a bit time. In the end another sort came to rescue.

Incremental: S15. In the DB case, S15 did not do any preparation but proceeded in a very orderly, incremental manner by picking and choosing the right mix of classes based on class level and difficulty from the original list, and making assignments for one row at a time. On each row he either completely balanced classes or, he tried to compensate for it on the next row (Figure 5).



	A	B	C	D
1	4	label	difficulty	person (A/B/C)
2	🟥	style	***	A
3	🟥	space	**	B
4	🟨	target	*	
5	🟥	behavior	**	C
6	🟨	click	***	
7	🟧	events	**	
8	🟧	assertion	***	
9	🟨	selection	**	
10	🟧	deeplinks	**	
11	🟨	read-only	***	
12	🟧	popup	*	
13	🟥	model	*	A
14	🟥	snap	*	B
15	🟧	tab	***	
16	🟨	ref count	*	
17	🟥	widget	**	C
18	🟧	parse	***	
19	🟨	reset	*	

Figure 6. S15 trying to assign defects to developers in order (A, B, and C) by priority but also trying to balance of the difficulties on each set of assignments (A, B, C).

Next, in the SS case, S15 started again in a similar fashion assigning defects to A, B, and C in order (Figure 6). He tried to maintain balance but when only three defects remained he concluded that he can't as remaining defects had different difficulties. He cleared completely, sorted by difficulty, and then by priority. He prepared expressions to sum difficulty levels by priority. He calculated that he needed total difficulty to be 12 for each developer. Thus, he assigned 2 defects from each priority level, totaling 12 in terms of difficulty and assigned them one at a time to developers. He did not do any verification. Performance time, even only counting the time after cleanup, was still a lot longer compared to DB. (267 sec. in SS vs. 92 in DB)

Analysis. In the DB case S15 successfully completed a purely incremental strategy and found the solution easily with no verification necessary at the end. In the SS case he applied the same incremental strategy but failed as defects being dispersed in the layout made it difficult to balance. After clearing he utilized a Decompose and Recompose strategy by sorting and got a good sense of the distribution. After several calculations he transformed the problem into a computational problem and solved it as such.

Heuristic (with Tweaking): S4. In the SS case, S4 applied a heuristic approach and started by going from highest difficulty down, independent of priority levels, and assigning defects to developers in sequence, A, B, and C and got to an initial solution quickly. She tweaked her solution by first sorting on priority and counting difficulties for each person and tried to balance it by swapping defects. She did so in an optimal way.

Next, in the DB case, the participant proceeded in a similar fashion by distributing classes of the same difficulty to professors in order of A, B, and C, if possible at the same class level. Towards the end the participant performed a quick check of the distribution of class levels by professor and completed by swapping the differences. The participant did a very quick check at the end.

Analysis. In both DB and SS cases participant followed the same strategy with similar performance levels. When the strategy was well-thought to its finest detail in advance performances turned out to be similar.

Scheduling Task

We report our observations of participants S1, who performed faster with DataBoard, and S2, who performed faster with Spreadsheet.

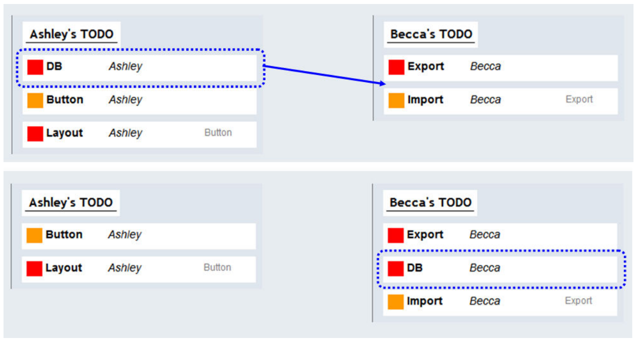


Figure 7. a) Upon seeing a priority violation across developers, i.e. Layout on Ashley vs. Import on Becca, S1 fixed it easily by moving DB to Becca, pushing Import down automatically.

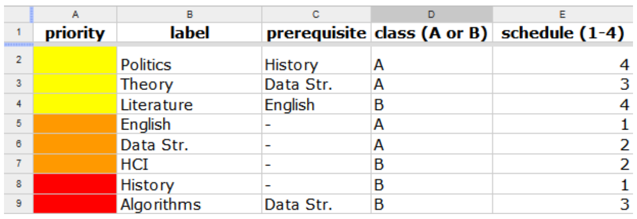
Incremental (with Trial and Error): S1. In the DB case, the participant started first by assigning the high priority defects to developers by taking dependencies into account. As she was assigning the remaining defects, S1 recognized a priority violation across developers and fixed it easily by moving a high priority defect to the other list shifting down defects accordingly (Figure 7). She then assigned the remaining defects to developers in order of priority, and quickly verified the solution.

Next, in the SS case, S1 started by sorting classes on class level and set the schedule order first (without assigning class type yet). Immediately after a few assignments she began taking prerequisites into account. Once all schedule assignments were completed, she assigned class types. Despite spending considerable time on verification (30 sec.) she didn't reach to an optimal solution, and failed on several violations of class level (Figure 8).

Analysis. The participant started in both cases by following a similar strategy, i.e. starting with high priority defects or classes and then considering dependency as needed. In both cases the participant didn't follow an optimal path to the solution yet in the DB case he was able to quickly fix it as DataBoard allowed for quick changes to ordering, facilitating flexible problem solving. In the SS case it took longer to come to an initial solution. While S1 reached to an optimal solution in the DB case, in the SS case she failed to see several violations of class level constraint because of the way classes were organized, as they were sorted by class level not by schedule order.

Decompose and Recompose: S2. In the SS case, the participant quickly developed a strategy based on specific decomposition of the defects by dependencies and priorities and executed his plan by focusing solely on ordering of classes in the schedule. The participant then assigned them to developers randomly. He did a very quick check at the end and completed the solution optimally.

Next, in the DB case, participant spent considerable time planning his strategy by organizing classes in the free space based on the prerequisites and class levels (Figure 9). His solution was just to move this onto the solution space



	A	B	C	D	E
1	priority	label	prerequisite	class (A or B)	schedule (1-4)
2	🟨	Politics	History	A	4
3	🟨	Theory	Data Str.	A	3
4	🟨	Literature	English	B	4
5	🟧	English	-	A	1
6	🟧	Data Str.	-	A	2
7	🟧	HCI	-	B	2
8	🟥	History	-	B	1
9	🟥	Algorithms	Data Str.	B	3

Figure 8. When classes were sorted by class level, S1 failed to see several violations of class level constraint.

(under class types) where he would combine several of these dependency graphs into two lists of class types. He did so in an optimal number of steps.

Analysis. In the DB case planning in free space turned out to be useful in terms of thinking about and planning for the problem but didn't carry performance benefits in the solution of the problem, particularly as it wasn't easy to move objects from the planning space to the solution space. In the SS case having a very concrete and specific plan in advanced enabled easy execution.

DISCUSSION – FREEFORM SPATIAL INTERACTION

To discuss the tradeoffs between freeform and structured spatial interaction, as they relate to the phases and strategies of problem solving we identified, we use Kirsh's categories of the use of space to organize our comparison [12]. Kirsh proposed three main categories of use of space: spatial arrangements that simplify choice, spatial arrangements that simplify perception, and spatial dynamics that simplify internal computation [12].

Simplifying Choice

Arrangements that simplify choice use spatial affordances to encode actions, which help by reducing the search space or suggesting available actions. This plays an important role in the planning phase of problem solving tasks, and as such it is crucial for strategies with heavy planning phases, such as Decompose and Recompose. In freeform spatial interfaces, we argue that the ability to map problem space features more flexibly without the constraints of the

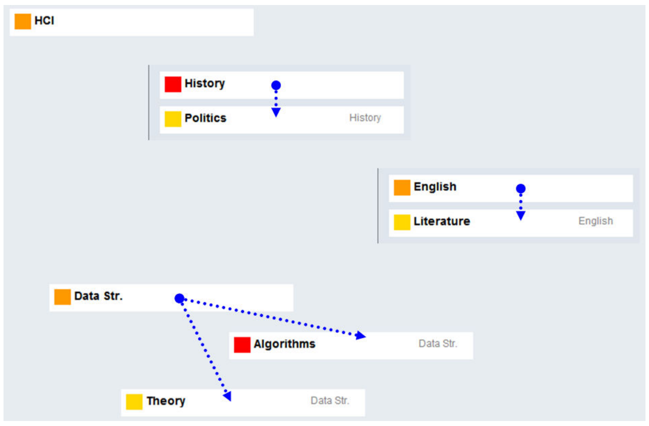


Figure 9. Participant S2 spent considerable time organizing classes in free form based on dependency and class level.

structure of tabular interface made it more applicable to a wider range of problems. In our post-interview, participants mentioned this issue particularly for scheduling task on the tabular layout, preferring the freeform nature of DataBoard. Even for the balancing task, the tabular layout's strong affordances limited participants; and they didn't consider arranging defects in flexible groups, as the DataBoard participants using the Decompose and Recompose strategies did in arranging defects by difficulty and priority. In the spreadsheet, the sorting operation was utilized heavily to compensate, but still the sorted arrangement was linear and didn't lend itself to the clear visual separations of objects that participants wanted. Also, for incremental strategies we saw that at each step of the process participants chose suitable objects from the unassigned set to make informed choices going onward.

Simplifying Perception

Spatial arrangements can also help simplify perception by making problem properties noticeable and thus making it easy to monitor the problem state. This was especially crucial for incremental approaches, where participants needed to track solution state at each step along the way. Many participants had problems in tracking state in the Spreadsheet interface, because objects were dispersed throughout the list, making it difficult to notice violations of problem constraints. This happened in situations as simple as ranking of red-orange-yellow priorities, as well as in situations with more complex dependency constraints. For heuristic approaches, which heavily rely on visual recognition of the solution state, it is critical to notice violations in the complete solution.

Simplifying Internal Computation

Problem solving is a complex cognitive task, involving representation, computation, and verification. Spatial interfaces allow users to represent the semantics of the problem spatially and visually. For example, consider the balancing task. On DataBoard assignments are represented by a list of objects under each assignee. When the user moves objects on the layout, from one list to another or onto free space, the user is intending to change assignments. On the spreadsheet, assignments are represented symbolically by names in the assignee column.

Easing internal (mental) computation involves creating visual cues to make certain properties explicit, thus bypassing the need for computing. This was particularly important for incremental approaches and to some degree in the recompose phase. Arranging defects with the same difficulty on the same row, or even compensating differences in the next row, was much easier given the horizontal arrangement of defects on DataBoard, essentially representing the summation of defect difficulties.

Spreadsheets have powerful operations, such as sorting and computation. Sorting groups assignments so that the distribution is clear and reduces internal computation. Spreadsheets also support defining computational formulas. For example, formulas to count assignments and sum up difficulties can help the user see the conformance of the problem state to the problem constraints. Yet, we saw little use of such formulas in our experiments, even though participants were fairly experienced spreadsheet users. The issue here wasn't that participants didn't know the sum and count functions; it was specifying what cells to apply these functions to. This was especially difficult, because all assignments were on a single column and it was difficult to specify cell ranges by assignee.

The DataBoard system used in the study did not provide such computational operations. However, it would be easy to add computations on spatial structures (e.g., formulas to count or add property values of items in lists) [5]. But note that spatial arrangements (e.g. horizontal alignment) reduced the need for such explicit computations.

There is an interesting tradeoff. While manually arranging items takes time, it cognitively involves the user; and thus the problem state is thoroughly understood and becomes memorable. While sorting in the spreadsheet is fast, it is perceptually jarring (as positions of objects change abruptly) and doesn't require cognitive involvement; and as such state may be less well understood and remembered.

Temporary Planning Spaces

Strategies had differing emphases in the planning, execution, and verification and recovery phases and thus made differing uses of space. In the DataBoard, a number of participants created separate planning spaces, where they represented problem space features, such as distribution of defects by priority and difficulty or dependency graph-like arrangements of defects. One of the participants called this a "parking lot," where one could temporarily park their partial solutions. In the Spreadsheet, participants had overlapping uses of space during planning and execution. A few participants created separate spaces where they did verification. They created tables to manually keep tally of assignments, but had problems updating the counts. In DataBoard, on the other hand, some participants had difficulty moving spatial arrangements off the planning spaces and merging them into the solution spaces. This was in part due to the fact that DataBoard didn't support list merge, as well as some specific list implementation issues.

CONCLUSION

Freeform satial interfaces help users to (1) flexibly map the problem onto a freeform arrangement and plan their strategies accordingly; (2) execute their strategies incrementally and easily check their actions at each step, leveraging reduced internal computation; and (3) verify and tweak their complete solutions easily, due to clear perceptual arrangements of objects in the space. These benefits, however, don't come for free, as they require involvement and effort from the users and thus may degrade overall performance.

There is a spectrum of spatial interfaces from completely freeform to completely structured. In this paper, we compared two specific instances of spatial interfaces near the ends of the spectrum. Our main goal was to further our understanding of spatial interfaces in support of problem solving tasks. We believe that there is benefit if we can provide a general way to transform structured domains onto freeform interfaces, augmented by some structuring and computational capabilities, to support more effective problem solving of common problems in these domains.

ACKNOWLEDGMENTS

We thank Steve Whittaker, Paul Maglio, and Chris Campbell for their input into the design and analysis of the study and Jeff Heer for suggesting important related work.

REFERENCES

1. Antle, A.N., Droumeva, M. and Ha, D. Thinking with hands: An embodied approach to the analysis of children's interaction with computational objects. In *Proc. CHI '09*, ACM, 2009, 3961-3966.
2. Arias, E., Eden, H., Fischer, G., Gorman, A., and Scharff, E. 2000. Transcending the individual human mind-creating shared understanding through collaborative design. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (March 2000), 84-113.
3. Agarawala, A. and Balakrishnan, R. 2006. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In *Proc*. CHI '06. ACM, 1283-1292.
4. Bailey, B.P. and J.A. Konstan. Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. In *Proc. CHI '03*, ACM, 2003, 313-320.
5. Burnett, M., Atwood, J., Djang, R., Gottfried, H., Reichwein, J., Yang., S. Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm, *Journal of Functional Programming* 11(2), March 2001, 155-206.
6. Ekstrom, R. B., French, J.W., Harman, H.H., and Derman D. (1976). *Manual for Kit of Factor-Referenced Cognitive Tests*. Princeton, NJ: Educational Testing Service.
7. Hailpern, J., Hinterbichler, E., Leppert, C., Cook, D., and Bailey, B.P. 2007. TEAM STORM: Demonstrating an interaction model for working with multiple ideas during creative group work. In *Proc. Creativity & Cognition* (C&C '07). ACM, 2007, 193-202.
8. Hinckley, K., Zhao, S., Sarin, R., Baudisch, P., Cutrell, E., Shilman, M, Tan, D. InkSeine: *In Situ* search for active note taking, In *Proc. CHI '07*, ACM, 2007, 251-260.
9. Jacob, R. J., Ishii, H., Pangaro, G., and Patten, J. 2002. A tangible interface for organizing information using a grid. In *Proc. CHI '02*, ACM, 2002, 339-346.
10. Jones, W. P. and Dumais, S. T. 1986. The spatial metaphor for user interfaces: experimental tests of reference by location versus name. *ACM Trans. Inf. Syst.* 4, 1 (Jan. 1986), 42-63.
11. Kang, H., Bederson, B.,B., Suh, B. Capture, Annotate, Browse, Find, Share: Novel Interfaces for Personal Photo Management. *Int. J. Hum. Comput. Interaction 23*(3): 315-337 (2007).
12. Kirsh, D. The intelligent use of space, *Artificial Intelligence*, v.73 n.1-2, p.31-68, Feb. 1995
13. Kirsh, D. and Maglio, P. On Distinguishing Epistemic from Pragmatic Action. *Cognitive Science 18*, (1994), 513-549.
14. Lansdale, M. W. Remembering about documents: memory for appearance, format, and location. *Ergonomics*, 34, 8 (1991), 1161-1178.
15. Malone, T.W. How do People Organize their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems 1,* 1 (January 1983), 99-112.
16. Mandler, J. M., Seefmiller, D., Day, J. On the coding of spatial information. *Memory & Cognition* 5, 1 (1997), 10-16.
17. Millen, D. R., Schriefer, A., Lehder, D. Z., and Dray, S. M. 1997. Mind maps and causal models: using graphical representations of field research data. In *CHI '97 Extended Abstracts,* ACM, 265-266.
18. Moran, T. P., van Melle, W., and Chiu, P. 1998. Spatial interpretation of domain objects integrated into a freeform electronic whiteboard. In *Proc. UIST '98*. ACM, 1998, 175-184.
19. Naveh-Benjamin, M. Coding of Spatial Location Information: An Automatic Process? *Journal of Experimental Psychology: Learning, Memory, and Cognition 13,* 4 (1987), 595-605.
20. Norman, D., & Hutchins, E. Computation via direct manipulation. Final Report to Office of Naval Research, Contract No. NOOO14-85-C-0133. University of California, San Diego, 1988.
21. Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D., van Dantzich, M. Data Mountain: Using Spatial Memory for Document Management. In *Proc. UIST '98*, ACM. 1998, 153-162.
22. Warr, A. and O'Neill, E. Tool Support for Creativity using Externalisations. In *Proc. Creativity and Cognition* (C & C '07). ACM, 2007, 127-136.
23. Zhang, J. The Nature of External Representations in Problem Solving. *Cognitive Science 21*, 2 (1997), 179-217.
24. Zhang, J. and Norman, D. A. Representations in Distributed Cognitive Tasks. *Cognitive Science 18*, 1 (1994), 87-122.