Brainstorming with Generic Tasks: An Empirical Investigation

Michael J. Mahemoff
Department of C.S. & S.E.
The University of Melbourne
Parkville, 3052, Australia
m.mahemoff@csse.unimelb.edu.au
Tel: +61 3 9344-9100
Fax: +61 3 9348-1184

Lorraine J. Johnston
School of I.T.
Swinburne University of Technology
Hawthorn, 3122, Australia
ljohnston@swin.edu.au
Tel: +61 3 9214-8742
Fax: +61 3 9214-5501

Abstract

This paper addresses the concept of generic tasks—low-level tasks which recur across different systems (e.g. remove, compare). We propose that generic tasks can facilitate rapid brainstorming of functionality during early system design, and also indicate that they could provide a common vocabulary for concepts at a higher level of abstraction than UI objects. A taxonomy of generic tasks was prepared from fourteen software projects; a pilot study was conducted to investigate how people use the generic tasks. In this study, six subjects were presented with two hypothetical systems, and initially used a standard structured technique to propose as many new tasks as possible for each. Subsequently, they were introduced to the generic task list. The results indicate that using the list helped subjects to generate new tasks which they had not previously proposed.

Keywords Generic Tasks, Design Reuse, Task Analysis, Human-Computer Interaction

1. Introduction

An important component of design is the ability to imagine new possibilities for the system under consideration. If potential features are not sufficiently explored early on, the likely consequence is a system with unsatisfied users. Because system features are decided upon so early in the software lifecycle, failing to identify appropriate features leads to a costly redesign process.

One promising way to help designers generate novel ideas is to improve opportunities to reuse tasks which have been implemented in earlier applications. Designers can gain from reasoning by analogy to similar systems, as opposed to deriving a design purely from first principles. Brooks [4] noted that it is unusual for engineering science to rely only on basic science to formulate new ideas. Pure scientific theory can provide *too much* detail, making it difficult for practitioners to draw analogies between their current situation and their past experiences.

Tasks which are typically performed by users can be placed into a taxonomy of "generic tasks", and designers of a new system can draw ideas from this taxonomy. In a survey of user interface development processes and methodologies, Hartson and Boehm-Davis [7] suggested the need for task taxonomies in HCI. According to the authors, taxonomies would help us to reuse task analyses. Taxonomies of HCI-related concepts such as tasks and user classes would also be valuable to designers. Reuse of detailed software design has also become very popular, evidenced by widespread interest in software design patterns and frameworks (e.g. [2, 8]). This heightens the need to reuse artefacts related to software design and code, such as task models; there are synergies to be gained if we can map between recurrent task models and recurrent software design constructs.

The concept of generic tasks also provides a structured way to generate novel ideas. It is based on the assumption that there is a small set of generic tasks which can form the basis for many systems. For example, many systems are equipped with a "Compare" task (e.g. comparing two graphs, comparing two word-processor documents). In the midst of a vigorous design process, designers (and users) may not even consider the possibility of allowing this functionality in their work. Once the system has been implemented, including the "Compare" task would likely require a substantial re-engineering effort. A couple of hours browsing through a list of generic tasks including the "Compare" task would reduce the likelihood of such events.

Of course, practitioners need to guard against "feature creep", i.e. using a list of generic tasks to derive obscure ideas and then implementing them without appropriate consideration of their usefulness. Indeed, the central aim of the generic task approach as a brainstorming technique is not feature generation for its own sake, but rather risk prevention; it helps to ensure designers do not overlook critical features. The proposal here is

that generic tasks be incorporated into an integrative process of the developer's choosing, which would include a selection of regular HCI techniques (user and domain analysis, participatory design, evaluation, etc.). A sensible application of generic tasks would take into account the capability of generic tasks to help explore avenues which might otherwise have been overlooked, and to do so at a very early stage.

In this paper, we describe how we obtained a list of twenty-two generic tasks from the requirements for fourteen industry-based, year-long, student projects. This process served to demonstrate that certain tasks do recur across different systems, and also laid the foundations for our pilot study. The predominately qualitative study was conducted to understand how people apply generic tasks and how they feel about the design technique. It should be noted that brainstorming is not the only application of generic tasks: the concept could also be used to form a common vocabulary. This would support interdisciplinary design and may also aid techniques for design reuse.

2. Identifying generic tasks

Some researchers have attempted to classify low-level tasks, e.g. Foley et al.'s [5] concept of "basic interaction tasks" for interactive graphics. For programs modelling two-dimensional space, they proposed four basic tasks: specifying a position, selecting an element, interacting with text, and specifying a numeric value. The taxonomy also describes three "composite interaction tasks": dialogue boxes, construction techniques (e.g. specifying a line), and dynamic manipulation (adjusting previously-constructed objects, e.g. rotating an object).

More recently, Baber [1] described several "generic actions" of human-computer interaction (HCI), on the basis of ISO/DP 7942. The purpose was to establish a framework for discussing how various input devices compare in supporting particular tasks. Under this scheme, four categories were identified (with a total of seven actions): selecting objects, dragging objects, changing the orientation of objects (e.g. rotation), and entering data.

There are also higher-level approaches to reuse, e.g. Breedvelt-Schouten et al. have demonstrated that segments of task hierarchies can be reused [3]. This work attempts to formalise the notion of recurring task hierarchies, e.g. "Define Query" consists of a number of "Enter Query parameters" followed by "Submit".

Our work extends these contributions by capturing a broad list of generic tasks. We have also performed the analysis in a systematic manner which could be repeated, for instance, by an organisation wishing to capture recurring tasks among its own projects. Fourteen software projects were analysed. The projects were performed by teams of about four people, as a full-year third or fourth-year software project, and were in the order of 10,000 lines of code. Clients were drawn from both industry and academia.

We narrowed down fifty candidate projects to the final fourteen according to two criteria:

Human-computer interaction The system had to interact directly with humans.

Complete documentation Complete software requirement specifications and user documentation were necessary. Requirements specifications had to contain usage scenarios.

There were no criteria relating to the the final product. In fact, the final product's usability was not considered because we were focusing on the *tasks* which emerged after requirements-gathering, rather than the ways in which the software supported these tasks.

Key tasks were extracted from the usage scenarios. This enabled us to quickly identify a large number of tasks from a variety of domains. However, there was a concern about the depth of this approach; it is conceivable that particular families of tasks are systematically left out of these scenarios. Furthermore, scenarios are only single episodes and can be difficult to generalise from—for example, to reason about common task sequences.

To address these concerns, seven diverse projects were selected for detailed task analysis (Table 1), although key tasks from the other seven were also retained. There were three principles in selecting the projects to undergo task analyses: (a) we required a diverse cross-section of projects; (b) relatively complex requirements were favoured; (c) successful projects were preferred. User documentation and requirements specifications were consulted to extract task models for the systems. The Jackson System Design (JSD) notation, adopted in the MUSE methodology for task analysis [9], was used to document the models. JSD was appropriate in this situation primarily because it is graphical and has a well-defined syntax. In combination, these attributes facilitate the process of analysing typical patterns.

3. The task taxonomy

After recording tasks for each project and placing them into categories, we obtained a classification reflected in Table 2. Although exploring these relationships is beyond the scope of this study, it is worthwhile acknowledging a few ways the tasks can be related to each other. One possible relationship is for a task to be a special case of another task. Another relationship is where two tasks complement each other. `Search` and `Navigate`, for example, could be used in tandem by a user trying to locate a passage of text, for example. Sometimes two tasks are necessarily connected to each other, but can be performed in either order, e.g. when the system allows either "noun-verb" or "verb-noun" actions.

Table 1. Descriptions and sample generic tasks for the projects which underwent task analyses.

Project Description	Key Tasks
Contractor selection tool: to produce a "short-list" of tenderers	Summarise, Add, Remove (tenderers); Navigate (through quality factors)
Timetabling tool: to construct tutorial timetables	Change Display Options (of timetable view); Add, Remove, Move (details); Update system state (to check against domain constraints);
3D Weather Visualiser: To explore weather prediction data	Change display options (e.g. wire-frame vs. solid), Enter data (display option parameters).
Radar Tracking System: To visualise objects being tracked by a radar	Navigate (to different map areas), Obtain guidance (about how to use the system).
Source Code Visualiser: To help programmers visualise source code	Change Display Options (e.g. direction of arrows), Add, Remove, Move (nodes representing source code modules).
Multi-lingual Botany Database: To enable queries to a plant database	Search (for matching data), Change Task Parameters (to set query language).
Online Tutorial System: To support real-time online interaction between tutors and students	Navigate (tutorial material), Enter Data (to answer questions).

The tasks were placed into five groups. Note that the term `material` is used to describe software artefacts, after Riehle and Züllighoven [11]. The groups are:

Change Material These tasks enable the user to add, remove, or change the material.

Handle Entire Material These tasks do not change a material, but operate on it as a whole. For instance, saving a file is an operation on a material which does not actually change its contents or structure.

Browse Material Browsing helps a user explore a material, and might be used for viewing only, or to facilitate the changing of material. Examples are searching and navigating.

Specify Material To help users change material, as well as perform other tasks, these tasks let them specify information to the computer, such as choosing from a list of items.

Meta-tasks Meta-tasks exist purely to moderate or expose the effects of other tasks. In isolation, they would not be very useful, but they can make tremendous contributions to overall usability when other tasks are possible. For instance, there are often functions which allow users to change the way certain tasks work, enabling a flexible workflow.

The second column of the table shows examples of the tasks extracted from the selected projects. The third column demonstrates the external validity of these tasks by showing how each is supported by a typical word-processor, MS-Word in this case. The list presents a wide range of tasks, but no attempt has been made to produce an exhaustive collection of recurring tasks. The purpose is to show that generic tasks can be systematically captured, and to investigate how such a list might be used.

4. Pilot study of generic task applicability

4.1. Aims

One possible use of the generic tasks is to provide rapid generation of ideas for new software functions. We wanted to investigate whether such a brainstorming technique was possible. Furthermore, the generalisability of tasks could be demonstrated by taking generic tasks from one set of systems, and applying them to other systems.

4.2. Method

The methodology was to provide subjects with a standard structured method for devising new tasks, and then investigate whether introducing the generic task list could help them think of additional tasks. Six subjects volunteered to participate in the pilot study. Four were postgraduate students in the Department of Computer Science and Software Engineering , and two were final-year undergraduate students in that department. All students had extensive programming experience. The subjects sat in individual sessions lasting 1.5-2 hours.

Two hypothetical systems were initially shown to subjects. One was ShopCart, a client-side application for internet shopping (independent of a web browser) and the other was Ready Reader, a hand-held reading device (Figure 1). Several sketches of screens were provided for each, along with accompanying descriptions of possible tasks at each stage. Written instructions informed the

Table 2. Generic tasks, ordered into categories. The "Example" column shows instances of the tasks captured from the projects being analysed. The "Word-Processor" column demonstrates the applicability of the generic tasks to a typical application—Microsoft Word.

Tasks	Example from Analysed Projects	Corresponding Word-Processor Task
Change Material		
Add	Add a new lesson to a multimedia tutorial.	Add text.
Remove	Delete a company from a list of tenderers.	Delete text.
Replace	Replace currently-loaded video sequence with a new sequence	Over-write a selected text block.
Move	In a timetable representation, drag a subject cell to a new timeslot.	Move a drawing object.
Transform	Re-layout a network of nodes and edges.	Underline a text block.
Handle Entire Material		
Store	Save a recently-generated animation	Cut text to clipboard.
Retrieve from Storage	Load source code which is to be visualised.	Paste from clipboard.
Duplicate	Copy a file	Copy to clipboard.
Browse Material		
Search	Search for text from an online product database.	Search for text pattern within document.
Navigate	Request information about an element in a multimedia chemistry tutorial.	Scroll to a desired text region.
Summarise	Show a summary of objects which have been shown in an animation.	Perform word count.
Compare	Compare a weather chart on two different days.	Track document changes by comparing two different versions.
Change Display Options	Select desired language of query results.	Enable text-wrapping.
Specify Material		
Enter Data	Enter data from keyboard.	Enter data from keyboard.
Select	Choose an internet relay channel.	Select a font.
Compose	Group together several graph nodes to form a single node representing the entire group.	Select several drawing objects at once.
Decompose	Ungroup a grouped node.	Select an individual object when several drawing objects have been selected.
Meta-Tasks		
Preview Task Effect	Change cursor to indicate effect of clicking within a certain area of a multimedia tutorial	View a preview of the document as it will appear after printing.
Obtain Guidance	Request context-sensitive help by clicking on a Help link	Select Help from an object's Properties menu.
Undo	Click Back button to return to previous page.	Reverse a text change.
Update System State	Click Verify button to see timetable cells which meet specified constraints.	Update an Include Text Field after a change to the file corresponding to this field.
Change Task Parameters	On a graph visualiser, modify default number of expansion levels which apply when a user adds a node's ancestors or descendants.	Change text entry mode from Insertion to Overtype.

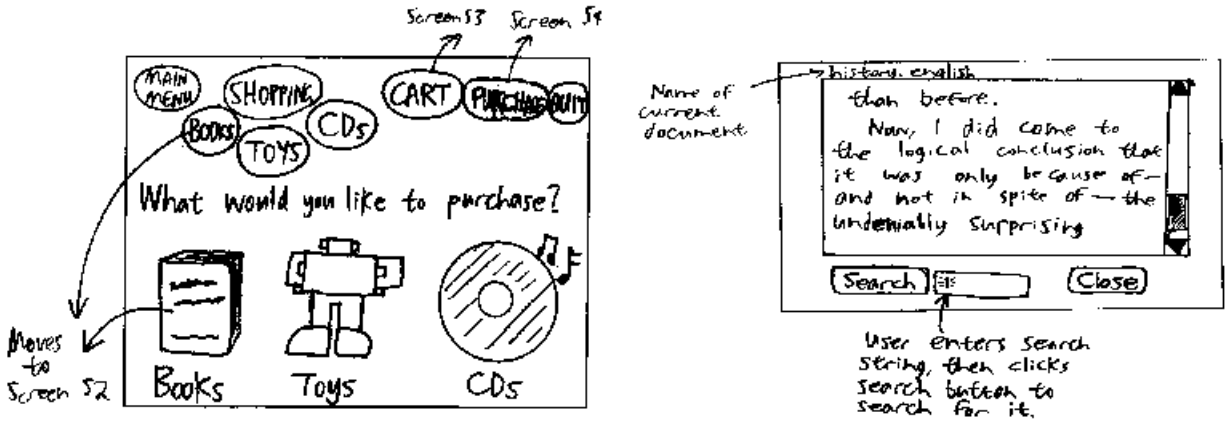

Figure 1. Sample hand-drawn sketches for ShopCart and Ready Reader applications. ShopCart screen shown is main menu, one of five screens. Ready Reader screen is document view, one of two screens.

subjects about the techniques they were being asked to employ.

In the first phase, subjects could use any means to produce the tasks. To make this stage comparable to the second phase involving generic tasks, a structured method was offered; however, it was made clear that this was only a brainstorming tool, and any other ideas were encouraged. The structured method involved asking subjects to write down users' goals in using the system, and then to identify the problems users might have in achieving these goals. From these goals and problems, subjects were asked to write additional tasks which users might be able to perform with the system. Subjects were given guidance on the structured technique, as well as a worked example, and were informed that it did not matter if they got the concepts confused (e.g. defined something as a task instead of as a goal) because the administrator would interpret their work later on. Our aim here was to encourage subjects to provide as much information as possible before being introduced to generic tasks.

When subjects felt they had exhausted their ideas for each system, they entered the second phase. The generic task list was introduced to them as an alternative method, from which more tasks could be added to their initial list. Subjects were asked to use the generic task list in a similar manner to the first approach, i.e. as a flexible brainstorming tool, so that they could feel free to suggest tasks which may not correspond directly to the generic tasks. To provide some structure, subjects were asked to step through each task, recording ideas for the system as they proceeded. Then, they were asked to write any other tasks they could think of. Finally, they were asked to step through the list and check if anything could be added. The experiment concluded with a semi-structured de-briefing discussion to clarify subjects' work and help us compare the approaches.

4.3. Results and discussion

Subjects' work was initially processed, with any identified goals or problems being converted to tasks if that was more appropriate. Tasks were eliminated if they seemed more like goals or problems, and counted once only if the person repeated them (it was quite common for people to rewrite a Phase 1 task as a generic task).

Many tasks proposed before the generic task list was introduced nevertheless mapped to a particular task (e.g. many people suggested a search function in the initial phase). Conversely, some tasks proposed after the list was introduced did not map to any particular generic task. Thus, we classified tasks according to (a) which phase they were identified in, and (b) whether or not they fitted a generic task.

4.3.1. Examples of new tasks

Before describing more detailed analyses, we now describe the tasks which were proposed more frequently, in both Phase 1 and 2. Following is a list of ShopCart tasks proposed by three or more people during Phase 1, accompanied by the matching generic task (the figures in parentheses indicate how many subjects listed each task).

Phase 1:	
Search for item (6)	Search
Remove item from cart (4)	Remove
Look at previous carts (4)	Retrieve
Phase 2:	
Specify item quantity (5)	Duplicate
Change language (3)	Change display options
Help on usage (3)	Obtain Guidance

For Ready Reader, tasks suggested by three or more people in the Ready Reader were as follows:

Phase 1:	
Show tree/list of all files at once (4)	Summarise
Search for word across documents (4)	Search
Create a new folder (3)	Add
Search for documents, e.g. by date (3)	Search
Phase 2:	
Change font (3)	Change Display Options
Group documents for delete/move (3)	Compose

It is also important to consider specific useful tasks which arose from the generic task concept. Some examples of tasks which were *only* suggested during Phase 2 of the ShopCart application are shown below. It is interesting to note that most of these tasks are supporting by existing e-commerce sites, but the generic task list seemed to trigger subjects to consider them.

- Move to "I might buy later area" (Move).
- Compare current cart with a previous cart (Compare).
- Compare among products in the same category (Compare).
- Change currency (Change Display Options).
- Select several items at once, e.g. for deleting or puchasing (Compose).
- Choose between deliver all products at once or delivering as they are available (Change Task Parameters).

Useful or novel tasks proposed only in Phase 2 for Ready Reader were:

- Change document orientation (Change Display Options)
- Show space used or free (Summarise)
- Show language of document or change language (Change Display Options)
- Merge documents (Compose)
- Summarise bookmarks (Summarise, after the subject suggested allowing bookmarks).

That subjects considered these possibilities only after being exposed to the generic tasks lends support to the notion that brainstorming can improve quality and reduce the likelihood of tasks being overlooked.

4.3.2. Comparison of methods

Overall, both phases yielded results in favour of the generic task intervention, as shown in Table 3. Mean and standard deviation were calculated, although there is no statistical test performed. It is clear, however, that the generic task list had a marked effect. All subjects were able to think of several tasks before being shown the generic task list, and introducing the list led them to add several more. This result was true for both hypothetical systems.

One interesting result is the wide individual variation which occurred for both phases. This is not entirely surprising, since people will have different abilities in solving this type of problem. Some people seem to gain a lot of support using the generic tasks (Subjects B and D), while others appear to derive less benefit (Subject A). All subjects experienced relative increases in usage of generic tasks during the second exercise, Ready Reader. De-briefing discussions suggested that this was due to a learning effect in which the unusual concept of generic tasks becomes more familiar.

4.3.3. Number of unique tasks generated

The previous table considered individual performances, so that a task was counted twice if two people suggested it. In this section, we consider the tasks generated by the group as a whole. In other words, we are looking at the number of unique tasks generated.

Table 4 shows that in both applications, introducing the generic tasks more than doubled the number of unique tasks suggested. This is despite the fact that individuals added fewer tasks on average in Phase 2 than they initially proposed in Phase 1 (Table 3). This demonstrates the generic tasks led to more variation among individuals; they led to many tasks which only one individual thought of. In contrast, there was more commonality among the initial tasks. People commonly proposed typical tasks such as Search for a book and Show cart total.

The table also shows that quite a few Phase 1 tasks mapped to a generic task. This makes sense, as designers presumably have their own models of generic tasks, some of which corresponds to those considered in this investigation. In de-briefing conversations, subjects indicated in the first phase, that they were often considering software they had seen before. Several people commented that they had features of Amazon.com in mind when designing ShopCart.

Many tasks were suggested in Phase 2 only. A concern about this methodology is that subjects may not consider items important enough to record in the first phase, but will record them in the second phase if they see a corresponding generic task. This may have happened in a couple of instances. For instance, a Help function was not recorded initially, but on seeing the generic task "Obtain Guidance", three subjects proposed it. However, this was not the case for most tasks. Two useful Ready Reader tasks suggested by three people in Phase 2 but overlooked by everyone during Phase 1 were (a) grouping documents together for deleting or moving ("Compose") and (b) changing font ("Change Display Options"). Furthermore, the "Both Phases" column suggests that at

Table 3. For each subject, number of tasks generated before generic task list shown (Phase 1) and after generic task list shown (Phase 2). Mean and standard deviation are also shown.

	ShopCart			Ready Reader		
Subject #	Phase 1	New in Phase 2	Total	Phase 1	New in Phase 2	Total
A	8	2	10	6	5	11
B	13	7	20	2	24	26
C	13	6	19	2	6	8
D	12	12	24	7	14	21
E	9	6	15	2	8	10
F	4	1	5	8	3	11
Average No. of Tasks	9.8	5.7	15.5	4.50	9.7	14.5

Table 4. Unique tasks generated by the overall group. The top row shows tasks which mapped closely to generic tasks (e.g. "Group items" relates to compose) and the second row shows tasks which does not relate strongly to any of the generic tasks. "Only in Phase 1" shows tasks generated before the generic task list was introduced (even though many suggestions mapped to generic tasks), "Only in Phase 2" shows those after the list was introduced, and "In Both Phases" shows tasks which at least one person suggested before and at least one person suggested afterwards.

	ShopCart				Ready Reader			
Task Type	Only in Phase 1	Only in Phase 2	In Both Phases	Total	Only in Phase 1	Only in Phase 2	In Both Phases	Total
Generic mapping	9	30	7	46	14	22	9	45
No generic mapping	12	1	0	13	6	2	0	8
Total	21	31	7	59	20	24	9	53

least for some tasks, there can be high assurance that the generic list was helpful. This column shows that there were some tasks which some users thought of initially, and others thought of later on.

4.3.4. De-briefing discussion

Most subjects reported that they had some difficulties using the first technique to map from goals to problems to tasks. They had difficulties understanding the distinction between goals and tasks, and also found there was a very close mapping between problems and tasks (Problem: "Does not have Search", Task: "Let user Search"). However, we asked subjects to show mappings between goals, problems, and tasks. This let us check how close the mappings were. It turned out that subjects were not overly-constrained by this technique, since they chose to add many tasks which did not correspond to particular goals or problems.

All subjects stated that they found it relatively easy to map from generic tasks to specific tasks in the system. Subject F—who did not perform many mappings—stated that he did not have problems doing this, but felt that he had proposed the major tasks already.

One surprise for us was that several subjects perceived the two phases as a cohesive methodology. We had intended the first phase as an initial benchmark for the generic tasks, and we structured our instructions to give the impression that two separate techniques were being investigated (we did not identify whether any technique was our own). In fact, one Subject (Subejct D) suggested that the goals helped to consider tasks based on users' perspectives, and the generic tasks act as a safeguard to ensure that any other useful tasks had not been omitted. Another (Subject F) confirmed this view, stating that if she had started with generic tasks she may not have covered everything. Therefore, she explained, it was good to start by considering users' goals, then moving on to tasks.

5. Conclusions

In the first half of this paper, we explained how studying tasks across a wide range of systems led us to a generic task list. This process demonstrated that certain tasks do recur across different programs and that generic tasks can be systematically documented. The initial classification we have provided may prove of some use to developers, as well as the method for capturing

generic tasks.

In the second half, we described a pilot study which demonstrated that generic tasks can have practical implications for designers. A repeated measures methodology was adopted, in which the generic task list was introduced once subjects felt their ideas had been exhausted. Since there was no control group, it is not possible to be certain that the subjects would not have arrived at the new tasks without the generic tasks being provided. However, there is significant evidence to suggest the generic tasks themselves had at least a significant influence on the ideas of subjects during the second phase. Firstly, the second phase took place immediately after the first phase, rather than a day or even an hour later. Secondly, many of the tasks proposed during the second phase were related to the generic tasks, with subjects indicating the associations. Thirdly, the de-briefing discussions indicated that subjects felt comfortable with the generic task concept and most felt they had been able to apply them directly to the design problem.

A useful property of this brainstorming technique is that it only needs one or two hours of work, even in an industrial setting (although several iterations may be required). Thus, we were able to conduct an experiment similar in duration to an industry setting. We did not have to use contrived, small-scale, systems as examples; the hypothetical ShopCart and Ready Reader applications have a similar degree of complexity to many real-life projects.

It would be useful to expand the technique into a more integrative process which takes into account users' goals and contexts. The technique might also involve participative design, since it should be very easy for users to comprehend the generic task concept. Further research might form more quantitative findings related to factors such as individual differences and application type.

The brainstorming technique can be used to design for systems which have no obvious direct analogy. Researchers have previously suggested the consideration of related systems, e.g. MUSE's extant system analysis [9], and the competitive analysis described in Nielson's Usability Engineering approach [10]. Generic tasks provide developers with a knowledge base, leading to greater confidence that existing systems have been considered.

The notion of generic tasks may be most beneficial when applied to a particular organisation's projects or confined to a specific domain. Using a method similar to the one presented here, it would be possible to "mine" tasks typically supported by a certain class of software. Databases, for example, support tasks such as "View record", "Filter records", "Search for data". Such a tool might prevent late change requests for a developer working on small projects where large-scale usability studies are prohobitive; for example, someone developing a database for video library holdings or staff management.

This paper has focused on only one application of generic tasks, i.e. as a rapid brainstorming technique. In fact, there is a broad spectrum of possible applications for generic tasks. Reusable HCI knowledge bases and design techniques, exemplified by the popular design patterns paradigm [6], can be informed by an understanding of the tasks users typically perform. The interdisciplinary HCI community can benefit from a common vocabulary; when a human factors specialist asks a software engineer to implement a "Change Display Options" task, the software engineer has a pool of knowledge to draw upon about how generic tasks may be implemented in different environments. In this way, generic tasks raise the level of abstraction from commonly-known GUI elements ("scrollbar", "button") to user tasks. Researchers can also benefit, by being able to compare interaction styles according to baseline generic tasks.

References

[1] C. Baber. *Beyond the Desktop.* Academic Press, London, 1997.

[2] K. Beck, J. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides. Industrial experience with design patterns. In *Proceedings of the 18th International Conference on Software Engineering*, pages 103–114. IEEE Computer Society, Washington, D.C., 1996.

[3] I. M. Breedvelt-Schouten, F. Paternó, and C. A. Severijns. Reusable structures in task models. In H. D. Harrison and J. C. Torres, editors, *Design, specification and verification of interactive systems '97*, pages 225–238. Springer, New York, 1997.

[4] R. Brooks. Comparative task analysis: An alternative direction for human-computer interaction science. In J. M. Carroll, editor, *Designing Interaction: Psychology at the Human-Computer Interface*, pages 50–59. Cambridge University Press, Cambridge, UK, 1991.

[5] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 2nd edition, 1990.

[6] R. N. Griffiths, L. Pemberton, and J. Borchers. Usability pattern language: Creating a community. In S. Brewster, A. Cawsey, and G. Cockton, editors, *Human-Computer Interaction: Interact '99*, page 135. BCS (for IFIP), Wiltshire, UK, 1999.

[7] H. R. Hartson and D. Boehm-Davis. User interface development processes and methodologies. *Behaviour & Information Technology*, 12(2):98–114, Mar. 1993.

[8] B. L. Kovitz. *Practical Software Requirements.* Manning, 1998.

[9] K. Y. Lim and J. Long. *The MUSE Method for Usability Engineering.* Cambridge University Press, Glasgow, 1994.

[10] J. Nielson. *Usability Engineering.* AP Professional, New York, 1993.

[11] D. Riehle and H. Züllighoven. A pattern language for tool construction and integration based on the tools and materials metaphor. In J. O. Coplien and D. C. Schmidt, editors, *Pattern Languages of Program Design*, chapter 2, pages 9–42. Addison-Wesley, Reading, MA, 1995.