Usability Issues in Software to Assist People with Brain Injuries

E. Bonneville, J.C. Muzio and M. Serra

Department of Computer Science
University of Victoria
Victoria, B.C. Canada
jmuzio@csr.uvic.ca, mserra@csr.uvic.ca

Abstract. The problems of producing a software system to assist in the rehabilitation of people who have suffered serious traumatic brain injuries are described. The challenges of ensuring a high level of usability by incorporating the best of graphic and HCI design into a well established software engineering methodology are discussed. The resulting suite of programs is now in use at a rehabilitation hospital in Victoria

1. INTRODUCTION

The Traumatic Brain Injury Program at the Gorge Road Hospital in Victoria, B.C., Canada, uses software programs to assist the cognitive rehabilitation of patients with brain injuries. Some programs are currently executed on 15 year-old computers, because there exists no version of them for more recent hardware. Even though the software is lacking many desirable features, it has proven to be very effective and is now an integral part of the Traumatic Brain Injury Program, Other North-American hospitals are facing the same dilemma, as the software was initially distributed widely, but never upgraded.
This project aims at truly benefiting persons with brain injuries. Specifically, the objectives are:

1. To fulfil the needs of the therapists and the patients by writing a suite of programs which, although they replicate the general behavior of the existing programs, also contain significant enhancements.
2. To yield a software product that can be used by other rehabilitation institutions and also by patients suffering from brain injuries working on their own.

Our system, INDIGO, is designed so that it can be repeatedly used and enjoyed by the patients and the therapists. The goal was to produce very good software, which would also provide a genuinely satisfying experience for those using it. While it has to be effective for retraining certain cognitive functions, it also has to track the progress of the patients during the rehabilitation program.

In this context, *experience* can be defined as the emotional, intellectual, and/or physical reactions the user has while using the software. Producing a “satisfying experience” might seem like a futile and esoteric goal for a software system, but it is not. In fact, it is particularly important to INDIGO. The patients who use INDIGO are engaged in a cognitive rehabilitation program not by choice, but as a consequence of an unfortunate and tragic incident. In addition, INDIGO is intended to assess and retrain cognitive functions which worked normally before the brain injury, and this is often a tremendous source of frustration for the patients. The circumstances surrounding the use of INDIGO are difficult, thus the importance of making it as pleasurable to use as possible. Moreover, INDIGO is imposed on the patients and only a satisfying experience will prompt them to accept it. The therapists are also more likely to prescribe INDIGO if both their patients and themselves enjoy it. In short, the success of INDIGO depends on its ability to provide a satisfying experience to its users.

Many factors contribute to the creation of compelling experiences in software [Winograd 96], but a primary request for the software is that it must be *usable*. Usability is the cornerstone of this project and it will be in the centre of all discussion.
Software engineers and human-computer interaction (HCI) specialists each propose methodologies for building software applications. HCI is user-centered and software engineering is system-centered. The prevailing model of software engineering is to first engineer the application, and conceive a visual representation later, if at all. Software has "traditionally been designed with a focus on the computing itself: algorithmic form, function, and implementation". However, most software engineering approaches ignore the end users. On the other hand, HCI emphasizes developing a deep understanding of user characteristics and tasks. HCI stresses the importance of testing design ideas on real users and of using formal evaluation techniques. However, most HCI methodologies neglect the role of software engineering relative to the HCI process.

Since the user experience is shaped by the software's user interface (UI), UI design deserves as much attention as engineering. Therefore, we should use both disciplines in the development of INDIGO, each in the domain it tackles effectively. Mitchell Kapor, the designer of Lotus 1-2-3, was one of the first to refer to software development as a design discipline as opposed to an engineering one. In general, design is present whenever an object is created for people to use. In software production, design is too often neglected, thus unconscious, and this is one reason for much of the poorly designed software that is in current use. Software design as defined by Kapor [Kapor 90] is a user-oriented field. This approach is similar to that proposed by HCI specialists. In fact, proponents of software design argue that the knowledge gained from the HCI field should be used in the development of software. However, software design is broader than HCI. Moreover, it doesn't ignore nor reject software engineering.

The Roman architecture critic Vitrivius declared that well-designed buildings were those which exhibited *firmness*, *commodity,* and *delight*. According to Kapor [Kapor 90], this is directly applicable to software:

- Firmness: a program should not have any bugs that inhibit its function;
- Commodity: a program should be suitable for the purposes for which it was intended;
- Delight: The experience of using the program should be a pleasurable one.

The first characteristic is the responsibility of the engineer. The latter two are the responsibility of the designer. This design approach to software seems very appropriate for developing INDIGO.

In this paper, we describe both the design process and the final software product called INDIGO. It is currently under evaluation at the hospital, but it has already been through considerable reviews and revisions from the therapists, doctors and other volunteer users.

2. BRAIN INJURIES, DESIGN PRINCIPLES AND SOFTWARE ENGINEERING

2.1. Brain injuries

The target users of INDIGO are brain injured persons. Apart from their injury, INDIGO's users are likely to have very different backgrounds, so nothing else can be assumed about them. Every year, approximately 50 000 Canadians and two million Americans suffer a traumatic brain injury and the economic costs are shocking. But the biggest issue of traumatic brain injuries is surely the lifelong and devastating consequences for the victims and their loved ones.

The brain is a delicate organ composed of millions of fine nerve fibers. Brain tissue is composed of billions of *neurons*, which cannot grow again or repair themselves. The neurons transmit nerve messages, which are in the form of tiny bursts of electricity. The brain guides virtually every aspect of a person's existence. It is notably the center of decision and execution. To handle these responsibilities, the brain must know, at each instant, what is happening inside the body and also outside the body, in the environment through the organs of the senses.

A *traumatic brain injury* [Marion 99] consists of physical damage to the brain caused by an external or recognizable force. Other events that involve a temporary lack of oxygen, such as drowning or cardiac arrest, may also impair the brain and are called *acquired brain injuries.* A brain injury can be classified according to its severity: minor, moderate or severe; moreover, it is classified according to the physical damage on the brain: it can be diffuse or focal. The general repercussion cycle (Figure 1) shows what typically happens to a person after a brain injury. The cycle starts with a physical trauma, which leads to a variety of physical, cognitive, emotional and behavioral deficits. These problems interact and can combine to create psychosocial problems, which, by feeding back into the four deficits, can create new problems.

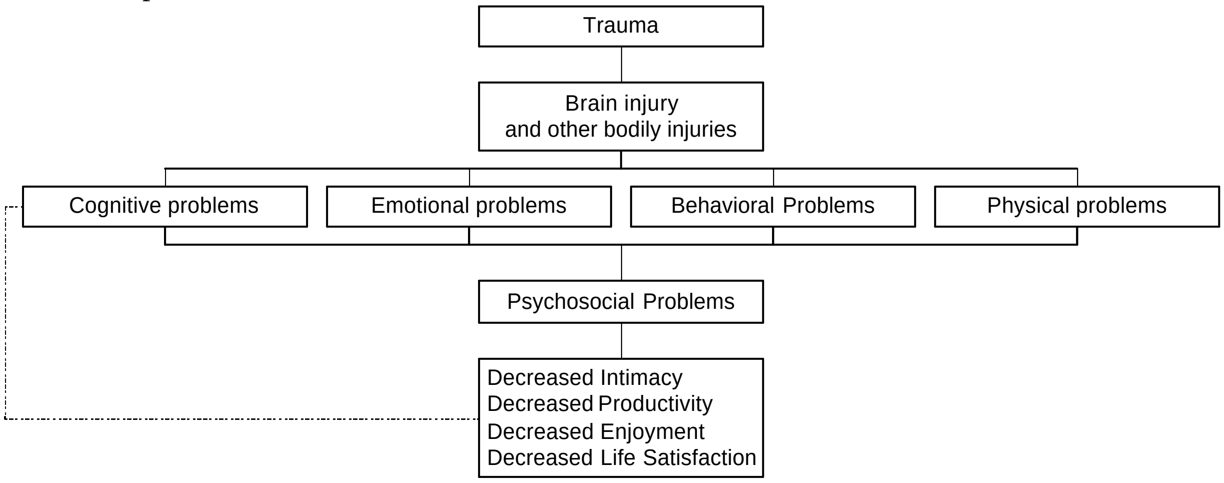

Figure 1: The Repercussion Cycle after a Brain Injury

While the peculiarities of each brain injury make it difficult to characterize the users, it is accurate to report that the victims of traumatic brain injuries typically suffer from many of the following: lessening of cognitive function and the accuracy and speed of responses, deficits in areas such as concentration, learning, memory, integrative thinking, planning and organizing, and control over emotions and behaviour. Since numerous cognitive functions are at work when interacting with software, cognitive disorders directly affect the victims ability to use software and are directly relevant to the design of INDIGO. The first three of the following domains of cognition are often impaired after a brain injury, while the latter two areas are sometimes affected:

- *Memory:* usually specific to new information while old learning is generally intact.
- *Attention and concentration:* sustain attention while carrying out a task, shift the attention from one task to another, divide the attention between two events.

- *Executive functions, reasoning and judgment:* analyzing and synthesizing information, sequencing (the ability to put things in the proper order*),* goal-directed activities.
- *Perception:* difficulties in auditory and visual functions; recognition of objects; impairment of space and distance judgment, difficulty with orientation.
- *Language abilities*: quality and the quantity of communication.

Many of these deficits are worse under conditions of stress, fatigue and anxiety. This is relevant to INDIGO because using software is, more often than not, a stressful experience.

2.2. Design Principles

The goal of the treatment process is to assist the patient in successfully adjusting and adapting to everyday living after a brain injury. In order to achieve these goals, the deterioration cycle of Figure 1 must be broken with occupational therapy, physical therapy and speech therapy to deal with the physical problems, and cognitive rehabilitation therapy and language therapy to tackle the cognitive problems. Given the goals of INDIGO, cognitive rehabilitation is obviously directly relevant. Moreover, it is important to note that the software is also intended for tracking the progress of the patient during the rehabilitation program. However, it can only play one role at a time.

Software for retraining specific cognitive functions began to appear in the 1980s. Surprisingly, there is still no empirical evidence for the use of computer software in cognitive retraining. Researchers consider them to be useful under the condition that they be carefully chosen and integrated in a rehabilitation program. Practitioners have seen results for themselves and continue to include them in their rehabilitation program.

Since we are concerned with software for human use, the knowledge gained from cognitive psychology is crucial to our success [Laurel 90]. We focus here on two of the primary cognitive processes relevant to software design, that is selective attention and learning. Selective attention is a voluntary mechanism by which we select what incoming information we need to store and process. In software, salient visual cues facilitate the selective attention mechanism. Learning, on the other hand, involves a wide variety of activities from rote memorization to complex rule learning to the acquisition of complex mental and motor skills. It is influenced by the properties of short-term memory. Analogy, structure and organization all facilitate learning. In addition, we tend to learn more easily when new information is presented incrementally.

Consequently, to facilitate the learning of software systems, it is beneficial to present information in a simple and organized manner, and in incremental units. The software designer can also build on the user's current knowledge by using analogies and possibly metaphors. Finally, the interface should be designed to demand minimal cognitive resources for handling the software system, so to leave as much resource as possible for the actual functional task that the user needs to fulfill. this requires us to design in consistency, to exploit metaphors, and to organize functionality effectively.

It is clear that design cannot be applied after the fact, when the fundamental organization of the artifact has already been determined (e.g. "design is ubiquitous", "to design is to decide", "to design is to create"). Not only must design be an integral part of the development lifecycle; it must be its starting point and its unifying factor. These ideas should be common to all software development, but they are crucial in our context, where deviations from optimal principles can cause immediate frustrations and complete failure to the brain injured user.

Because they each introduce knowledge and solutions of their own, the three visual design disciplines - graphic design, industrial design and architecture - are separate. Nevertheless, all three disciplines aim to create solutions that are both functionally effective and aesthetically pleasing. Graphic design is directly applicable to software design. It is the discipline of effective visual communication, to give form using text and pictures, to communicate facts, concepts and emotions. The output of graphic design consists of static screens in the interface. The principles and techniques for graphic design derive from five essential characteristics of effective visual displays. These characteristics are *elegance and simplicity, scale, contrast and proportion,* and finally *organization and visual structure* (see [Mullet 95]).

The various principles and techniques introduced here are contradictory at times. Graphic design is not an exact discipline, and so it requires skillful judgment. We note here a number of underlying principles which we used extensively.

- *Unity*. The element in the design must be unified to form a coherent entity.
- *Refinement*. Each element of the design as well as the design itself must be reduced through successive refinements.
- *Fitness*. The design must solve the communication problem at hand, be a fitted solution.
- *Reduction*. Reducing the design to its essence by removing non-essential elements.
- *Regularization*. Regularizing the element of the design, by using regular geometric forms, simplified contours, and muted colours wherever possible, by making similar forms identical in size, shape, color texture, lineweight, orientation, alignment or spacing, and by limiting variation in typography to a few sizes from one or two families.
- *Leverage*. Combining elements for maximum leverage. Leverage is especially important in user interface, as the available space is rather limited. However, leverage must be used sparingly as overuse will result in a complex and unusable interface.
- *Scale, contrast and proportion*. Scale contrast and proportion are about relationships among the elements in a design.
- *Clarity*. Contrast in the design must be unambiguous and clearly intentional.
- *Harmony*. There must be a pleasing interaction of the design’s parts, to form an harmonic whole.
- *Restraint*. Contrasts must be strong but few in number, and limited to one dimension.
- *Establishing perceptual layers*. Perceptual layers allow the viewer to attend selectively to one aspect (or layer) of a design. The layers are established by grouping each item of information into a few categories, by ranking each category, by determining the appropriate visual variables for the layers, by maximizing the differences between categories, and minimizing the differences within them.
- *Sharpening visual distinctions*. Sharpening ensures that the distinctions between two contrasting elements are large enough to be recognized.
- *Integrating figure and ground*. Integrating figure and ground ensures the impression of a single unified design, by fitting the design to the context in which it appears. One general way is to provide enough space around the margins of the figure, as generous margins can alone dramatically improve a screen.

- *Organization and visual structure*. Organization and visual structure provide visual pathways needed to follow the design in a systematic way. It brings together disparate elements so that they work towards a common communication goal, providing unity. It creates a framework, conveys integrity, enhances the design's readability by dividing the information into convenient subsets and brings control to both the designer and the viewer. Indeed, structure is the means by which the designer influences how the visual display is explored. By the same token, it improves the understandability of the design and facilitates navigation through the composition, all of which empower the viewer.
- *Grouping*. Related elements must be grouped into higher-order units.
- *Hierarchy*. The groupings must be ordered in a hierarchy using the visual variables.
- *Relationship*. The elements must be visually related to one another, again using the visual variables. Position is especially useful to convey relations between elements.
- *Balance*. The above principles must not disrupt the overall equilibrium of the composition. The various visual elements must be balanced on the screen.
- *Using symmetry*. Symmetry ensures balance and clear organization in almost any design.
- *Using alignment*. Alignment establishes visual relationships in a design, coordinates the visual activity of its diverse components, and so ensures that all elements work together regardless of their individual roles.
- *Shaping the display with negative space*. Empty spaces in a composition are essential because this negative space directs the viewer's attention to adjacent regions containing critical information, allows proper figure ground integration and yields a meaningful global structure.
- *Color*. Color can communicate facts, concepts, emotions and moods. In information design, color has four fundamental applications: to *label*, to *measure*, to *represent or imitate reality* and to *enliven or decorate*. The use of color in a software system directly influences its usability. Although it may appear simple at first, choosing and placing color on a visual display is very difficult. Each color in a visual display is highly sensitive to its surroundings. Therefore, colors must be chosen in context, not in isolation. Color has been found to have a considerable effect on a person's mental and physical state.
- *Typography*. Typography is a basic component of graphic design. It is concerned with the choice and the arrangement of letterforms and its goal is to enhance readability.
- *Images and signs*. Images and signs play a significant role in the user interface. They provide identification, expression and communication. Images and signs take the form of icons in software, and the user has to understand as to what they represent. Unfortunately, producing understandable icons is a real challenge. The perceived meaning of an icon depends on the context in which it appears, combined with the user's background, knowledge and interests. Hence, icons must be carefully designed according to the context in which they appear and the intended audience.

2.3. Software Engineering

All software engineering methods include three components: a notation, a process, and tools [Booch 94]. The *notation* is a "language for expressing each model". The *process* consists of various tasks that lead to the construction of these models. Finally, the *tools* are used to build the models: they facilitate this activity and enforce the models' rules. A *methodology* is "a collection of methods applied across the software development life cycle and unified by some general, philosophical approach". The prevailing software engineering methodologies divide the software-life cycle into a number of general phases which can be roughly described as: analysis, design, implementation, testing and maintenance.

We believe that the standard engineering design process is not good at connecting the actions of the software developers with the concerns of the users, so we have attributed this task to software design. The objective of this phase was to select a method for the implementation of INDIGO. In the context of software engineering, *design* refers to both the outcome and the process that provides a path from requirements to implementation - a blueprint for coding. To confuse matters even more, the *architecture* is viewed here as what we have called the internal constructs of a software system. Given that meaning, it is noteworthy that the architecture of a system is not directly relevant to its users. However, a clean architecture is necessary to produce a system which is understandable, extendible, maintainable and as much as possible, bug less. To tame this complexity, all software engineering methods use problem decomposition and we use the approach of problem decomposition in an object-oriented paradigm: each grouping in the system represents an object which collaborates with the other objects to perform all the actions necessary to solve the problem.

To justify this choice, it is useful to consider the system as a whole: INDIGO consists of ten games and exercises, which are to be accessed by a single menu. The user interface of INDIGO is graphical. The ten games form a consistent system: their options, their features and their user interfaces are to be very similar. The graphical user interface (GUI) of INDIGO justifies the use of an object-oriented design method. In practice, the development of a GUI is best performed with an object-oriented programming language. The need for consistency within INDIGO strongly suggests the use of object-oriented techniques, especially reuse mechanisms. Also, the problem space of INDIGO - games, scores, players, - is naturally viewed in terms of objects. The object model is comprised of four essential components [Booch 94], without which the model is simply not object-oriented: *abstraction, encapsulation, modularity, hierarchy.*

An *object* is characterized by its state, its behavior and its identity. A *class* is a set of objects that share a common structure and a common behavior. The identity of an object is therefore what differentiates it from all other objects, including those from the same class.
A fundamental issue in object-oriented design is to identify the objects and classes that form the design, and to organize them into hierarchies.
The following actions were required to achieve the implementation of INDIGO:

- Determine the classes, the relations among them;
- Select the applicable design patterns from the catalog;
- Produce class diagrams to illustrate the class hierarchy and the class relationships;
- Produce interaction diagrams to illustrate typical and exceptional scenarios;
- Produce the corresponding code.

As prescribed by the object-oriented process, these steps are performed incrementally and iteratively, using UML notation.

The HCI principles, the background from brain injuries studies, the philosophy and practical guidelines from design strategies have been carefully integrated to produce the INDIGO software, whose usability and robustness must be optimal to achieve its therapeutic goals (see also [Collins 95] and [Mayhew 91]).

3. INDIGO: THE SOFTWARE

3.1. Requirements Analysis

In general, Requirements Analysis results from observing existing systems, from interviewing the potential users and from studying the application domain area. In this particular case, the existing programs provided the minimal set of requirements for the new system and they were therefore examined and used extensively in order to grasp what services the system should provide and identify the aspects that could be improved. As importantly, discussions of the desired enhancements took place with the occupational therapists. The study of the application domain area was performed concurrently.

The existing programs consisted of ten exercises, of which some are used for therapy, and others are used periodically for assessing the improvements in the cognitive capacities of the patients. The programs did not have a graphical user interface. Instead, they used a question and answer dialog style, as well as primitive graphics to display various shapes and figures. The keyboard was the sole input device for most programs, though two exercises used a joystick instead. Most exercises required the user to press a key from a limited set. To facilitate the recognition of the possible keys, color stickers were placed on these keys. The patients with serious tremor used a special keyboard guard, which facilitates the use of the keyboard by restraining the number of keys that can be pushed.

Between four and six of the ten programs are used during a typical session with the programs. In these sessions, the patient sits in front of the computer while the therapist stays besides them and provides instructions. The therapist does not touch the keyboard during the exercises, but usually manipulates the computer to start the programs and to switch from one exercise to another. All programs generated various scores, but most programs did not allow the printing or saving of the user's results. When the patients worked with the programs that did not record their results, the usual procedure was for the therapists to transcribe the scores themselves during the execution of the exercise.

We briefly describe here the 10 programs to give a flavour of the new product.

1. *Vision Drill:* to assess visual-motor capacities, this is designed to help improve eye concentration on a single point. Two vertical bars move from the sides of the screen to the centre of the screen. The user must stop the bars in the exact centre by pressing the space bar.
2. *Missing Number:* to assess concentration, memory and sequencing capacities. Nine digits are displayed one after another, inside a box drawn on the screen, in the specified order (i.e. random or sequential) and at the specified speed. The user must enter the missing number.
3. *Find the Shape:* to assess visual capacities. A group of shapes appears on the screen. The target shape is alone at the top of the screen. The task is to count the shapes that match the target shape. The user is allowed to select the amount of time the shapes remain displayed.
4. *Number Search*: to assess visual capacities, and is used as part of the pre-driving assessment. The screen is divided into 4 areas with each area labelled either A, B, C or D. Then the numbers 1-28 are drawn in random order on the screen. The user must identify the area in which each number is located and may choose 2 levels of difficulty.

5. *Stack Up:* is a rendition of the Towers of Hanoi, used to improve the problem solving skills. Three tables appear with blocks stacked on table 1. The user must move the blocks to table 3 without ever placing a larger block on top of a smaller one, and may use table 3 as needed or may quit at any time.
6. *Circle Chase:* to assess visual-motor capacities. The user sees a large and a small circle and uses the joystick to keep the small circle inside the large circle. Level of difficulty and choices of running times are available.
7. *Reaction Time Test:* to assess visual-motor capacities. The screen is blank. After a random period of time, a small square is drawn in the middle of the screen. The user must erase the square as quickly as possible by pressing the joystick button. Else if the user selected the aural stimulus, after a random period of time, a sound is emitted. The user must stop the sound as quickly as possible by pressing the joystick button.
8. *Memory Challenge:* to assess memory capacities. The user sees a large rectangle in the center of the screen. A number of smaller rectangles surround the larger one. In each small one a letter appears. The user studies the letters and, when ready the letters are replaced by numbers and one of the letters appears in the center box. The user must enter the number of the box where the matching letter hides.
9. *Mirror Image:* to assess visual-motor capacities. The screen is divided in half and 16 squares labelled A-P appear on each side. When ready to start, one must press "return" and various symbols appear in the squares. The matching squares on each side contain the same symbol, but in one of the squares the symbol is inverted. One must enter the letter of the square that contains the inverted symbol.
10. *Logical Sequence:* to assess the sequencing capacities of the patients. The screen is divided in four quadrants A, B, C and D. A box appears in the middle of the screen with the work 'key' above it. An expression is displayed in the box and four related expressions appear in the four quadrants. The user must select the next logical step.
11. *Analyse Data:* to allow to view the results of exercises for a particular user.

In order to produce a software system of genuine quality, several aspects of the existing programs required considerable improvement, to take advantage of recent computer technology, to better suit the needs of the persons with brain injuries and to meet the demands of the therapists. First of all, the problems observed on the existing programs should absolutely not arise on the new system. Secondly, the existing programs underwent a major reorganization, by gathering them into one coherent system. This system now is consistent, that is, the options and features are very similar from one part to another. The software computes and saves precise users' scores for all the exercises and the stored data is readily available for consultation. The therapists have asked for meaningful scores for each exercise: the scores appear along with their unit, or with the average or the minimal score for the exercise. Also, they have requested that the order of failures and successes of the user be recorded. This feature allows the therapist to spend more time working directly with the patients instead of recording themselves the performance of the patients. In this respect, the new system takes advantage of the *complementarity* of people and machines.

Before the start of an exercise, the user is now able to choose how many times the exercise is to be executed. After each try, the system displays feedback before continuing and allows the user to quit at any point during a game. The system has a graphical user interface with an harmonious look and feel. The graphical aspect of most of the exercises thrusts a graphical user interface upon the system. By 'harmonious look and feel', we refer to the visibility, the usability and the understandability of the user interface. It is now easy to interpret and understand; it contains visible clues to its operations and provides appropriate feedback on the user's action. The user interface and the system in general should be perfectly adapted to the users, that is persons with brain injuries. For example, it now provides keyboard input in addition to mouse input to account for the patients who cannot use the latter input device. Finally, as suggested by the therapists, a few individual improvements were made on some programs.

3.2. Cost/Benefit Analysis

INDIGO offers the following benefits to the Traumatic Brain Injury Program:

- It offers the same functionality as the existing programs but with additional useful features. Consequently, the system serves the needs of the patients and the therapists better.
- The new system is easier and more enjoyable to use, which could make the patients feel more comfortable than with the existing programs.
- Once the new system is in place, the Hospital will be free to discard the old hardware.
- Because the new system is designed for hardware and platform upgrades, future transitions will be straightforward (without having to rewrite the whole system).
- The new system was developed without cost, within the University of Victoria's research facilities, ensuring a high quality standard and the use of the latest software development methods.

INDIGO offers the following costs to the Traumatic Brain Injury Program:

- Time is required from the therapists, mostly for interviews, for feedback and during the testing phase.
- The therapists have to learn how to use the new software. However, the learning curve is not too steep because it is a requirement that the new system be user friendly.
- The system is not being maintained or upgraded by the developer. However, the source code will be supplied to the Gorge Road Hospital along with the program.

The overall benefits of the new system are very clear.

3.3. Requirements Specifications

The purpose of the requirement specifications is to thoroughly describe the new system and are derived from the requirements analysis, which have been validated by the therapists through a questionnaire. A small lexicon is useful at this point.

Activity. A particular exercise, e.g. Vision Drill, Number Search, etc.

Session. When a patient uses the software and the scores are being recorded, there is an ongoing *session.*

Execution. One repetition of an activity.

Option. Parameter to choose before playing: *activity options*, which are meaningful to the scores and *play options* which represent the number of repetitions.

Execution Feedback. Result displayed to the user after every execution.

Try Feedback. Result displayed to the user after every try.

Feedback Information. Explanation displayed to the user with the feedback.

Execution Result. Execution feedback and activity options for a particular execution. Kept in memory during the whole session (scope: session). Displayed on demand during a session.

Score. Meaningful summary of execution results for each activity. Can be saved on disk (scope: as long as patient is kept in database). Displayed on demand during or after a session.

Our concept for the system is described below in point form.

- A direct manipulation interaction style is used, with the mouse as the pointing device. However, to account for patients who cannot use the mouse due to motor impairments, keyboard input is also allowed. Namely, INDIGO is fully usable either solely with the keyboard or with both the mouse and the keyboard.
- A "game" metaphor is used. This suits the application well since it consists of ten games. This metaphor is used to make INDIGO fun yet challenging, just like a game. It does not bring anxiety to the patients as regular assessment exercises can. It is the enforcement of the game metaphor that provides the delight of INDIGO.
- Windows are not used and the information is presented on the full screen. This decision is guided by the study of brain injuries. Windows can be useful to view multiple sources of information on the screen at once and to manage those views. However, this is not necessary in INDIGO because it consists of a sequence of single screens. In addition, the use of windows potentially hides some information and would complicate the interaction for persons with brain injuries.
- On-screen, contextual help is available and easily accessed at all times during the execution of the program, even during an activity. Providing genuinely useful on-line help is a challenging and multi-faceted task.
- There is no such thing as a user error in INDIGO. The software should be as helpful as possible when the user plays with the interface. To keep the user in the "right track", feedback messages are displayed when unexpected user actions occur
- All the controls are visible on the screen (e.g. there is no menu bar to hide the menu choices). This is also guided by the study of brain injuries. Hiding choices under a menu bar requires users to remember that some options are available even if they are not visible. This draws on memory. We are trying to minimize the load on cognitive functions
- Whenever an option is not available, its control simply doesn't appear on the screen. Some systems make options lighter or grey when they are not usable. This could be confusing so we simply avoid it.
- All the controls are grouped in a "control bar". This allows the user to always go to the same location to perform any action. It might take a little while for the user to have the reflex to look at the "control bar" but it should be very handy, once the habit is installed. To conform to the previous decision, options are constantly added and removed from the control bar. The context determines which controls are shown on the screen. Some controls remain even during ongoing activities.
- Various sounds, including speech output, are used for feedback and positive reinforcement after a good performance. Also, to maximize the chances that the user perceives the various messages, speech output is used concurrently to written output. The sound option can be easily turned off from any screen.
- Consistency is enforced in INDIGO. Various features are added to every exercise in order to bring consistency across the software. The options are harmonized as much as possible. For instance, the number of repetitions is an option for every exercises. Also, the try feedback, the execution feedback, the execution results and the scores are unified. In other respects, the control bar keeps the buttons in a consistent location. The look and behaviour of the buttons and other controls are coherent. A colour coding is applied to all screens.

- Feedback about the performance in the activities is very important for the user, so it is distinctly displayed. Also, an execution feedback is always displayed once the activity is performed. The user's execution result and scores are not displayed automatically but they are shown on demand.

Figure 2 illustrates our conceptual model. The selection menu is INDIGO's hub: one must pass there to access the different activities. This diagram also shows a crucial aspect of INDIGO: the user cannot get trapped in a screen. In INDIGO, there is always one button directly to the exit and another one to the selection menu (i.e. board).

An administration module is provided to the therapists. The administration module is separate from the activities: it must be executed on its own. It is used to: register and unregister patients; display and/or print the scores; set various default values for the activities' options (e.g. default values can be applied to a single patient or to a group); update a patient's record.

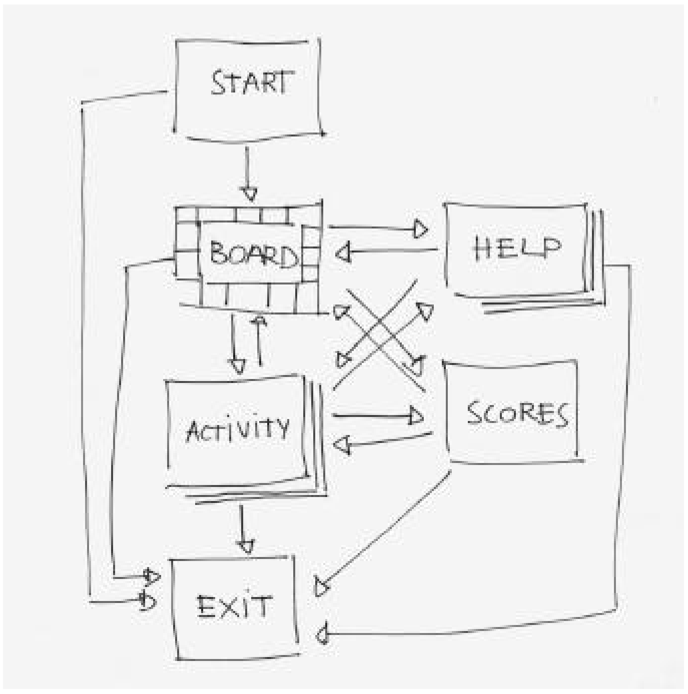

Figure 2. Conceptual Model

In INDIGO, the main focus with regard to help is to emphasize the implicit help, that is the helpfulness of the interface and the software itself. This implicit help is integrated in INDIGO by the way of its affordances, constraints, and feedback messages, to name a few. The ultimate aim of the implicit help is to provide a self explanatory software. Of course, it is unrealistic, not to mention arrogant, to claim that we can achieve such perfection. Therefore, explicit on-line help is provided to supplement the interface. One recurrent guideline dictates that on-line help should be dependent on the user's working context. In INDIGO, the on-line help is truly context sensitive: each screen in the software has a link to its own help page.

The questions that trigger the use of on-line help can be classified into five categories.
Goal-oriented questions: "What kind of things can I do with this program?".
Descriptive questions: "What is this? What does it do?". The user interface element which is most likely to require a definition is the icon button. In INDIGO, the buttons have pop-up labels which display a few words about their behaviour when the buttons have the keyboard focus.

Procedural (or task achievement) questions: “How do I do this?”. The contextual help in INDIGO explains how to achieve the tasks related to that screen.
Interpretive (diagnostic) questions: “Why did that happen?”. In INDIGO, the “error” messages might provide answers to such questions.
Navigational questions: “Where am I?”. The conceptual model of INDIGO is made explicit so that the user is well equipped to construct an appropriate mental map. This information is presented in the help page accessible through the selection menu.

Our goal with INDIGO is to make it impossible or very unlikely for the user to make mistakes. We are aiming to design a system which constantly gives transparent feedback on the user’s actions and which is clear on how to navigate and how to operate the exercises. Nonetheless, we prefer not to use the term *user error* because it implies abnormal actions from the user, and it places the blame on the user. In this spirit, it can be highly effective to display feedback messages when something unusual happens. It is noteworthy that an absolute requirement for these messages is that they be short, clear and courteous.

4. THE IMPLEMENTATION OF INDIGO

A complete description of the architecture and class structure of INDIGO is beyond the scope of this paper. The UML diagrams for the activities and the framework are available from the authors. Initially, we decided to implement INDIGO in the object-oriented programming language *Java,* which was then relatively new. We developed a few prototypes in Java, with the intention to turn them into the final system. However, *Java* proved to be too low-level, overly complex, and possibly too new and unstable for our needs. In addition, it quickly became tedious and complicated to program the GUI exactly as prescribed by the design.

Our experience with the prototypes had shown that we needed a stable development environment, intended for the production of highly customized GUI. After considering available authoring systems and development environments, we turned to the multimedia authoring system *Macromedia Director*, which is a high-level authoring system. With its film metaphor, Director allows us to produce GUIs easily and quickly. Lingo, the scripting language in Director, can be used to program the GUI and the system’s back-end. Various plug-ins to Director, known as Xtras, can be added for additional functionality, such as database connectivity, or networking capabilities. Lingo is not purely object-oriented, but it presents some object-oriented features that are very handy when developing a complex system like ours.

First of all, Lingo allows us to define *behaviours* with self-contained scripts. These behaviours have properties and methods (called *handlers* in Lingo), and are attached to graphical objects on the interface. A behaviour can be reused with various graphical objects. For instance, we have a button behaviour that is used for every single button in the system. Secondly, Lingo has object-based scripting capabilities, called *parent-child* scripting. In parent-child scripting, *parent* scripts are class definitions in that they describe how a theoretical entity should behave. On the other hand, *child* scripts create an instance of the parent scripts. Inheritance is made possible with *ancestor* scripts, which are the equivalent of a super class. Lingo allows for multiple inheritance, since parent script can have more than one ancestor. Like behaviours, child objects can be associated with physical objects, but they can also remain abstract.

This pseudo object-orientation allowed us retain the underlying architecture that we had developed with an object-oriented paradigm in mind, adapting it as needed. For example, a

focusable behaviour is attached to all GUI widgets - buttons, text fields, drop down list ... - that can receive the focus, and thus have the obligation to inform the focus manager when they did. In addition, they all have an action script attached to them, to be carried when activated. However, we had to let go the idea of a purely object oriented system in order to make efficient use of Director's capabilities. For instance, the help, and options objects are kept, but not coded as parent-child scripting. Instead, their responsibilities are coded in separate director movies, which make them self-contained, encapsulated and modular, but not purely object-oriented. With Director, we developed prototypes which were refined and iteratively became the real system. The architecture - adapted for Director - was implemented without excessive trouble.
A snapshot of the main board is shown as a simple example in Figure 3.

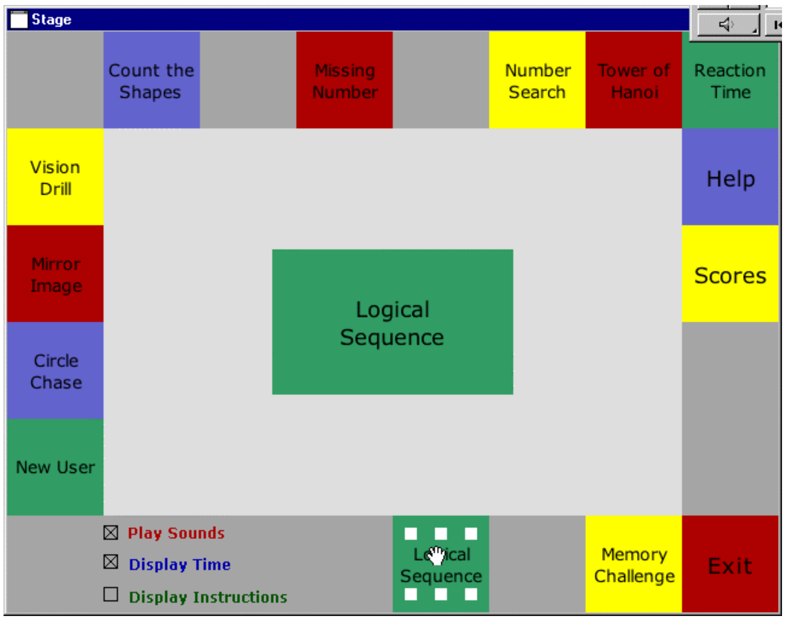

Figure 3: The primary screen for "Logical Sequence"

5. CONCLUSION

We describe the development of INDIGO to assist in the rehabilitation of those who have suffered serious traumatic brain injuries. It is shown that it is critical to incorporate the best ideas of graphic design and HCI into the development process if we are to ensure the high level of usability for the system. This is absolutely critical when many of the users will be suffering from a lack of ability to concentrate, diminished memory capacity and similar difficulties as a result of their injuries. INDIGO is now in use and evaluation at the Gorge Road Hospital in Victoria, and following the evaluation and any resulting modifications, it will be considered for distributions to hospitals across North America.

REFERENCES

[Booch 94] Grady Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, 1994 (ISBN 0805353402).

[Collins 95] D. Collins, *Designing Object-Oriented User Interfaces*, Benjamin-Cummings, 1995 (ISBN 080535350).

[Kapor 90] Mitchell Kapor, *A Software Design Manifesto*, http://www.kapor.com/homepages/mkapor/Software_Design_Manifesto.html, 1990.

[Laurel 90] Brenda Laurel, *The Art of Human-Computer Interface Design*, Addison-Wesley, 1990 (ISBN 0201517973).

[Mayhew 91] Deborah J. Mayhew, *Principles and Guidelines in Software User Interface Design*, Prentice Hall, 1991 (ISBN 0137219296).

[Mullet 95] Kevin Mullet, *Designing Visual Interfaces: Communication Oriented Techniques*, SunSoft Press,1995 (ISBN 0133033899).

[Marion 99] Donald W. Marion, *Traumatic Brain Injury,* Thieme Medical, 1999 (ISBN 0865777276).

[Winograd 96] Terry Winograd, *Bringing Design to Software*, ACM Press, Reading, Mass. And Addison-Wesley, 1996 (ISBN 0201854910).