A Usability Problem Inspection Tool: Development and Formative Evaluation

Vikrant Colaso

Thesis submitted to the faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Computer Science

Advisory Committee:
Dr. H. R. Hartson (Chair)
Dr. Scott McCrickard
Dr. Manuel Perez-Quinonez

Apr 18, 2003
Blacksburg, Virginia.

Keywords: Usability evaluation methods, UEM, UAF, UPI, UPI tool

A Usability Problem Inspection Tool: Development and Formative Evaluation

Vikrant Colaso

Abstract

Usability inspection methods of user interaction designs have gained importance as an alternative to traditional laboratory-based testing methods because of their cost-effectiveness. However, methods like the heuristic evaluation are ad-hoc, lacking a theoretical foundation. Other, more formal approaches like the cognitive walkthrough are tedious to perform and operate at a high-level, making it difficult to sub-classify problems.

This research involves the development and formative evaluation of the Usability Problem Inspection tool – a cost-effective, structured, flexible usability inspection tool that uses the User Action Framework as an underlying knowledge base. This tool offers focused inspections guided by a particular task or a combination of tasks. It is also possible to limit the scope of inspection by applying filters or abstracting lower level details.

Acknowledgements

I wish to thank my advisor Dr. Hartson, whose immense patience and guidance helped me complete this thesis. Inspite of an extremely busy schedule, he made time at very frequent intervals to accommodate my various requests. I would not have been able to complete this thesis in the short time that I did without his help.

I am also grateful to Dr. Scott McCrickard and Dr. Manuel Perez, my committee members. Their valuable inputs helped me shape this thesis.

My fellow labmates – KG (Narayanan Kodiyalam) , Varun Saini and Reenal Mahajan kept me company during those long hours in the usability research laboratory. I could always count on one of them to help me out with problems I had in the thesis. KG and Reenal also helped me during the special set-up required for the laboratory study.

I am also grateful to my parents, sister and family. I could always count on their continuous support and encouragement throughout my tenure as a graduate student.

Lastly, I would like to thank my girlfriend, Payal Mistry. Payal, you've helped me in innumerable ways during this thesis. I am also grateful for your emotional support.

Special mention needs to be made about all my friends and faculty at Virginia Tech who made my experience as a graduate student enjoyable.

Table of Contents

Table of Figures

Introduction

1.1. Problem Statement

Unlike in the past, awareness of the need for usability in user interaction designs is increasing. Numerous usability evaluation methods (UEMs) exist for determining if a particular user interface has usability problems. The process of evaluating a user interface design is called an inspection. According to Nielsen and Mack [1994], an inspection is a generic name for a set of methods based on having evaluators (usually usability experts) inspect or examine usability-related aspects of a user interface. While an inspection by definition is neutral and can be used to report both positive as well as negative usability issues for a user interface design, usability inspections are generally aimed at finding usability problems.

As an economical alternative to laboratory-based usability testing, usability inspection methods like heuristic evaluation, cognitive walkthrough, etc. have been developed. However, as argued by Andre, Hartson & Williges [2002], there seems to be a dearth of cost-effective usability evaluation tools. To this end, they have developed the User Action Framework (UAF) which is a hierarchical taxonomy of usability issues modeled somewhat on Norman's stages of interaction and proposed a suite of cost-effective tools. The Usability Problem Inspection tool is a part of this suite of tools.

Inspection techniques like the *heuristic evaluation* [Nielsen, 1992; Nielsen & Molich, 1990] are ad-hoc. Also, the heuristic evaluation method is prone to reporting false positives and requires more evaluators and experience [Jeffries, Miller, Wharton & Uyeda, 1991]. There is a need for a more formal theory-based structure.

The cognitive walkthrough approach [Lewis, Polson, Wharton & Rieman, 1990; Wharton, Bradford, Jeffries & Franzke, 1992] based on the concept of Norman's stages of interaction, is a more formal approach. However, besides being more tedious, it has a high level conceptual form and lacks ability to emphasize lower level categories (for

example, evaluation of outcomes). Thus, the cognitive walkthrough method has a tendency to often omit recurring and general problems [Jeffries, Miller, Wharton & Uyeda, 1991], because it lacks a complete hierarchical structure to guide a systematic and thorough inspection.

What is needed is a cost-effective usability evaluation method that has a more formal structure that allows for sub-classifying of problems to a desired level.

1.2. Goal

In developing the Usability Problem Inspection tool, we attempted to address the problems of ad-hoc, inflexible approaches to usability inspection without adequate usability support tools.

The goals of this thesis are to provide:

- Tool-support for usability inspection that builds upon an underlying User Action Framework.
- Structured (guided) inspection instances
- Flexibility by customization of inspection instances
- Cost-effectiveness through focused task-based inspection instances

1.3. Scope

This thesis encompasses the following:

- Design and implementation of the Usability Problem Inspection tool.
- Iterative design walkthroughs and redesign.
- One final, formal formative usability evaluation study (usability testing) to identify as many remaining usability problems as possible. Participants were selected to match the intended user class.

- Cost-importance analysis of usability problems found after which several were fixed. Solutions were also suggested for unsolved problems.

The following is beyond the scope of this thesis:

- A summative evaluation of the tool performance as a usability evaluation method (e.g., in comparison to other established methods like the heuristic evaluation and cognitive walkthrough methods).
- All the problems found were not fixed, though solutions have been suggested for all.
- Report management of the usability problems was only partially done. Reading, reviewing or modifying existing usability problem reports is not a part of this thesis.

1.4. UAF and UAF tools

1.4.1. The User Action Framework (UAF)

"The User Action Framework (UAF) is a theory-based structure for organizing usability concepts, issues, design features, usability problems and design guidelines that has evolved as a common core for a multiplicity of usability methods and tools for usability practitioners." [Hartson, Andre, Williges, van Rens, 1999].

Building upon Norman's theory of action [Norman, 1986], the UAF is a hierarchical structured knowledge base; it is technically not a taxonomy. Instead of simply identifying problems, the UAF classifies and categorizes design problems with respect to a user's interaction-based behavior.

1.4.2. UAF-based usability engineering support tools

The UAF serves as a common underlying foundation for a suite of usability engineering support tools being developed. Each of the tools plugs into the shared UAF knowledge base,

drawing upon it for content. While the content and structure of the UAF do not change from tool to tool, the *expression* of each concept reflects the specific purpose of the tool.

The UAF-based tools include the:

- UAF Explorer tool for teaching usability concepts;
- Usability Problem Diagnosis tool for extracting, analyzing, diagnosing, and reporting usability problems by problem type and by causes;
- Usability Database Tool for maintaining a life history record of each problem within a project and for supporting aggregate data analysis such as cost-importance analysis [Hix & Hartson, 1993] and usability data visualization;
- Usability Problem Inspector tool for conducting focused usability inspections, guided by the categories and sub-categories of the UAF; and
- Usability Design Guidelines tool for organizing and applying usability design guidelines in a systematic way.

1.5. Related Work

Traditional lab-based testing has reluctantly been used because of economical concerns leading to approaches such as usability inspection methods [Nielsen & Mach, 1994] that make a compromise on exhaustive complete results, opting instead for cost-effectiveness.

Nielsen's heuristic evaluation method [Nielsen, 1992] is part of an overall approach called "discount usability engineering". Nielsen describes heuristic evaluation as a method for finding usability problems in a user interface design by having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the "heuristics").

Each individual evaluator performs a heuristic evaluation by inspecting the interface alone, keeping in mind a set of usability heuristics (Nielsen, 1994). These heuristics

represent general rules that seem to describe common properties of usable interfaces. Evaluator findings are combined should be combined only after all individual evaluations have been made. Also to ensure independent unbiased evaluations from each evaluator, they should not communicate until all evaluations are complete. The heuristic evaluation produces a list of usability problems in the interface with reference to the heuristics that were violated in each case.

This method is quite popular because of its low cost and relative ease of use. Advance planning is not required for the heuristic evaluation method. Also, the method itself is intuitive, making it easy to motivate people to use it (Nielsen & Molich, 1990).

However, it has its drawbacks, as pointed out by Doubleday, Ryan, Springett and Sutcliffe (1997). While the number of usability problems identified is large, they are quite often low priority. In addition, evaluators are sometimes easily led to reporting false alarms, since the 'heuristics' are basically general usability principles (Sears, 1997).

Several evaluators are required to merge results because of low overlap in problems detected [Jeffries, Miller, Wharton & Uyeda, 1991].

Another approach, the cognitive walkthrough [Lewis, Polson, Wharton & Reiman, 1990; Wharton, Bradford, Jeffries & Franzke, 1992], focuses on ease of learning using an interaction-based approach, somewhat similar to the UAF. Unlike the heuristic evaluation method, the cognitive walkthrough method is more theory-based and focuses on the user's cognitive activities like the goals and knowledge when performing a specific task [Wharton, Bradford, Jeffries & Franzke, 1990]. The method works as follows. Experts have information about the target user group and perform a set of tasks on the system attempting to emulate typical user behavior. The method is good for finding mismatches between the users' and designers' mental models. Thus the method is good mainly for finding design problems that would interfere with learning for new users. However, this method usually requires special training. More knowledge of cognitive science terms, concepts and skills is required than most other usability evaluation methods [Lewis, Polson, Wharton & Reiman, 1990]. Other drawbacks include the tediousness of the process, the types of problems identified and longer time needed [Desurvire, 1994; Lewis et al., 1990; Rowley & Rhoades, 1992].

There is little development with regards to usability engineering support tools. A few website usability evaluation tools do exist, but are relatively domain-specific dealing with issues like accessibility and presentation, not affording a more complete inspection.

Terence Andre, in his dissertation [2000], has tested a very basic hard-wired prototype of the Usability Problem Inspection tool and claimed that usability inspections with this version of the Usability Problem Inspection tool are more effective than the heuristic evaluation inspection method and at least as effective as the cognitive walkthrough inspection method. Our work takes his basic prototype and extends it.

He conducted a comprehensive comparison study to determine if the Usability Problem Inspector (as he called it) along with the UAF as a theory based framework could be effectively used to find important usability problems in an interface design, relative to the two popular well established methods of heuristic evaluation and cognitive walkthrough. "A lab-based usability test with 20 actual users was used to establish a baseline set of 39 unique and real usability problems for the comparison of data from the Usability Problem Inspector (UPI), heuristic evaluation and cognitive walkthrough. For the comparison study, 30 professional, expert evaluators participated, each using one of the three methods. His results showed that the UPI was more effective than the heuristic evaluation in terms of thoroughness, validity and effectiveness and consistent with the cognitive walkthrough for these same measures."

With respect to the heuristic evaluation method, he found two primary advantages: a foundation in a theory –based model of user interaction and not being a mere abstraction of guidelines. By using the UAF to structure and guide the inspection, the UPI has an advantage over the heuristic evaluation technique, in which a guiding interaction-based model is absent. Also, the heuristics, being abstracted guidelines, often turn out to be too vague to be applied to a particular design situation. In the UPI, this problem is not there, as problems are embedded within the interaction cycle. With respect to the cognitive walkthrough method, his claim is that it will fall short in detailed descriptions of usability

problems, since it lacks a hierarchical structure to classify problems at a more detailed level as compared to the UPI.

Thus, his claim is that “the UPI lies between the low cost (of learning and using) and ease-of-use of the heuristic method and the completeness and task-driven structure of the cognitive walkthrough method involving extensive modeling and analysis.”

2. Design of the Usability Problem Inspection tool

The Usability Problem Inspection tool integrates with a suite of other usability tools that build upon the underlying UAF. Using the contents of the UAF as a knowledge base, the Usability Problem Inspection tool poses usability inspection questions to usability inspectors that help them identify and report problems in a user interaction design.

An *inspector* is a person trained in usability evaluation familiar with inspection methods and a typical user of the Usability Problem Inspection tool. Throughout this text, users of the tool are referred to as inspectors.

2.1. Components of the Usability Problem Inspection tool

The Usability Problem Inspection tool is made up of two major components: the Inspection Wizard and the UAF Tree Visualization. They provide the functionality of finding, classifying and reporting usability problems with a user interface design.

As shown in figure 1 on the next page, the screen space of the Usability Problem Inspection tool is split into two parts with the UAF Tree Visualization occupying about a third of the screen and the Inspection Wizard occupying the rest of the screen, to the right.

Since the UAF Tree Visualization is the secondary display, it has been given less screen space. This may cause problems because horizontal scrolling is needed for many of the UAF nodes, which have long descriptions. However, the frames are resizable allowing inspectors to override the default layout of the screen space. Also, we felt that inspectors would develop familiarity with UAF node names after a while and recognize them without the need to scroll horizontally or resize frames, hence this would not be an issue.

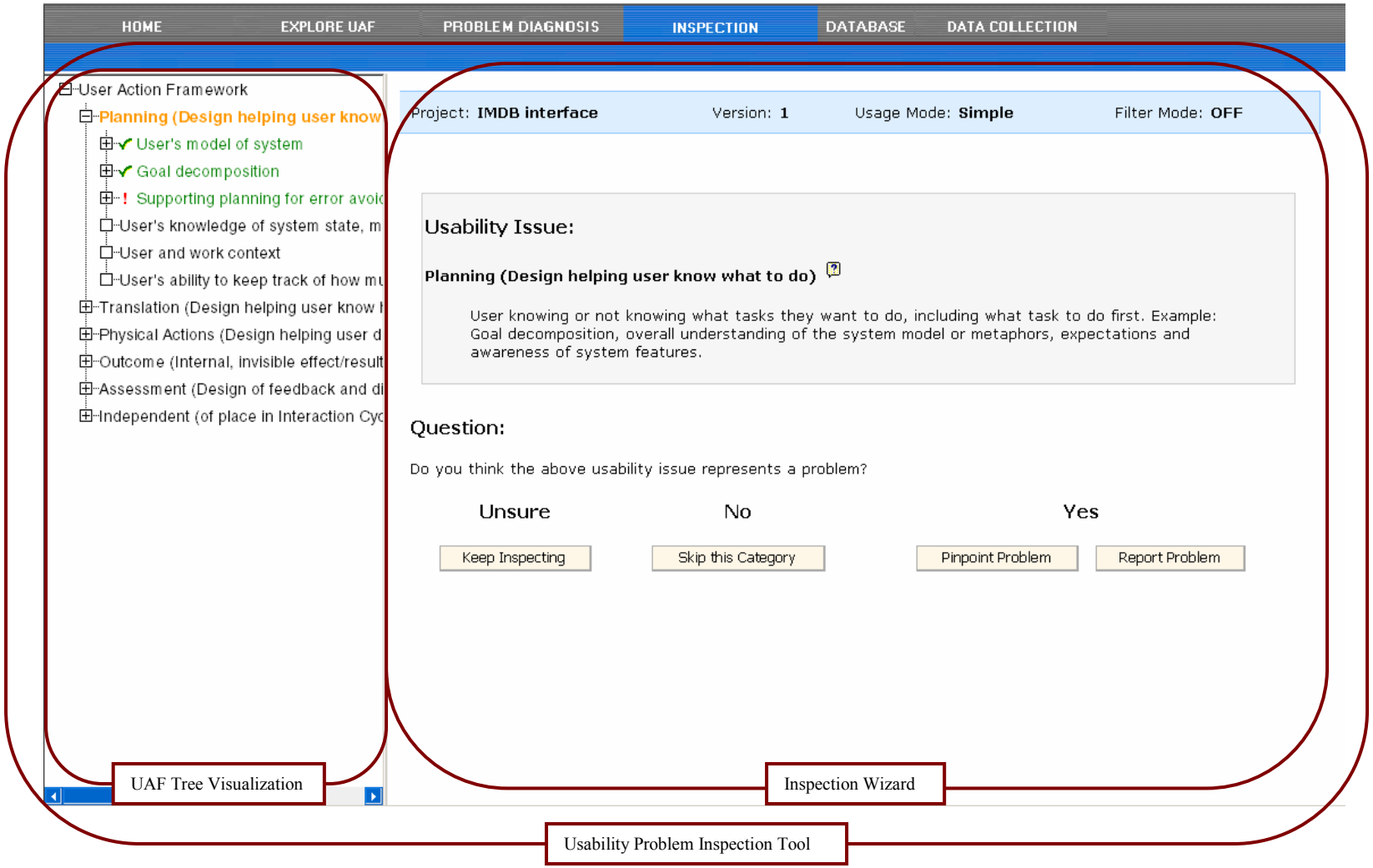


Figure 1: Components of the Usability Problem Inspection Tool.

2.1.1. The Inspection Wizard

The Inspection Wizard, in the right frame of the screen is the main component of the Usability Problem Inspection tool. It works in a typical *wizard* fashion, asking the inspector questions and determining the path of the inspection based on answers to these questions.

2.1.2. The UAF Tree Visualization

The UAF Tree Visualization occupies the left frame of the screen. It acts as an alternative method of navigation through the UAF especially if the inspector wishes to change the path determined by the Inspection Wizard and take control of the inspection.

2.2. Overview of the Usability Problem Inspection tool

The Usability Problem Inspection tool is designed to be a part of a suite of tools that use the UAF as a knowledge base. It assists inspectors in performing focused inspections for a user interaction design, using the UAF to find, classify and report problems.

Inspectors first customize an inspection instance according to their needs. An *inspection instance* refers to a particular inspection being conducted. Inspections can be performed in task-based manner, either with one task or multiple tasks as a context for inspection or in a *free exploration* manner, in which the general interface is inspected. Also, inspectors can limit the scope of an inspection in two ways, either by filtering out nodes of the UAF that are not of interest (using a filter key word) or by setting an abstraction level, thus removing lower levels of detail.

After selection of the inspection parameters, the Inspection Wizard asks the inspector questions based on the UAF. The inspector must make a decision whether a problem exists with the current UAF category or not. If the answer is affirmative, the Inspection Wizard persists within that node category, allowing the inspector to home in on the exact

problem by looking at more and more detailed sub-categories. It is, of course, possible to report the problem at any of the nodes without sub-categorizing.

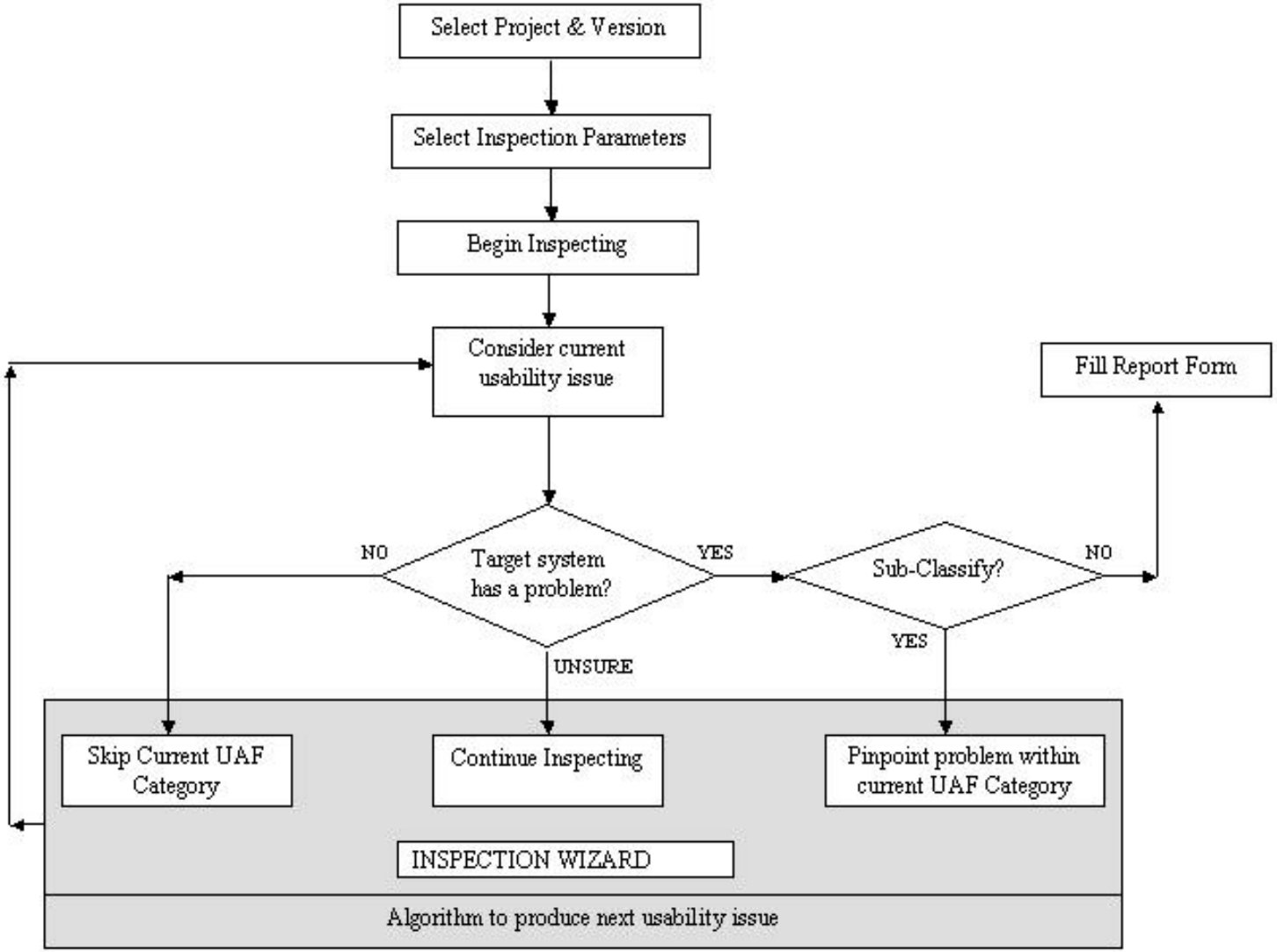
The inspector can also delay the decision if unsure whether the problem exists at the node. In this event, the Inspection Wizard will continue on to the next node that has not been considered and go on in this way until it exhaustively covers all nodes of the UAF.

Using the UAF Tree Visualization, the inspector can jump to any node when desired. This is a quick way for the inspector to navigate to a particular node in mind during the inspection.

While reporting, the inspector can describe what the problem is while also suggesting a remedy. After reporting the problem, the inspection will continue, using the Inspection Wizard until all nodes in the UAF have been considered (or marked off as non-problematic).

After reporting problems with the Usability Problem Inspection tool, it will be possible to draw upon these problems using the Usability Problem Diagnosis tool and perform further analysis.

The following diagram represents an overview of the logic flow in the Usability Problem Inspection tool.


Select Project & Version
Select Inspection Parameters
Begin Inspecting
Consider current usability issue
Fill Report Form
NO
Target system has a problem?
YES
Sub-Classify?
NO
YES
UNSURE
Skip Current UAF Category
Continue Inspecting
Pinpoint problem within current UAF Category
INSPECTION WIZARD
Algorithm to produce next usability issue

2.3. Software Architecture

The Usability Problem Inspection tool was developed using Active Server Pages (ASP) and HTML (Hyper Text Markup Language). The UAF knowledge base and other tool-related information was stored in an Microsoft Access database. For the hierarchical node-based structure of the UAF, every node is stored in the database such that its parent node is associated with it. With the help of this organization, it is possible to generate the entire tree-like structure of the UAF over which the Inspection Wizard will perform its node-traversal operations.

2.4. Setting up an inspection

Certain steps need to be taken when performing an inspection. Once, the parameters for that inspection are set up, inspectors can continue with that inspection later.

2.4.1. Selecting a project and version for inspection

In order to perform an inspection using the Usability Problem Inspection tool, the inspector logs into the UAF suite and clicks on the 'Inspection' tab. The next step is to select a *project* for inspection (refer to figure 2 on the following page).

A *project* is the target system that the inspector wishes to inspect. At any time, inspectors may be inspecting several user interaction designs and hence there may be several projects in the 'project list'. Alternatively, there is also a facility for the inspector to add projects for inspection.

The inspector must also select a *version* for each project. Project may have more than one version. For example, inspectors may suggest changes to the designer after which the designer modifies the interface and develops a new version of the same interface. An inspector may again wish to inspect this modified version of the same version.

The inspection cannot proceed until this selection has been made. By disabling unavailable options for the inspector, the inspector is forced to perform the task as expected. Disabling buttons was chosen over not displaying to prevent hiding the functionality.

Once the inspector has determined a 'project-version' combination, options of continuing an inspection, beginning a new inspection, etc. are available. Direct buttons were chosen over radio buttons to reduce number of clicks.

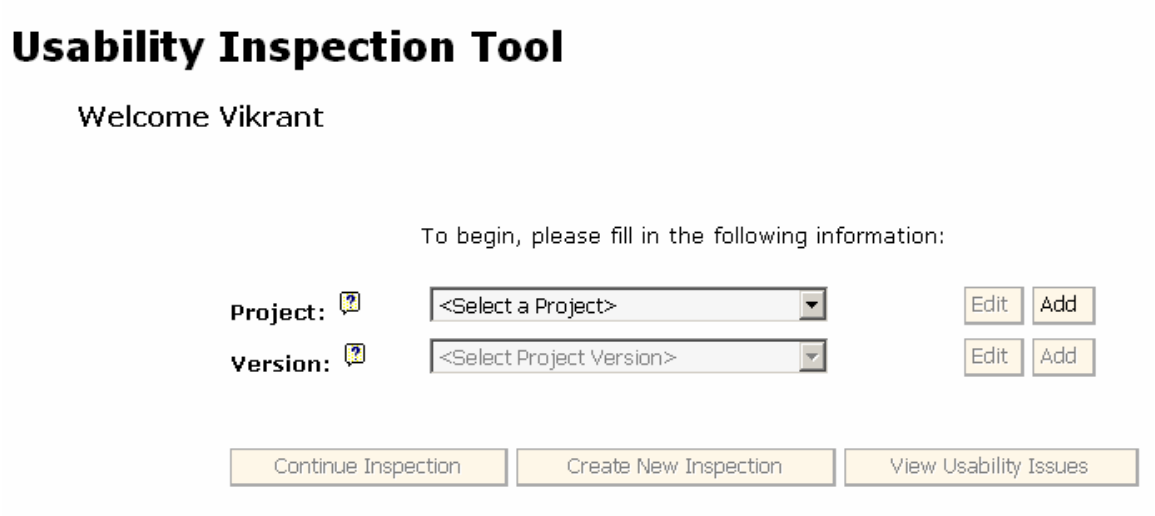


Figure 2: Selecting a project and version - In this particular instance, the inspector 'Vikrant' already has some projects in his project list. However, he has not yet selected a project. Hence, although he can select a project or add a new one, all other options are grayed out. As he makes, his selections, other buttons will get enabled.

- If no projects exist for an inspector, all buttons are grayed out except the 'Add' button.
- If a project exists, the 'Select Project' drop-down is enabled.
- As long as a project is not selected, the version input controls and project "Edit" control is disabled. Only on selecting a project, can the inspector select, add or edit a particular version for it.
- Only on selecting both a project and a version do the three *proceed* buttons get enabled.

- If an inspector selects a project-version combination for which no inspections exist, the inspector is directed to create a new inspection. If instead, multiple inspection instances exist, selection can be made on the basis of when the inspection instance was created.

2.4.2. Creation of an Inspection instance

In order to begin a new inspection, the inspector must create what is called an *inspection instance*. When creating this inspection instance, the inspector specifies various parameters that govern the behavior of the inspection.

Also, in most cases, a thorough inspection would require exhaustive browsing of the UAF, which will take considerable time depending on the inspector's familiarity with the UAF. Hence, there is an option for the inspector to stop the inspection at any time and resume it later from the point of termination. The entire state of the inspection is saved at this time so that on resuming, the inspector is at the same point as before.

Various inspection parameters that the inspector sets, govern the behavior of the inspection. They are as follows:

1. Tasks

As a context for a typical inspection, the inspector would devise a set of tasks that runs through most of the functionality of the interface being tested. These tasks are usually generic and form the context for an inspection. Alternatively, the inspector may also wish to do a general inspection with no particular task in mind. This is called a *free exploration* mode where an inspector looks at the interface as a whole, with no particular task in mind.

Since the tasks are presumably different for every inspection, the inspector has the option of adding tasks besides selecting them for an inspection of a particular project. When a new task is selected from the task list, the task description automatically changes.

On clicking one of the ‘Edit’ or ‘Add’ buttons, it was decided to pop-up a new window rather than proceed to another page. This would emphasize that the selection of parameters was the primary task while the ‘adding’ or ‘editing’ tasks are auxiliary and temporary.

2. Usage Mode

There are two usage modes for the Usability Problem Inspection tool. The initial idea was to perform an inspection with a single task at a time as context for the usability inspection. This is called the ‘Simple’ usage mode. While this may be ideal for a novice user of the Usability Problem Inspection tool and the UAF, it may prove tedious for expert users. Expert users might find it easier to inspect usability issues with *all* tasks in context simultaneously. This is called the ‘Advanced’ usage mode. This mode is more efficient and faster but may create a higher mental load on inspectors causing them to miss out on reporting some problems.

The simple mode can also be called a *task-centric* mode because the tasks are considered one at a time. The advanced mode is referred to as the *node-centric* mode because all tasks will be considered together and the inspection will proceed by examining each node. In this mode it is imperative that almost all nodes are considered. If a node is checked off saying there is no problem with it, this effectively means that the inspector has considered the usability issue associated with that node for all tasks and find no problems for *all* the tasks.

3. Filtering

In many cases, inspectors may only be interested in performing abbreviated inspection. For example, an inspector may only be interested in inspecting a graphical user interface (GUI) for the display or affordance of buttons in it.

For such a case, it is possible to apply an *inclusive* filter that only includes nodes that contain a given filter keyword in the result set. The Inspection Wizard will then only cycle through this subset of the UAF. Thus, the inspection will take less time and be more focused using a filter.

Figure 3 on the next page displays the default parameters for an inspection instance. Inspectors can change these parameters to customize their inspection. By default, filters are not set. The abstraction user interface has been developed but the implementation is not functional.

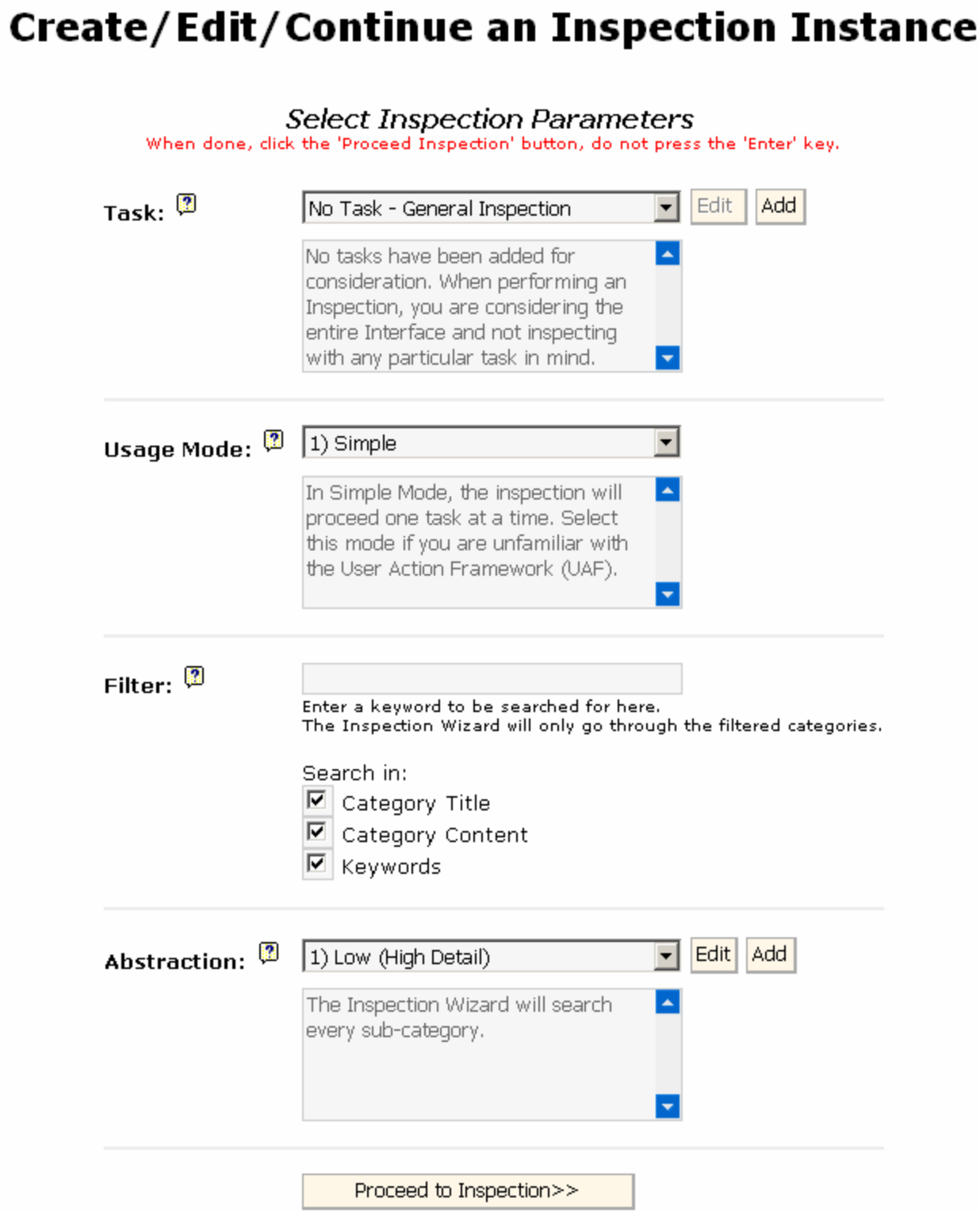


Figure 3: Selecting inspection parameters.

2.5. The Usability Problem Inspection tool

2.5.1. Inline Help

We decided to provide help within the interface of the tool itself rather than as a separate help page so that inspectors get quick pertinent help. We used a standard, intuitive, non-distracting help icon –  . Clicking the help icon produces a pop-up instead of navigating to another page. This supports the theory that help-seeking is an auxiliary task; on completion, inspectors would return to their primary task.

The help for the Usability Problem Inspection tool has been designed so that help modification is easy for the developer. The *inline* help on pages allows quick access; it is not hard-wired and is drawn up from a database of definitions. Hence, it is very easy to add or modify an explanation for any part of the tool.

For example, if inspectors wish to find out what 'project' means in the context of the Usability Problem Inspection tool, they can click on the help icon next to the term and get a detailed explanation. Similarly, if they need a more detailed description of any of the UAF nodes, they can click on the help icon.

2.5.2. Aesthetics and consistency with the look and feel of the UAF tools

The Usability Problem Inspection tool shares the UAF database with other usability evaluation tools that form a part of the UAF suite. Efforts have been made to make the interface blend in with the existing designs for the other tools.

2.5.3. Rationale for the Tree visualization of the UAF

The Inspection Wizard will proceed in a predetermined fashion once the parameters are set. The Inspection Wizard keeps a track of what nodes (usability issues) have already been considered and dismissed so that it will not re-visit those nodes. However, at any

time, inspectors might wish to take control of the inspection and navigate to nodes of their own choice. Also, in a direct node-by-node consideration, inspectors might lose context and get lost.

An inspector might start an inspection with a certain problem in mind and wish to skip directly to a sub-category of the UAF. Also, if a filter has been applied, the Inspection Wizard will only search through a subset of nodes. At some time, the inspector may wish to go back or go *out of order* from the normal process.

Thus it was decided to provide an alternative navigation means for flexibility and greater maneuverability through the UAF by which the inspector can swiftly jump to other nodes if desired. The best way to do this would be by producing a hierarchical view of the *entire* UAF. It is not just enough to display the path (or context) of the current node. Other nodes at the same level are also displayed, sometimes in a collapsed tree view, with the current node being expanded.

While the Inspection Wizard will proceed along the UAF tree in a depth-first manner, the *display* of other nodes at the same level by the tree visualization will produce more options to the inspector and perhaps help with lateral consideration of node categories too. Also, this would help the inspector develop familiarity with the UAF and help make sense of the seeming randomness with which the Inspection Wizard throws up nodes.

By displaying the current node in the tree and other nodes that the Inspection Wizard has inspected, the inspector will hopefully start to understand the progress of the Inspection Wizard and should soon be able to predict its path.

2.6. The Inspection Wizard

The Inspection Wizard is the heart of the Usability Problem Inspection tool. Using the UAF as a knowledge base, it produces a new ‘Usability Issue’ for the inspector to consider. It helps inspectors identify and report usability issues in a user interaction

design. For this purpose, it visits the nodes of the UAF one at a time and asks the inspector if there might be a usability problem in the target system that is related to the current UAF node for which the definition is provided. If the inspector feels there is a problem at this node, the problem can be immediately reported.

2.6.1. Basic Design

The Inspection Wizard guides the inspector through the Inspection by asking the inspector to consider one usability issue after the other. A preliminary design of the inspector with the basic requirements in mind envisaged *display* of a 'Usability Issue' from the UAF and then asking the inspector if the issue was a problem with the current task.

However, this is not just a 'Yes' and 'No' answer. The inspector may not be sure at that time whether there is a problem. Hence, the inspector is given three choices:

- Unsure
- No and
- Yes.

If 'Unsure' is selected, the inspection proceeds as normal, on to the next node of the UAF (*Keep Inspecting mode*). If 'No Problem' is selected, the Inspection Wizard will mark off the current category as non-problematic and no longer search for Usability Issues within that category (*Skip Category mode*). However, if there is a problem, there are two options. The inspector can either report the problem at the current stage or drill down to the exact problem (*Pinpoint mode*) (refer to Section 2.6.3 for a detailed description)

Implementation-wise, the 'Pinpoint' mode of the 'Yes' answer is not much different from the 'Keep Inspecting' mode of the 'Unsure' answer. In both modes, the Inspection Wizard will continue its depth-first search of nodes (i.e. it keeps on inspecting). However, for the inspector, there is a subtle difference. When asked if there is a problem with the target system with respect to a particular usability issue, there are any of three

distinct answers – 'Yes', 'No' and 'Unsure'. It would be confusing if the same button of 'Keep Inspecting' is mapped to two disparate courses of action. As a result, even though the implementation of both the *Pinpoint* mode and the *Keep Inspecting* mode are very similar, to the inspector they are presented as different actions.

Thus, the bare-bone interface of the Inspection Wizard must contain a definition of the UAF node being considered along with a question and four buttons which represent the different courses of action. The Inspection Wizard has been designed keeping this in mind.

2.6.2. Graphical User Interface

The Inspection Wizard graphical user interface has the following main components in the layout: the header, the usability issue, the question and buttons for the different answers to that question (refer to figure 4).

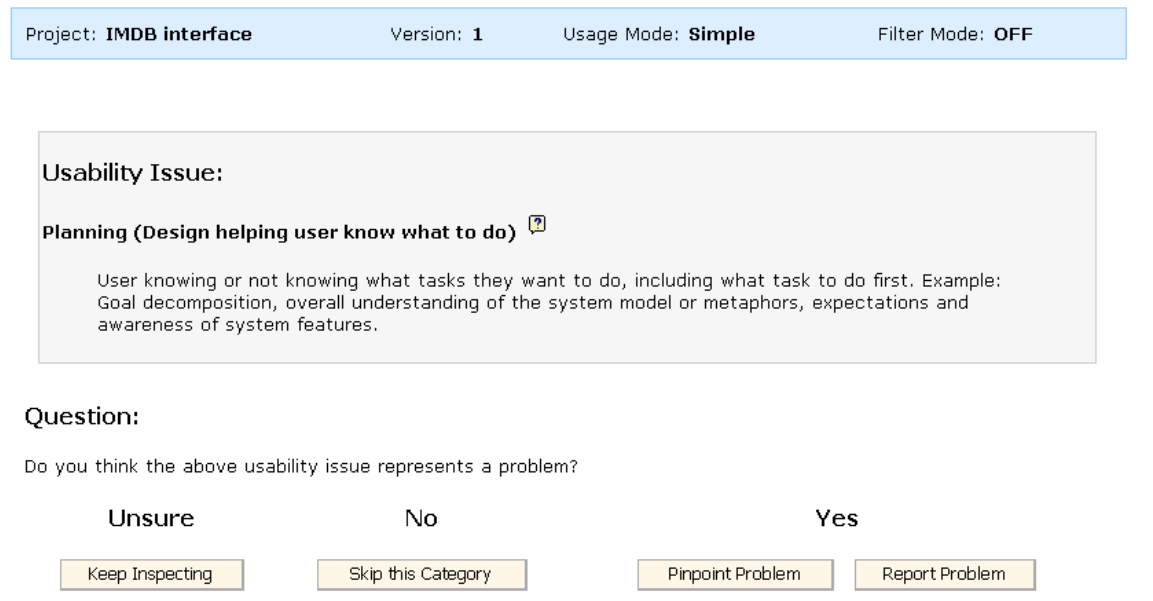


Figure 4: The Inspection Wizard.

The Header

The Header (refer to figure 5) at the top of the Inspection Wizard displays the static parameters of the Inspection Instance at any time. An inspection is performed for a particular project and version. Also, parameters like the 'Usage Mode' and whether or not a 'Filter' has been applied remain constant.

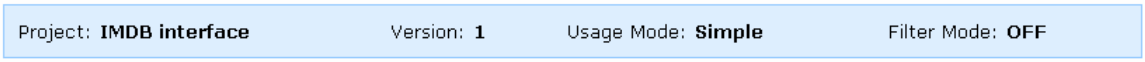


Figure 5: The Header – It provides display of static parameters for an inspection.

Boxing of the 'Usability Issue'

Since the Usability Issue is the focus of the tool, it should stand out in some way. Hence, it has been boxed and highlighted (refer to figure 6) to give it a different appearance.

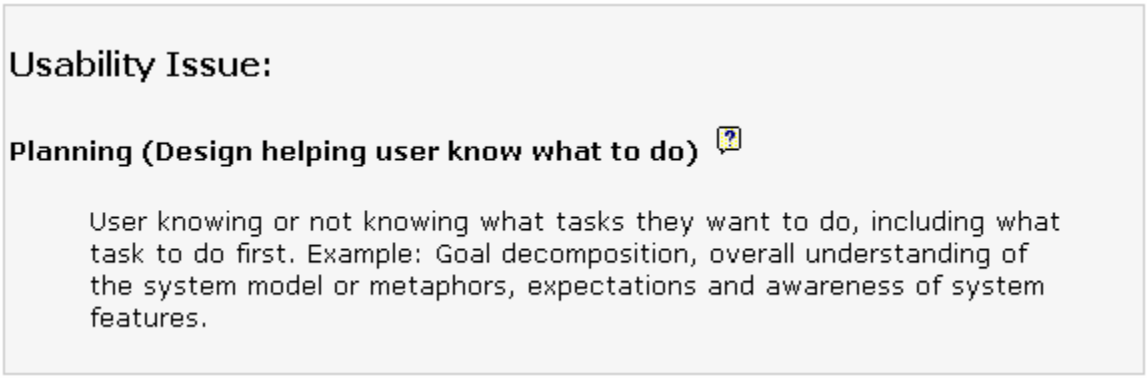


Figure 6: The Usability Issue is boxed and highlighted gray so that it stands out.

Choice of the word 'Usability Issue' as against 'Usability Problem'

Since an Inspection tool naturally supports reporting problems, initial choice of wording was 'Usability Problem' for the Inspection Wizard. However, keeping in mind the neutrality of the UAF and the fact that the Inspection Tool could also be used to report positive aspects of an interface, the wording was changed to 'Usability Issue'.

Extended definitions of the UAF nodes

An easily recognizable help icon consistent with the rest of the interface is appended to each node category name. While an abbreviated description of a node is displayed on the Inspection page for quick recognition of a node, in case an inspector is ever confused, he can instantly draw upon a more detailed description by clicking on this.

Wording for Buttons

After careful consideration, it was decided to name the buttons according to the 'actions' they perform rather than 'Yes', 'No' and 'Unsure'. As shown in figure 7, the buttons were placed below the answers to the questions posed so that it was clear that they were associated with them.

Figure 7: Buttons of the Inspection Wizard.

In the case of the advanced usage mode, the wording of the question was changed slightly so that the inspector had to select from the task list before reporting. However, instead of putting the 'Report for Selected tasks' button below the task list, it was kept in the same place as the previous report button to maintain consistency.

2.6.3. How the Inspection Wizard works

The UAF is a hierarchical representation or taxonomy of usability issues (or usability design guidelines). For the purpose of inspection, these issues must be presented to inspectors as a guide that assist them in making decisions about whether or not an interface violates these guidelines. Thus, using these 'guidelines' from the UAF, inspectors can classify and report problems with the interface being inspected.

When asked a question if there is a problem in the interface with the current category, inspectors have three possible recourses of action. They can either answer ‘Yes’, ‘No’ or ‘Unsure’. Depending on the answers, the Inspection Wizard selects a new node from the UAF for consideration.

‘Pinpoint’ & ‘Keep Inspecting’ Actions

If the inspector is unsure whether the current UAF node is a usability problem with the interface design being inspected, the Inspection Wizard will continue inspecting, moving on deeper into the UAF. Even if the inspector is certain that there is a problem in the category and answers affirmatively, the Inspection Wizard will follow a similar path of action by descending deeper into the tree. This action is called *pinpointing* and the inspector is presumably homing in on or sub classifying the exact problem in this mode. Thus the two buttons are almost identical in action but are separated out for the inspector because they are semantically distinct answers to the same question to the inspector,.

When descending, the Inspection Wizard will return the *first unvisited* child node of the current node in the UAF. As long as the inspector clicks on ‘Unsure – Keep Inspecting’ button or the ‘Yes – Pinpoint Problem’ button, the Inspection Wizard will continue its depth-first progression until it reaches a leaf node at which point it warns the user that it can proceed no longer along this path.

This is where the two buttons differ slightly. If the inspector is in ‘pinpoint’ mode and a leaf node is reached, the Inspection Wizard will produce a warning that an attempt is being made to sub-classify further in the UAF when no sub-classification exists. The inspector then has the option to either ‘Report’ the problem or voluntarily certify that there is ‘No Problem’ at the current node. The Inspection Wizard then proceeds to the next *unvisited sibling* of the current leaf node. If the inspector is not pinpointing, the Inspection Wizard will keep searching jumping to the next node in the same category and only halt and warn when it has to make a backward jump, up the tree.

If no unvisited sibling exists for the current node, the Inspection Wizard jumps to the *first unvisited* sibling of the *parent* of the current node. It will then proceed to search for usability issues in this category.

'No Problem' Action

If at any node, the inspectors click on the 'No problem' button, this would indicate that there are no usability issues for the current category. The Inspection Wizard then skips ahead directly to the first unvisited sibling. If all siblings are visited, it skips ahead to the first unvisited sibling of the parent category that it finds.

To simplify the working of the Inspection Wizard, the 'Unsure' and 'Pinpoint' modes make it perform a depth-first search, *down* the UAF. However, the 'No Problem' mode jumps laterally in the UAF, marking off categories as non-problematic. The Inspection Wizard will not visit these categories again. This might be restricting, especially if the inspector has just remembered that he wishes to report a problem at a node that he previously checked off, as non-problematic. However, this default pattern of progression through the UAF can be overridden at any time using the UAF Tree Visualization to navigate.

'Reporting' Action

When the inspector foresees a problem at a node, besides 'pinpointing', he also has the option of reporting the problem. It is not necessary to report the problem at a leaf node. A problem can be reported at any node. In fact, it is an assumption that the UAF is exhaustive in categorizing every usability issue. The truth is that the UAF is still undergoing modifications and additions, hence it is possible that when trying to sub-classify, none of the subcategories match the problem. At this point it is better to report the problem at the current node itself, rather than try to pinpoint for a non-existent subcategory.

Also, inspectors may not even wish to go to such a level of detail, as this is time-consuming and may not even be necessary as in the case of a low-fidelity, throwaway prototype.

Clicking on the 'Report Problem' button takes the inspector to a 'Report' form. After the problem has been reported, the Inspection Wizard will follow the same logic as that when in 'Unsure' mode to find the next viable node.

The report form collects data from the inspector related to the usability problem. Besides entering a description of the usability problem, suggestions for remedy can also be entered (refer to figure 8). If there are other tasks in the Inspection and it is proceeding in the 'Simple' usage mode, they are also given the option of reporting the same problem for other tasks if desired.

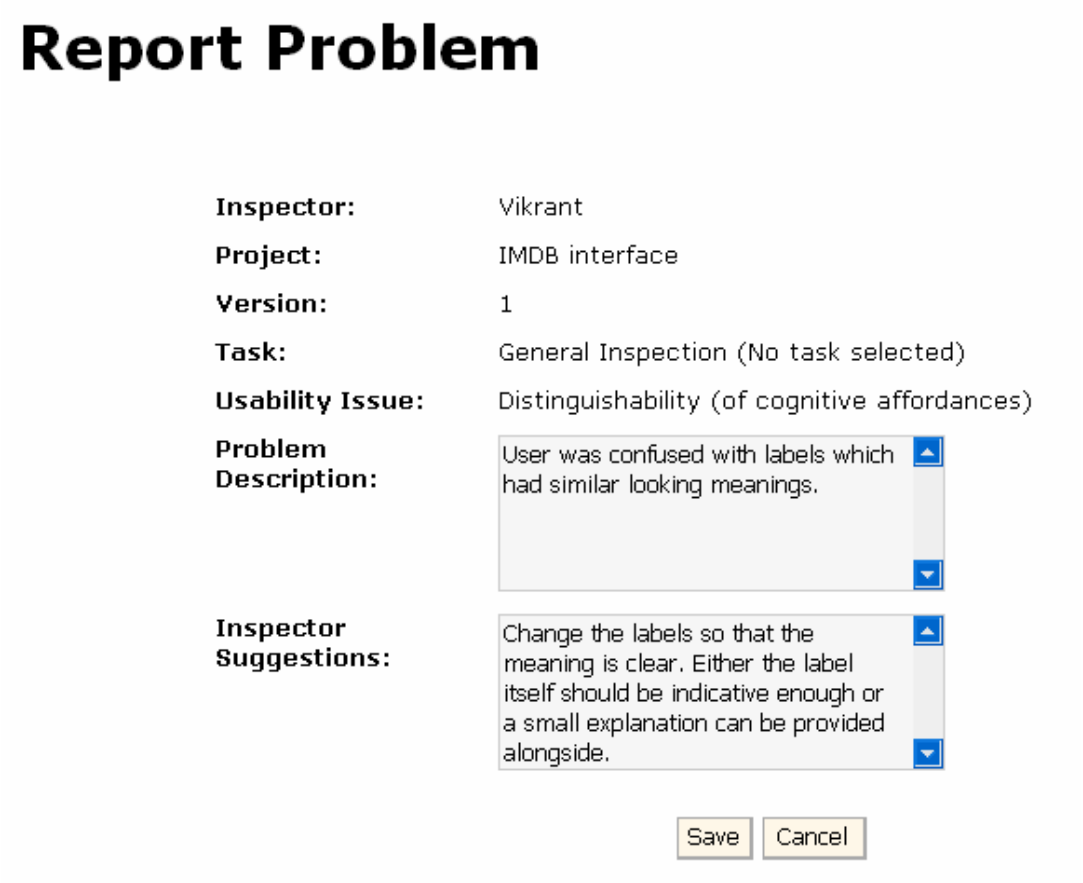


Figure 8: Report form.

Once the inspector submits the report, there is a confirmation of his report submission. Also, if there are other tasks that are associated with the user interface being inspected, there is an option for the inspector to report the same problem for them (refer to figure 9). This is a quick way of reporting and may help the inspector find more problems.

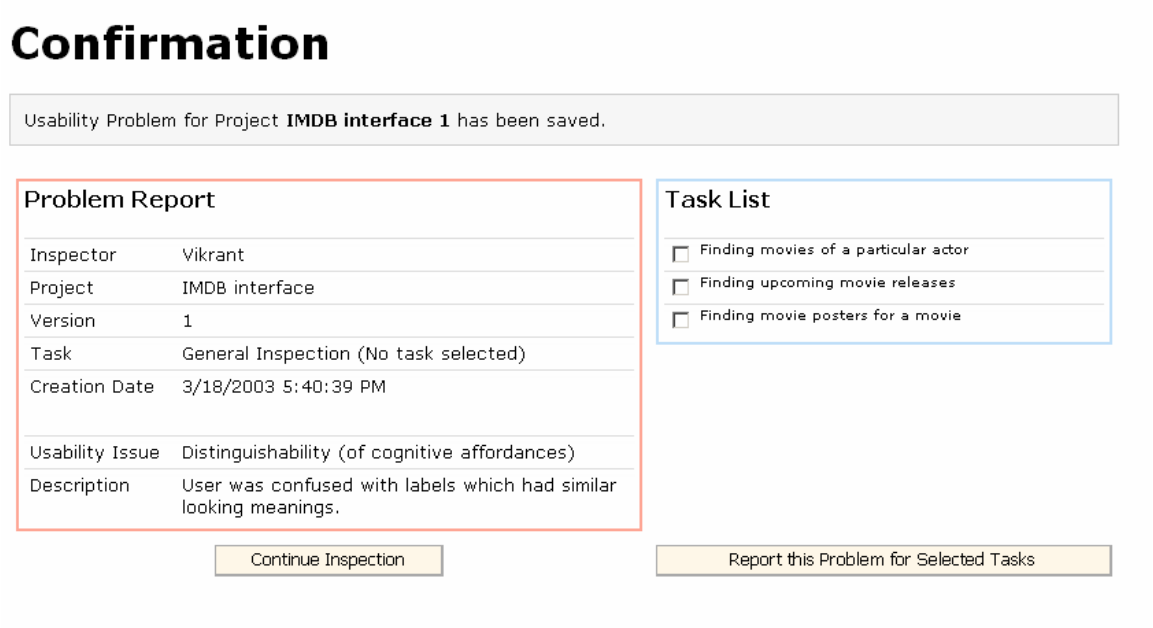


Figure 9: Confirmation of Report submission.

2.6.4. Two Usage Modes of the Inspection Wizard

The Inspection Wizard has two modes of interaction with inspectors, depending on their experience with the UAF and the Usability Problem Inspection tool.

Simple Usage Mode

In *Simple* usage mode, the inspection is carried out with a single task in consideration. Hence it can also be called a '*Task-centric Inspection*'. Here, the Inspection Wizard will ask the inspector questions (from the UAF knowledge base) and the inspector answers them, keeping a single task in mind (refer to figure 10 on the next page).

In this mode, the inspector is not exposed to or distracted by the other tasks that may also be a part of the Inspection. This should help the inspector focus more on classifying and reporting problems with a particular task. However, when an inspection is large, featuring a number of tasks, this method might prove tedious, especially for an experienced user who is familiar with both the Usability Problem Inspection tool and the UAF. Hence there is a need for an advanced usage mode that allows consideration of more than one task.

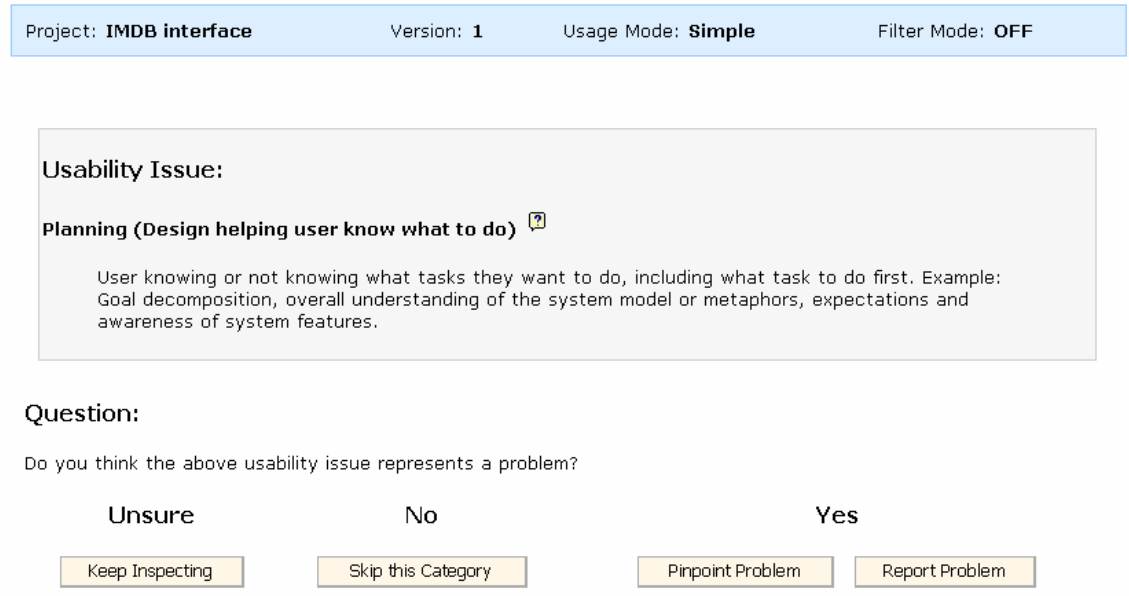


Figure 10: Simple Usage Mode.

Advanced Usage Mode

In *Advanced* mode, no particular task is selected for consideration. Instead, throughout the inspection, the inspector considers all the tasks associated with a project. For this mode, a task listing that was prominent at all times has been displayed so that the inspector can easily check each task with the current node that the Inspection Wizard presents (refer to figure 11 on the next page). Thus, the Inspection can be called a '*Node-centric Inspection*'. For every node, all tasks must be inspected. If there is a problem at any node, the inspector can check it off and report the problem.

While this may definitely be a much faster way to perform the inspection, it will presumably put a greater degree of strain on the user. There is also some ambiguity about how to resolve the issue of pinpointing here. While the inspector may identify a problem with one task in the task list, other tasks in the list may not have the same problem. Also, as the number of tasks increases, it may become increasingly difficult for the inspector to 'anticipate' whether there would be any problems in subcategories for *all* tasks. Hence, it would probably be better to exhaustively go through every node in the UAF (hence the term node-centric). In that case, the *pinpointing* action itself becomes redundant, and all the inspector does is consider every node of the UAF and inspect all tasks for that.

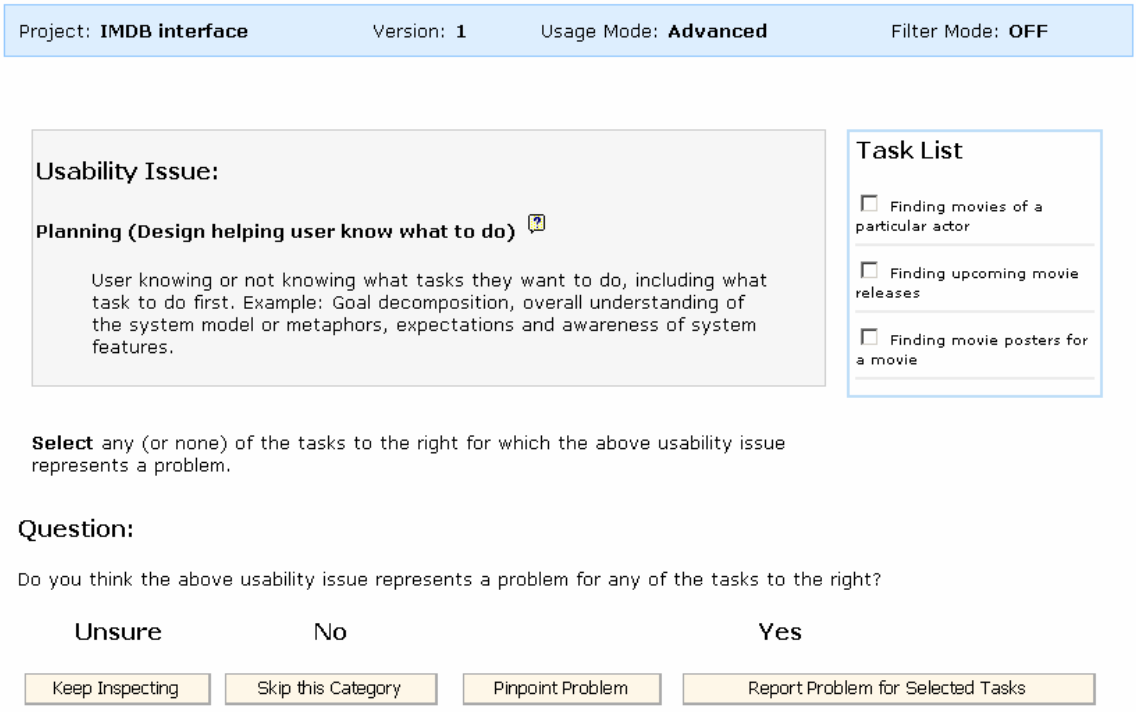


Figure 11: Advanced Usage Mode.

2.6.5. Terminating an inspection

An inspection is only complete when the Inspection Wizard has guided the inspector through all categories. The inspector need not visit every node of the UAF; when

checking off a parent node in the UAF as non-problematic, all the child nodes are also checked off as non-problematic. However, a thorough usability inspection may take a long time and it may not be able to complete one in a single sitting.

For this reason, inspectors can leave an unfinished inspection and at any time return to the inspection. The Usability Problem Inspection tool stores the entire state of an inspection, even past history of which UAF nodes were visited so that on resuming an existing inspection, inspectors will be able to pick off at exactly the same point.

Although inspections may take a while (and be resumed at a later stage), they are usually exhaustive. Hence for the Usability Problem Inspection tool, there is no explicit way to terminate an inspection. The only way is to complete the inspection.

2.7. The UAF Tree Visualization

The UAF Tree Visualization was added on later as a supplementary navigation method to allow for more control and feedback during an inspection. For instance, an inspector might wish to take control of the Inspection and not rely on the Inspection Wizard to direct the inspection. In other words, if he wanted to directly skip ahead to a usability issue, the Inspection Wizard would not allow him to do so. However, using the UAF Tree Visualization, he can directly jump to any part of the UAF and report a problem there.

Also, the UAF Tree Visualization provides some feedback of the Inspection progress like which node of the UAF is currently active, which nodes have been marked off as having no problems, which nodes have problems, which nodes have already been considered, etc.

The visualization has the following attributes:

- It displays the currently active node
- It displays the context (path) of the current node
- It displays other nodes in the UAF
- It allows inspectors to navigate to any of the other nodes

- If a node has already been visited, the Inspection Wizard will not revisit it and so the inspector must be made aware of this. Hence, an indication is provided that a particular node has been visited.
- There is an indication for nodes where Usability Issues have been identified.
- There is also an indication for nodes, which are part of the filter set when a filter is applied.

As the Inspection progresses, the UAF Tree Visualization also changes to provide feedback to the inspector. As shown in figure 12, any unvisited nodes are marked *black* in color. Once an inspector has 'considered' a node, it is color-coded *green* in the UAF Tree Visualization. Simply visiting a node (using the UAF tree, instead of the Inspection Wizard) will not cause the green color-coding to be applied. Only when the inspector has presumably spent some time considering the Usability Issue and then clicked on one of the four buttons of the Inspection Wizard does the Usability Problem Inspection tool mark off that node as being 'considered' and color codes it green.

The currently active node is color-coded *yellow* and the font is emphasized. Other cues provide feedback about the inspector's actions. When the inspector clicks on the 'No problem' for a particular node, a 'green checkmark' appears on its side. If he reports a problem, a 'red exclamation' mark appears at that node.

After much deliberation, it was decided that a 'green tick - ✔' should be used to indicate that no problems existed at that category when an inspector decided that no problems existed for that category. When a problem has been reported a 'red exclamation - **!**' would indicate this.

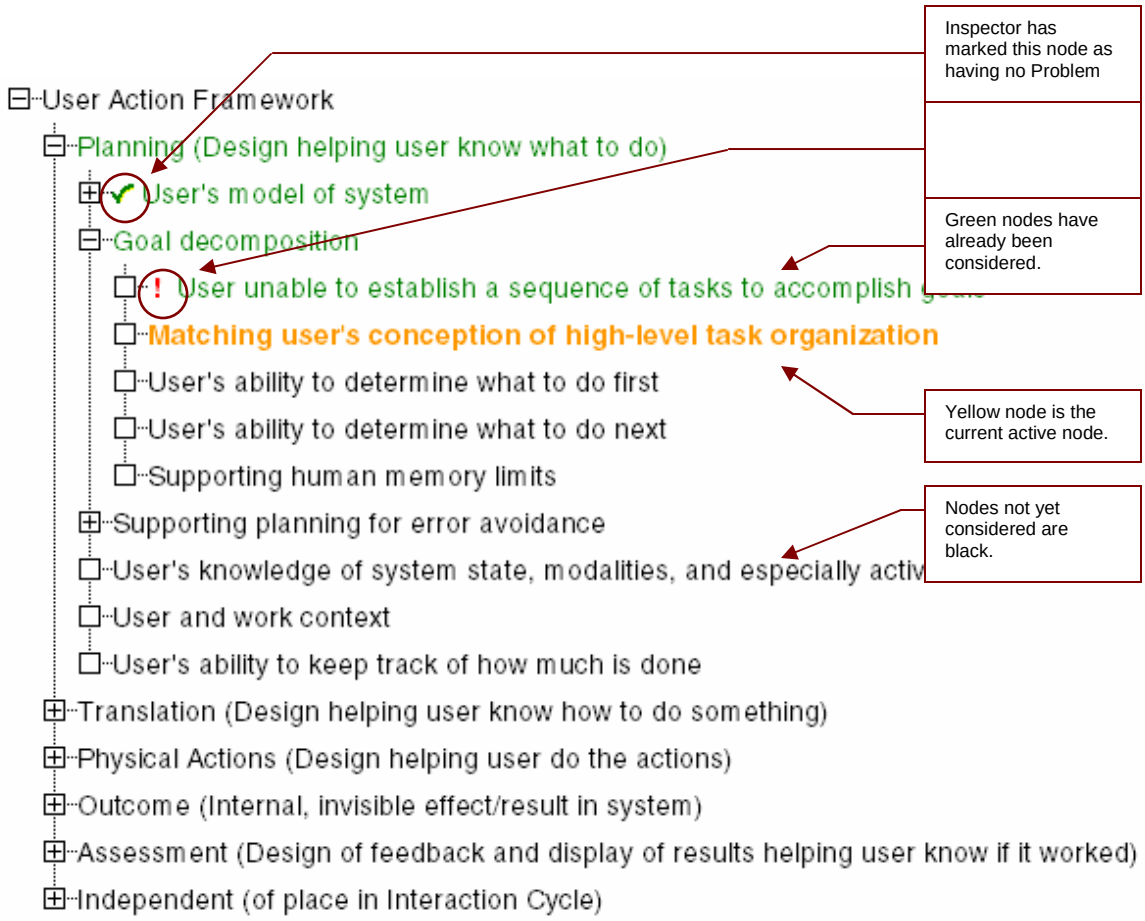


Figure 12: The UAF Tree Visualization.

The UAF tree is always completely expanded for the current active node. Other parts of the tree are usually collapsed, unless the inspector has explicitly turned them off. While each of the nodes is a hyperlink, the normal affordance of a link (blue underlining) is hidden so that it does not distract the inspector. Also, it is tedious to read long strings of text like the UAF category names when they are underlined. However, when you hover over the links, they change to blue underlined hyperlinks.

2.8. Interaction between the Inspection Wizard and the UAF Tree Visualization

In normal operation, the Inspection Wizard produces a node that the inspector is asked to consider. At this point, the UAF Tree Visualization displays the current active node in

bold yellow color. Other nodes that have been 'considered' will be displayed green while nodes not considered are displayed black (refer to figure 13 on the next page).

Now consider the following scenario. The inspector, during an Inspection, suddenly remembers some other problem in the interface being inspected and wants to immediately classify it. If he now follows the normal course of the Inspection Wizard, it may take a while and he may ultimately even forget the problem. By clicking and navigating through the UAF Tree, he is able to quickly find the pertinent category and report the problem.

The feedback provided by the UAF Tree Visualization may be used by the inspector to find out which nodes he has already visited, which nodes he has reported problems at, which nodes he has checked out as having no problems, etc.

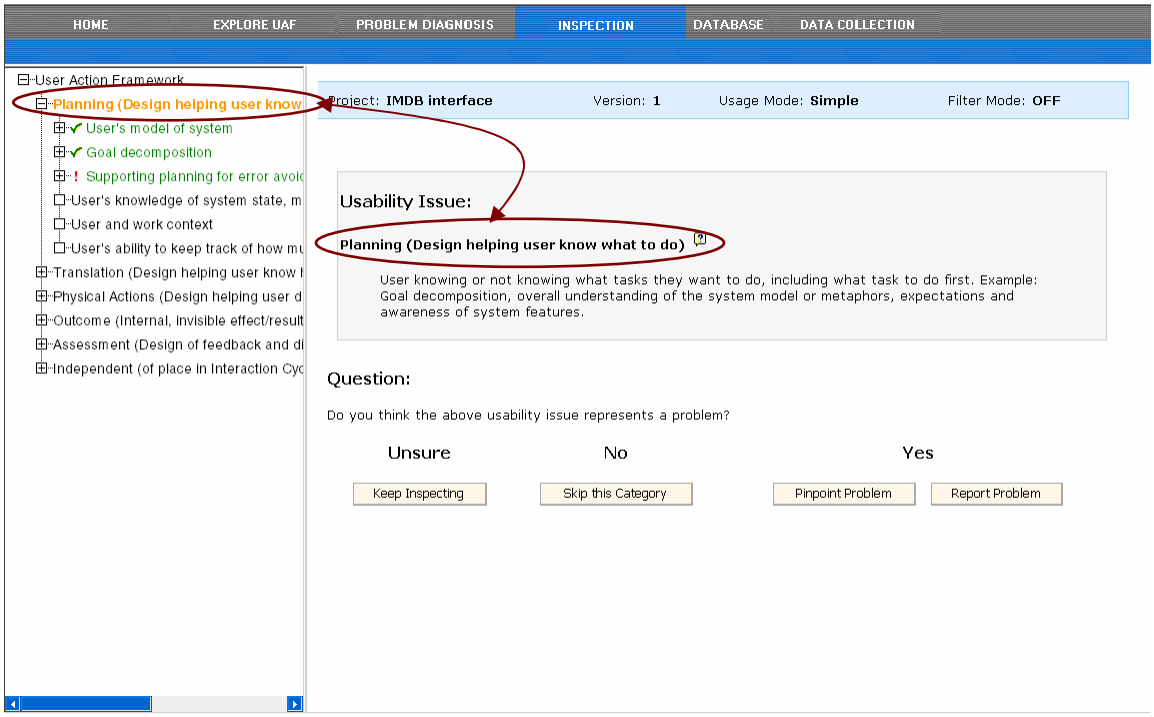


Figure 13: Interaction between the Inspection Wizard and the UAF Tree Visualization - Clicking on any of the UAF nodes on the left, loads that node in the Inspection Wizard in the right frame.

2.9. Filters

A complete usability inspection is usually expensive and often not necessary. The filter option helps focus a usability inspection and limit its scope. The filter is a way to prune the UAF tree to reduce the number of nodes visited in an inspection, thus reducing the number of questions asked by the Inspection Wizard. Thus, filters help contribute to the cost-effectiveness of the Usability Problem Inspection tool.

In order to apply a filter, the inspector specifies a filter word and then uses checkboxes to select the fields where the filter should be applied. These fields are the UAF Category Title (node name), UAF Category Content (node description) and Keywords (a table that indexes keywords in the UAF). The Keywords table stores all nodes associated with a particular word. Hence even if the word does not appear in the node name or node description, the Keywords table can be used to associate the word with that node. For example, the keyword ‘button’ could be associated with nodes that deal with labels and affordances.

For a particular inspection instance, when a filter is applied, only nodes that are associated with the filter word will be included during the inspection. The Inspection Wizard does not go through any of the nodes that are filtered out. They are marked off as having already been considered by the Inspector. However, using the UAF Tree Visualization, the Inspector is free to jump to these nodes at any time.

As an example of how the filter works, consider an inspector who may be evaluating the affordance of labels on a website. The inspector keys in the filter word - ‘button’ and checks off all fields for consideration. In this scenario, only nodes that contain the filter word ‘button’ in their titles and descriptions or nodes that have been indexed for the ‘button’ keyword will form the limited filter set that the Inspection Wizard focuses on. All other nodes will be ignored for the inspection.

Nodes that are ignored can still be visited using the UAF Tree Visualization. They are still an important part of the inspection as the provide context for the nodes that are part of the filter set. Hence, filters while limiting the inspection to fewer nodes, do not destroy the structured framework that the UAF provides to guide the inspection.

When a filter is applied for the Usability Problem Inspection tool and either of the 'Keep Inspecting', 'Pinpoint' or 'No Problem' buttons are pressed, the Inspection Wizard does not follow on its normal course to the next node in order, but instead directly jumps ahead to a node that is part of the filter result set. In the process, it also marks off all intermediate nodes as having been considered. The filter icon - will be displayed next to nodes that are part of the filter set.

In the example shown below, the inspector has applied a filter for the word 'button', hence the Inspection Wizard has ignored other nodes and jumped directly to a node that is associated with the filter word. For this particular node, the filter word 'button' is part of the node description.

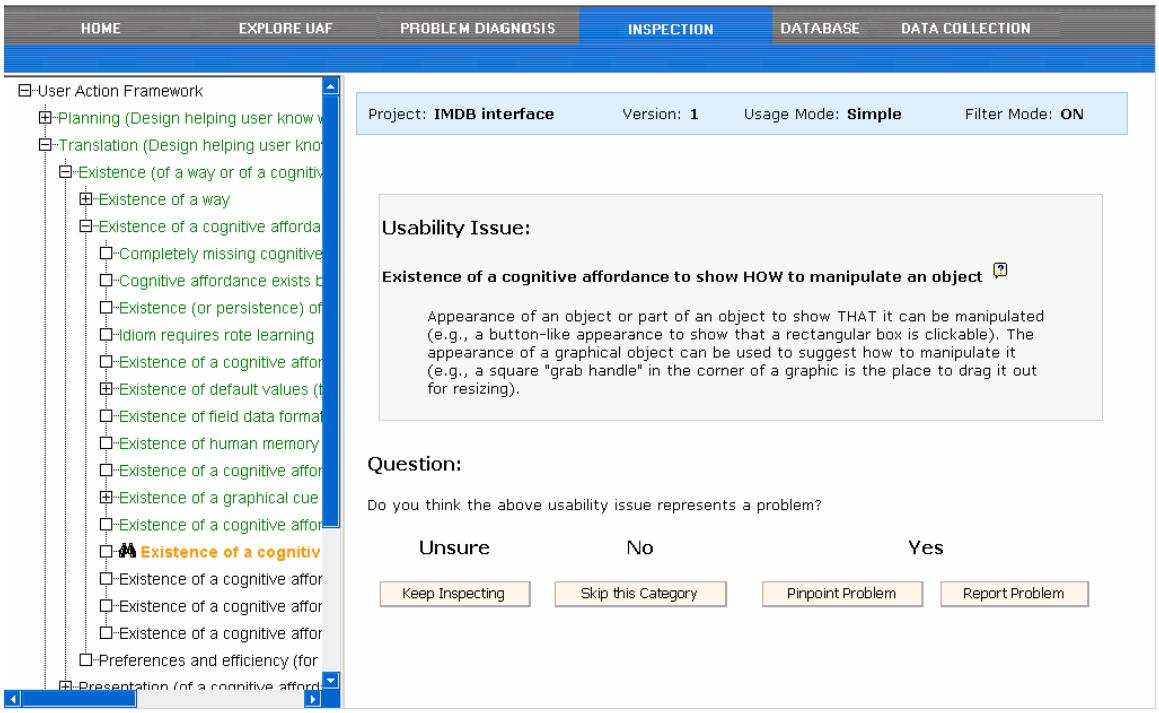


Figure 14: The Filter Action

2.10. Abstraction

Besides filtering as a means of reducing the number of nodes in an inspection, abstraction will also help prune the UAF tree. With abstraction, inspectors can toggle on or off lower levels of detail for consideration by the Inspection Wizard thus preventing entire sub-categories of the UAF from being visited. The UAF is an exhaustive knowledge base of usability issue and at times it may not be necessary to perform an inspection that goes through the entire UAF.

For example, when evaluating a low-level prototype, inspectors do not expect low-level details like font colors, font sizes, etc. to be important and hence for the purpose of the inspection, these low-level categories should not even be considered. Restricting the number of nodes should focus the inspection saving time and effort and adding to the cost-effectiveness of the tool. (Also refer to section 5.1).

3. Formative evaluation

A *formative evaluation* is an evaluation of the usability of a user interaction design for the purpose of finding and fixing usability problems to improve the design. It is usually a continuous iterative cycle of design, evaluation and redesign [Hix & Hartson, 1993].

For the purpose of this study, a formative evaluation was conducted over the whole life cycle of the design and consisted of two kinds of evaluation: a large number of cycles of design walk-throughs (refer to section 3.1) and one final usability testing study (refer to section 3.2). This formative evaluation is intended to be a part of an even larger iterative life cycle that includes future developers of the tool. Further development is required, which can be worked upon as suggested in section 5.

3.1. Iterative Design Walkthroughs

The current design of the Usability Problem Inspection tool has come about from numerous iterative design walkthroughs during the development. Dr. H. Rex Hartson, my thesis advisor and an experienced usability engineering professional, would act as a typical user, identifying and also anticipating potential usability errors during the design itself. This brought about a new modified design, which was again evaluated in the next design walkthrough and modified if necessary. User interface design also followed general design guidelines. The following section is a description of the outcome of these iterations.

3.2. Laboratory-based Study

3.2.1. Before Data Collection

Participants

For the purpose of the evaluation study, we tried to select participants who would match the intended user class as closely as possible. Typical inspectors would be familiar with both the UAF and the usability inspection process. We conducted a usability study with seven users, all of whom had more than average usability experience. Users were asked to fill out a self-evaluate their knowledge of the UAF and usability in a feedback form at the end of the study (refer to figure 15 on the next page). Six of the users had taken the graduate-level course in *Usability Engineering (CS5714)* and one had taken a similar undergraduate course. All users had participated in a usability evaluation before.

Many of the users had a fair knowledge of the UAF and had either done course projects using it or research with it. A brief tutorial was also given before the test to familiarize users with the UAF (refer to http://hemlock.cs.vt.edu/uaftraining/uafquicktrainer.htm) since the tool requires familiarity with the UAF. The tutorial is in the form of a brief summary of the high-level categories of the UAF, followed by a few examples of typical problems encouraging the participant to guess the classification as per the UAF. Solutions were provided along with reasons why other choices were wrong.

While knowledge of the UAF is not crucial to the usability testing of the Usability Problem Inspection tool, it was a desirable aspect so that testing would proceed smoothly.

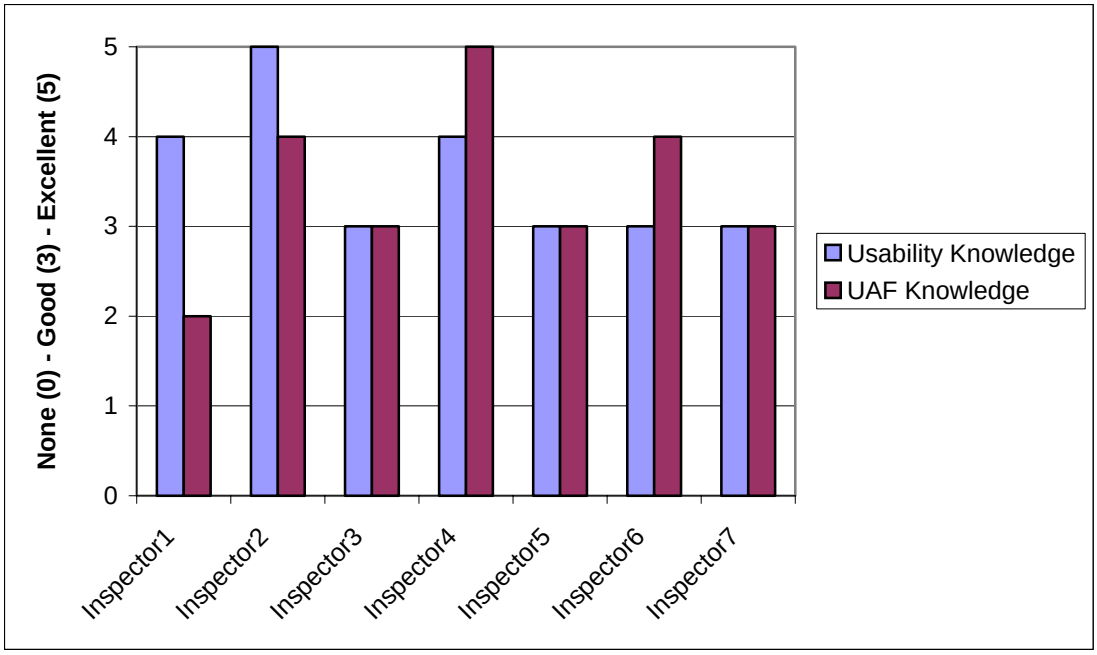


Figure 15: Participant Knowledge of the UAF and usability.

Experimental Setup

While we made use of usability testing laboratories for the experiment, we had to set up new equipment for the purpose of this test. Older recording of the screen by VHS tapes was replaced by direct digital video capture. The equipment setup is detailed below.

Laboratory Rooms

For the purpose of the experiment, two adjacent usability-testing labs were used. Users would sit in one lab and the evaluator in another so that users would be more comfortable. The two labs had an intercom system so that the evaluator and user could communicate if the user had a problem. The evaluator room also has a one-way mirror, which users were made aware of, so that the evaluator can see the user.

Screen Control and Capture

Even though the evaluator was in the other room, it was possible to view the same screen that the user was seeing. This was made possible by using VNC (Virtual Network Computing), which feeds the evaluator with a continuous live stream of the user's desktop. Using VNC, it was even possible to control the mouse and keyboard at the user's computer. This was very useful at times, for example, to reset screens between benchmark tasks or when a software bug surfaced or when users were totally confused by a usability error and could not proceed further.

A video was made of the user's screen action using screen capture software – *Camtasia*, at the inspector's machine. Using a sound mixer, voice input from both the user in the other room and the evaluator could be added into the video stream. The video format is in the form of ".avi" files. Using the video, the evaluator was able to later review the experiment session in detail. These videos helped in further identifying and analyzing critical incidents.

Dual Monitor Display

At the evaluator's machine, a 19-inch dual monitor display was used while a 17-inch dual monitor display was used at the user's machine. This gave the evaluator screen space to keep other important windows open while the testing was on.

Since we were setting up the equipment for the first time, a short equipment manual (refer to Appendix C) was created so that other evaluators using the laboratories would also be able to use it.

3.2.2. Data Collection

Procedure

The Usability Problem Inspection tool was maximized on one monitor and the IMDB interface being inspected was maximized on the other monitor for the user. This helped minimize the tediousness of having to deal with window management. Also, we feel it

would mimic a typical environment, for example, when inspectors perform inspections, they might use the Usability Problem Inspection tool on a personal laptop and inspect the user interaction design on another machine (it may not even be a computer software).

The users imitate typical inspectors. Hence for the purpose of testing the Usability Problem Inspection tool, they perform mock inspections on a user interface. The interface chosen as a target for the inspection was the website (http://www.imdb.com) of the Internet Movie Database (IMDB). However, rather than identifying usability errors of the IMDB site, the main aim of the study is to identify usability errors of the Usability Problem Inspection tool which they were made aware of this.

As the inspectors encountered problems in the interface, they reported them. Both the inspectors actions as well as audio inputs were captured using the laboratory setup described earlier. The evaluator also took down notes of problems as the inspectors reported them along with the time they occurred relative to the start of the inspection session. The time noted down, helped home in on the problem, during review of the video later. Also, it was not necessary to go through the entire video (which often ran for 40 minutes to an hour) later because of the notes taken down simultaneously.

A combination of the notes and video reviewing was used to generate a list of usability problems (with the Usability Problem Inspection tool) for each inspector. These usability problem lists were later combined to get a final list of all usability problems found.

Benchmark Tasks

For the tests, three benchmark tasks (see Appendix A) were devised. The purpose of these tasks was to go through most of the functionality of the Usability Problem Inspection tool so that inspectors would test its usability. The first two benchmark tasks test the simple and advanced usage modes of the Usability Problem Inspection tool respectively. The third task tests the *filter* option of the tool.

For example, here is a sample of the first benchmark task, which was designed to test the functionality of the simple mode of the tool. It involved a simple generic user task of *finding all movies of a particular foreign language of a particular genre that released during a particular year.* In particular, the following user task was suggested – "Finding French *mystery* movies that released during years 1966-1967".

3.2.3. After Data Collection

A feedback form (see Appendix B) was distributed after the experiment. This was used to gauge user experience with usability engineering and the UAF and also to collect suggestions and comments about the tool. The feedback form asked inspectors to rate their usability knowledge, UAF knowledge and whether or not they had participated in a usability evaluation study or taken a usability engineering course before. They were also asked for suggestions and recommendations about the tool and the experimental setup.

3.3. Results

3.3.1. Usability Problems found

A total of 33 usability problems were identified in the Usability Problem Inspection tool. These have all been listed in a problem sheet in Appendix D. Usability Problems have been sorted according to the interface screen where they were present. Suggestions for improvement of design have also been provided. These came either both from the inspectors and the evaluator. Also, 14 minor software bugs were also found.

After the study, a feedback form was distributed (refer to Appendix B). Most of the suggestions and comments from users were complimentary. A majority of the comments were recorded during the video recording and were helpful in capturing the usability problems and suggestions for improvement.

All the users were comfortable with the experimental setup and many liked having a dual screen during inspection. One of the users thought there might be a bit of a 'learning

curve to understand the tool concepts and instructions'. Another liked the aesthetics of the tool and found it pleasing to the eye, but recommended sticking to conventional text boxes to avoid confusion.

3.3.2. Cost-importance analysis

Formative evaluation usually results in finding a large number of usability problems, often beyond what resources can rectify. Our formative evaluation resulted in identification of 33 usability problems. Hence we decided to use a technique called cost-importance analysis as explained in Chapter 10 of Hix & Hartson [Hix and Hartson, 1993] to determine which problems to focus on and which to fix.

Cost-importance analysis allows prioritization of usability problems so that it is possible to determine which can be fixed with the limited resources. A priority ratio is determined; a numeric estimation of importance of the problem divided by the estimated cost to fix the problem. Importance in this context, is the importance to fix the problem and can be based on problem severity and also other variables such as marketing inputs, management judgment etc. The cost is calculated in man-hours required to fix the problem. Problems are then sorted on the basis of this priority code and a cut-off point is determined for solving these prioritized problems. Higher the ratio, greater the priority to fix the usability problem.

For each of the usability problems found, we made an estimation of the time (in man-hours) it would take to fix the problem. Also, on a scale of 1-5, we estimated the importance to fix (or seriousness of) a usability problem where importance = 5 meant that the problem was most serious. The Importance/Cost ratio was calculated and problems have been sorted accordingly (see Appendix E). Higher the ratio, greater the priority to fix the usability problem.

We drew a cut-off at an Importance/Cost ratio of 4000 and picked the first 11 problems to fix. Problems have been fixed according to the remedies suggested in the usability problem sheet (see Appendix D).

3.3.3. Discussion of significant usability problems

Amongst the usability problems found, we discuss the more serious problems below. Also refer to Appendix A (problem numbers are indexed).

Confusion over the 'Pinpoint' and 'Report' buttons [Problem #6]

Most of the inspectors had a problem deciding which of the two buttons to click on after detecting a usability problem (refer to figure 16 on the next page). It was not immediately clear that the 'Report' button meant filling out a form. Also, it was not easy for them to understand what the pinpoint button does either. (For a further discussion, refer to section 3.3.4)

Besides differentiating between the two buttons, most people had difficulty understanding their functioning. The text labels did not seem to convey enough about what would happen next. After clicking on the 'Pinpoint', 'Keep Inspecting', or 'Skip this Category' button, another node is visited with no indication why that node was chosen, creating confusion.

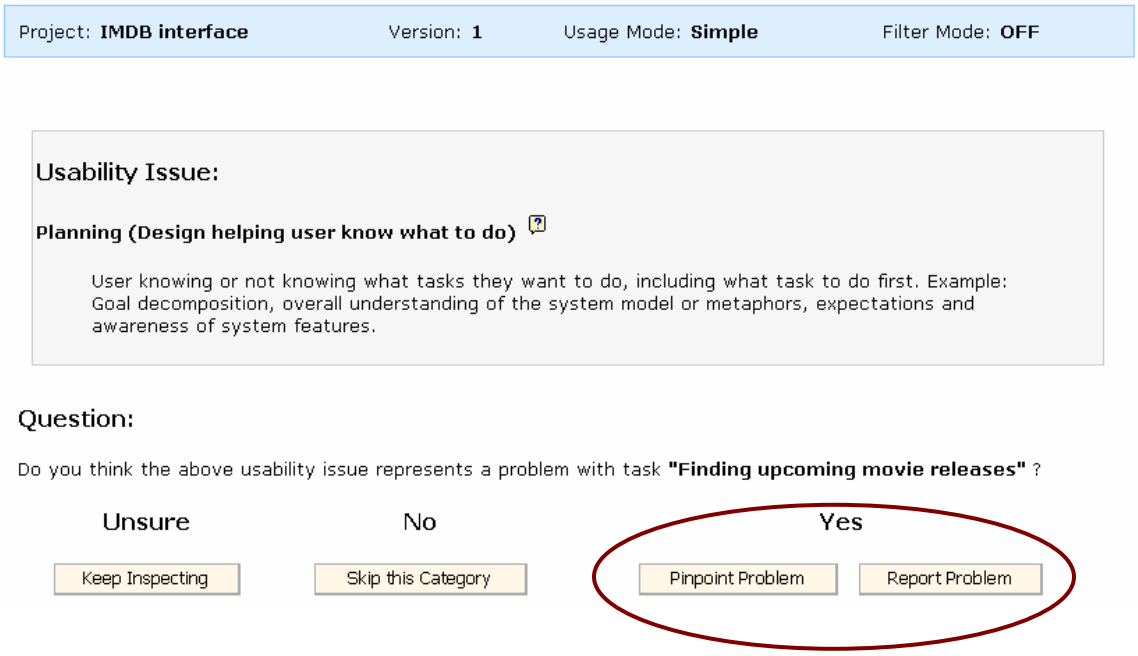


Figure 16: Inspectors had problems understanding and differentiating between the 'Pinpoint Problem' and 'Report Problem' buttons.

Date format entry problem [Problem #1]

There was an error with the date entry when adding a project. The date format that appeared in the input text box disappeared when clicking inside the box. Most inspectors forgot what the format was after clicking inside the box and there was no way to get back this desired format.

No way to terminate the inspection [Problem #7]

Some of the inspectors felt that there should be a way to terminate the inspection at any point. This feature is missing and the only way an inspection can terminate currently is when all the nodes have been visited.

Confusion over the 'select and report' feature in the advanced usage mode [Problem #11]

In the advanced usage mode, many inspectors did not associate the task list with the 'Report' button (refer to figure 17 on the next page), although instructions were given to select and then report. Since there was so much of a separation, they did not understand that selecting was only to be done at the time of reporting. As a result, some users selected tasks even while pinpointing, which was not supposed to be done. They then expected the task list to change according to what they had selected.

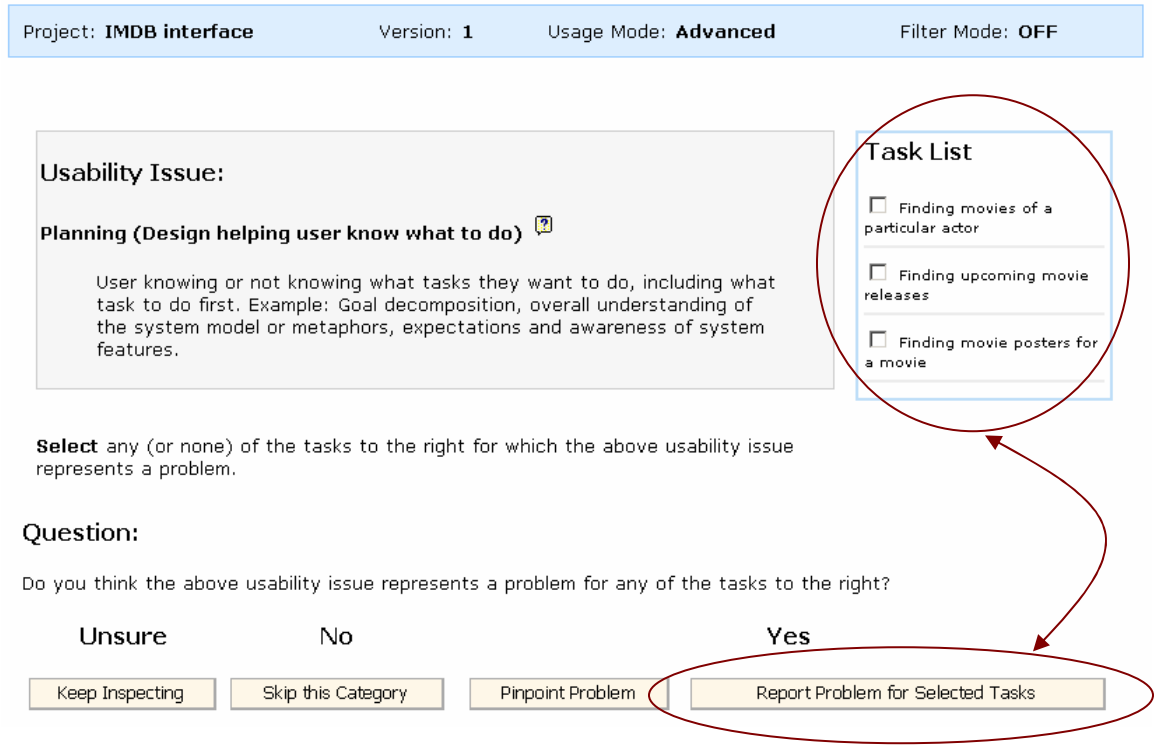


Figure 17: Inspectors had problems relating the 'Report button' with the task list.

Inspectors could not decide which task to select for inspection in an advanced mode [Problem #21]

When setting the inspection parameters, advanced mode considers all the tasks at a time, hence there is no need to select a task. Inspectors were confused with this. Most selected a task anyway and were then puzzled that all tasks showed up in the task list.

3.3.4. Different ways of performing the inspection

The Usability Problem Inspection tool has been designed with the Inspection Wizard as the main focus. The tree view has been added only as an alternative way to navigate. If an inspector has a particular problem in mind, it will be easier to use the tree to directly navigate to the UAF node at which the problem is and report it there.

Thus, there are two ways in which inspectors can use the Usability Problem Inspection tool. First, inspectors can let the Inspection Wizard guide them with the inspection. Second, taking a more active role, inspectors can directly home in on problems using the UAF tree.

However, there is a danger with using the UAF tree alone. Inspectors might be able to report all the problems they already have in mind but not find any new ones. The Usability Problem Inspection tool may be reduced to a mere reporting tool. On hindsight, though this requires further evaluation, it may not be too advantageous to display the UAF tree.

3.3.5. Difficulty in understanding the ‘Pinpoint’ and ‘Keep Inspecting’ actions

One of the most significant aspects of the Usability Problem Inspection tool was the design and wording of the Inspection Wizard buttons. We spent a considerable amount of time trying to work out button names that would be indicative about the actions being

performed. Inspite of this, some inspectors were confused with the buttons (refer to section 3.3.1, problem #6).

When inspectors notice a problem, the wizard helps sub-classify the problem. We needed an appropriate word that would indicate that the next node returned by the Inspection Wizard is a sub-category of the current node. We felt 'pinpoint' was the best choice and seemed to represent this action. Also, when there was no problem with the interface, we chose to use the 'Skip this category' label which we felt would indicate that the Inspection Wizard would no longer follow a path below the current node.

However, the 'No Problem', 'Keep Inspecting' and 'Pinpoint' buttons contain a huge amount of meaning hidden from the inexperienced user. There is a complex and subtle model of navigation behavior (refer to section 2.6.3) behind these buttons that is a very important part of the tool and that cannot possibly be conveyed to tool users by a few short button labels.

We believe something more should be provided – possibly the  help icon that has been used in other parts of the interface should supplement the button labels with a detailed description. Alternatively, even a separate tutorial on this portion could be provided.

3.4. Redesign for improvement

Depending on the usability problems noted, the various interfaces were modified. Some of the changes made have been summarized below. Please refer to Appendix D for a detailed explanation of the problem along with the problem number.

3.4.1. Welcome page

Many users clicked on the 'Continue Inspection' button (Problem #30), even when they needed to create a new inspection, perhaps mistaking it to be a 'Continue' button. Also,

when unsure what ‘Project’ and ‘Version’ were (Problem #31), they clicked the help icon -  but the help was unsatisfactory. Also, after adding a project, it did not show up in the page by default and the page had to be refreshed (Problem #32) to display it. These problems have been fixed (refer to figure 18).

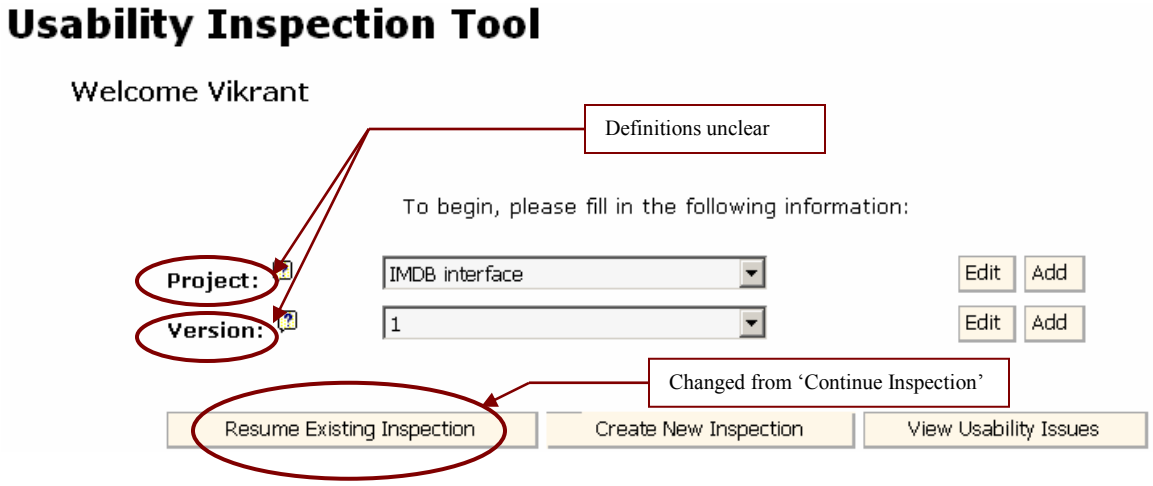


Figure 18: Labeling and definition problems were fixed in the Welcome page.

3.4.2. Add Project interface

Some users misunderstood the ‘Name’ field and filled in their own names instead of the ‘Project name’ (Problem #2). Also, the date format was required unnecessarily in one place and kept disappearing after clicking in the input field (Problem #1). This field now automatically gets filled in with the current date (refer to figure 19).

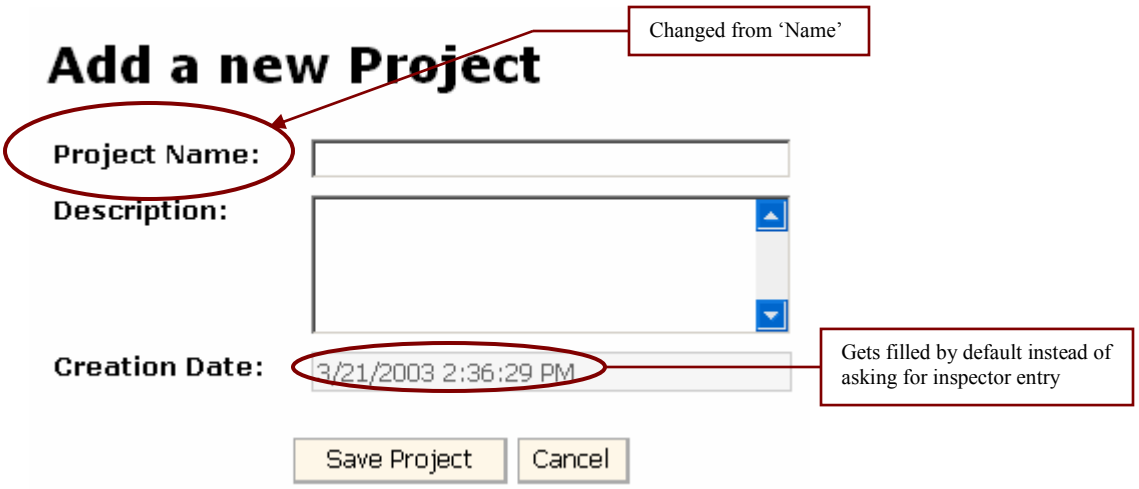


Figure 19: 'Name' label and 'Creation Date' input box were fixed.

3.4.3. Setup Inspection Parameters page

Numerous problems (Problems #17, #18, #19, #24) were fixed. The biggest problem was that newly added tasks would not appear in the page unless it was refreshed which was not clear. The page now automatically refreshes after adding a new task and the new task appears selected by default. Also, it was decided to add the filter icon here, so that it was familiar to the inspector when he used the filter (Problem #25). 'Description' labels were added for clarity in places. The checkboxes had been checked by default previously, leading users to believe that the filter was being applied when it was actually not (refer to figure 20 on the next page).

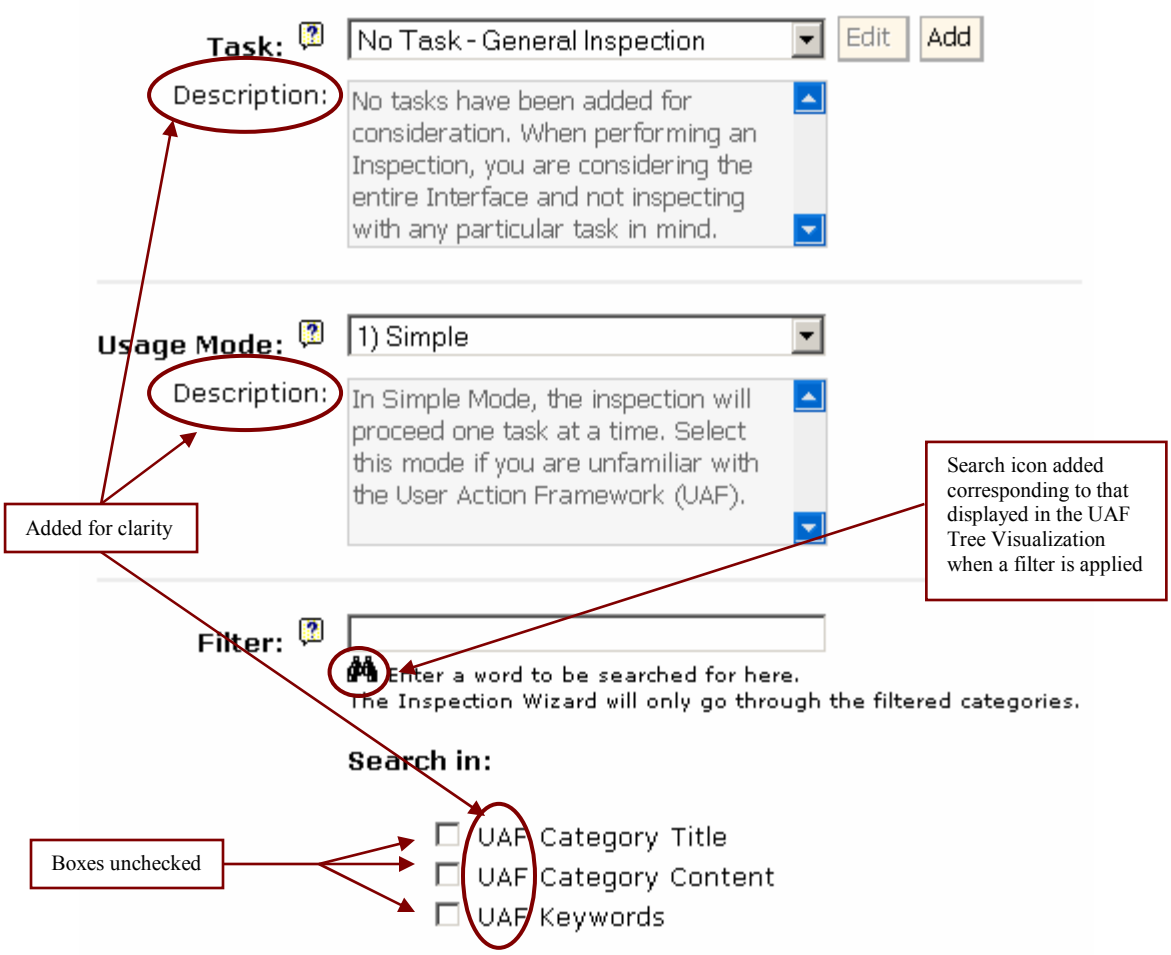


Figure 20: Setup Inspection Parameters page modifications.

3.4.4. Report confirmation page

When inspectors reported a problem, they had the option of reporting the same problem for other tasks in their task list. However, they would get confused since the task list no longer showed the task for which the problem has just been reported. This has been changed so that the task list also shows tasks in which problems have been reported with their checkboxes checked and disabled. Also, to make it clearer, the question has been restated with the 'Report' button moved closer to the task list (refer to figure 21 on the next page).

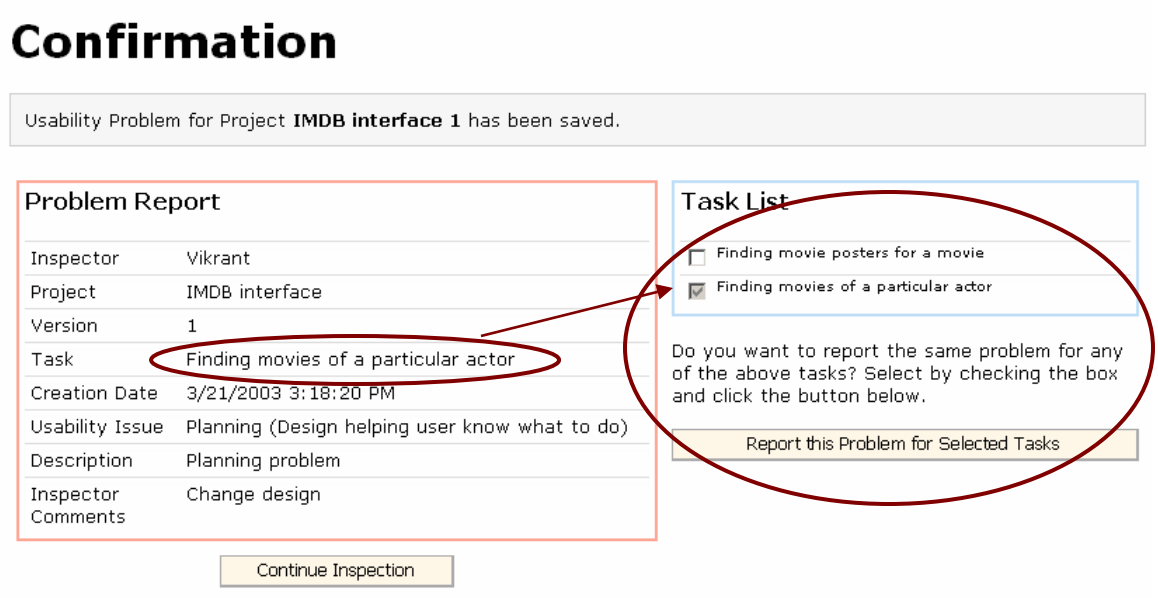


Figure 21: Task list on the Confirmation page was modified

4. Conclusion

The Usability Problem Inspection tool has been designed as an alternative usability inspection method. A formative evaluation of the Usability Problem Inspection tool was performed to identify usability problems with it. A cost-importance analysis was performed for these problems and after determining priority on the basis of cost to fix and importance of the problem, 15 problems were fixed out of 33.

Inspectors were comfortable with the tool and liked it. While a summative evaluation (which is beyond the scope of this thesis) was not done to judge the quality of problems detected by the inspectors in the sample interface (IMDB website), inspectors were able to identify, classify and report expected problems.

Revisiting the goals of this thesis:

Goals	*Accomplishments*
Tool-support for usability inspection that builds upon an underlying User Action Framework.	Designed, implemented and formatively evaluated a usability problem inspection tool that uses the User Action Framework as an underlying knowledge base.
Structured (guided) inspection instances.	The Usability Problem Inspection tool features structured, guided inspections with the help of the Inspection Wizard and the UAF.
Flexibility by customization of inspection instances.	The tool allows inspectors to customize the inspection process according to their experience level.
Cost-effectiveness through focused task-based inspection instances.	The tool provides focused inspection instances via filters and abstraction which require fewer inspection questions, therefore clearly making the

	inspection more cost-effective than an unfocused inspection demanding significantly less time on the part of the usability engineer to meet the specific inspection needs of a given instance.

5. Future work

5.1. Abstraction Development

Sometimes, inspectors may not wish to perform an inspection beyond a certain level of detail. For example, in a quick cursory examination of an interface, especially an early-stage prototype in which details are not well developed, it may not be necessary to inspect with regard to every usability issue in the UAF. Higher-level categories of the UAF hierarchy will suffice, most likely.

Inspectors should be able to specify different levels of abstraction that prevent the Inspection Wizard from navigating through lower leaves including terminal nodes of the UAF tree hierarchy. At other times, they may be interested in a low-level abstraction (or high-detail) of the UAF in which the Inspection Wizard navigates through every sub-category. The choice of the depth or detail to which the inspection proceeds is an interesting feature for the Usability Problem Inspection tool and will provide for enhanced customization and flexibility in an inspection.

There are several ways in which this abstraction may be achieved. The simplest method with least flexibility would allow an inspector to specify a depth-level. For instance, he would be able to specify that he only wants to search to a maximum of two levels deep within the UAF. However, this kind of abstraction based on arbitrary tree depth may not be very meaningful or match the inspector's needs.

A solution to this would be to develop customized abstraction structures or templates. For example – the 'Early Prototype' template would eliminate the UAF nodes that deal with font color and size from consideration in the usability inspection, thus preventing the Inspection Wizard from traversing through these nodes. Other templates could be 'Interface Sketch', 'Complete Inspection', etc. Inspectors should also allowed to create their own templates by manually pruning the tree or by pruning an existing template, adding nodes or cutting off nodes that they not interested in before starting the inspection.

Once a template is created, it can be used for any future inspection. Currently, although the interface has been developed for the abstraction concept, it has not been implemented.

5.2. Importing tasks from other inspection instances

In the case of the project being the same but the version being different, the inspector would most likely want to use the same or a similar set of tasks. Hence, it is desirable to view and if desired import tasks from other projects or other versions of the same project. Also, in an environment where multiple inspectors are testing the same user interaction design, importing of tasks created by other inspectors will be helpful. This could be done by expanding the functionality of the inspection parameter selection page. When adding tasks to an inspection, inspectors could be given the option to select from already existing tasks in the database. Most likely, only tasks from other versions of the same project would be pertinent hence perhaps only this subset of tasks should be shown to the inspector to choose from.

5.3. Reporting *positive* usability issues

An often neglected aspect of usability inspection reports is the reporting of positive findings [Redish, Bias, Bailey, Molich, Dumas, Spool, 2002]. Although, the purpose of inspection is largely to find usability problems, positive reports are encouraging to the developers and also indicative of design aspects that can be applied elsewhere. Rather than discard prototypes totally, the reporting of good usability design will indicate to designers which parts of the design to retain in later versions or in other user interaction designs.

The UAF is deliberately neutral in language and specifies areas where there may be usability *issues*. Hence, the Usability Problem Inspection tool can also be used to report positive aspects with little alteration.

For example, currently the button label text of the tool asks the inspector to ‘Report Problems’. This could be changed to a more neutral ‘Report Usability Issue’. At the point of form filling, the inspector could be asked to classify whether the issue is a problem or a positive design aspect that needs to be retained.

5.4. Viewing and modifying Reports

The current version of the Usability Problem Inspection tool does not have a way of viewing the usability problems reported. These problems are currently stored in a shared database so that other tools of the UAF suite may also use them. However, a report-viewing and modifying interface needs to be built for the Usability Problem Inspection tool too. This is itself a separate project that requires a separate user interface. In a real-world scenario, report viewing and analysis is very important.

5.5. Advanced mode

Currently, in the advanced usage mode of an inspection, *all* associated tasks are selected as context for inspection by default. Instead, inspectors should be allowed to select which tasks they want to use for the advanced multiple-task inspection. This could be very useful when there are a large number of tasks. Inspectors may wish to group them in some way and perform an advanced-mode inspection with this subset as context.

This can be approached as follows. Inspectors get a high level drop-down in which they select the usage mode. If the usage mode is simple, they have to then select a single task. If the usage mode is advanced, they get all tasks associated with the inspection as a list of checkboxes so that they can determine which ones they want to consider for the advanced inspection.

5.6. Disabling Browser '*Back*' button and providing alternative navigation

Many of the users during the evaluation study, found it difficult to develop a mental model of the way the Usability Problem Inspection tool was working. They could not figure out that the 'No Problem' button would cause a lateral jump in the tree while the pinpoint and keep inspecting would go down the tree. Also, there was no way to go back in the inspection, without clicking on the browser 'Back' button, which often caused a software bug.

On analysis, I feel it would be nice to supplement the buttons with navigation arrows like ► for 'No problem' and ▼ for the 'Keep inspecting' and 'Pinpoint' buttons. Also, similar buttons could be used in a separate navigation bar to give inspectors further control over the inspection.

5.7. Enhanced filtering

The filter action is currently very basic. In addition to 'whole word' matches, it also returns partial matches (for example, *representation* for a filter on *presentation*) to the filter result set. This can be confusing to users, sometimes returning undesired nodes. Another way to enhance the filter would be highlighting the filter word on the page that might help decrease confusion and time spent thus aiding in cost-effectiveness.

5.8. Enhanced Design for the UAF Tree Visualization

To some of the inspectors, the representations of the ✓ icon (*no usability problems*) and the ! icon (*problem reported*) in the UAF Tree Visualization were not immediately apparent. Although tool tips do exist for these, it might be better to incorporate this help more intuitively in the interface itself. For example, the ✓ icon appears as a consequence of clicking the 'No Problem' button, while the ! button appears as a result of clicking the

‘Report Problem’ button. Hence, these icons could perhaps be added on to the button labels, which would provide clear indication of their meaning. Alternatively, a legend may be required in the UAF Tree Visualization.

References

Andre, T. S. (2000). Determining the effectiveness of the usability problem inspector: a theory-based model and tool for finding usability problems. Unpublished dissertation, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Andre, T. S., Hartson, H. R., & Williges, R. C. (2002). Determining the effectiveness of the usability problem inspector: A theory-based model and tool for finding usability problems. *Human Factors, accepted per revision*.

Desurvire, H. W. (1994). Faster, Cheaper! Are usability inspection methods as effective as empirical testing? In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp.173-202). New York: John Wiley & Sons.

Doubleday, A., Ryan, M., Springett, M., & Sutcliffe, A. (1997). A comparison of usability techniques for evaluating design. In *Designing Interactive Systems (DIS '97) Conference Proceedings (pp. 101-110),* New York: ACM Press.

Hartson, H. R., Andre, T. S., Williges, R. C., & van Rens, L. (1999). The User Action Framework: A theory-based foundation for inspection and classification of usability problems. In H. Bullinger & J. Ziegler (Eds.), *Human-computer interaction: Ergonomics and user interfaces (Proceedings of the 8th International Conference on Human-Computer Interaction, HCI International '99)* (Vol. 1, pp. 1058-1062). Mahway, NJ: Lawrence Erlbaum Associates.

Hix, D., & Hartson, H. R. (1993). *Developing user interfaces: Ensuring usability through product & process*. New York: John Wiley & Sons, Inc.

Jeffries, R., Miller, J. R., Wharton, C., & Uyeda, K. M. (1991). User interface evaluation in the real world: A comparison of four techniques. In *Proceedings of the CHI '91 Conference Proceedings*, ACM Press: New Orleans, LA, 119-124.

Lewis, C., Polson, P., Wharton, C., & Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In

Proceedings of the CHI '90 Conference Proceedings, ACM Press: Seattle, WA, 235-242.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In *Proceedings of the CHI '92 Conference Proceedings*, ACM Press: Monterey, CA, 373-380.

Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 25-62). New York: John Wiley & Sons.

Nielsen, J., & Mack, R. L. (Eds.). (1994). *Usability Inspection Methods*. New York: John Wiley & Sons.

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the CHI '90 Conference Proceedings*, ACM Press: Seattle, WA, 249-256.

Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 31-61). Hillsdale, NJ: Lawrence Erlbaum Associates.

Redish, J., Bias, R., Bailey, R., Molich, R., Dumas, J., Spool, J., (2002). Usability in Practice: Formative Usability Evaluations – Evolution and Revolution. In *CHI '02 extended abstracts on Human factors in computer systems*, ACM Press: New York, NY, 885-890.

Rowley, D. E., & Rhoades, D. G. (1992). The cognitive jogthrough: A fast-paced user interface evaluation procedure. In *CHI '92 Conference Proceedings (pp. 389-395),* New York: ACM Press.

Sears, A. (1997). Heuristic walkthroughs: Finding the problems without the noise. *International Journal of Human-Computer Interaction*, 9(3), 213-234.

Wharton, C., Bradford, J., Jeffries, R., & Franzke, M. (1992). Applying cognitive walkthroughs to more complex user interfaces: Experiences, issues, and recommendations. In *Proceedings of the CHI '92 Conference Proceedings*, ACM Press: Monterey, CA, 381-388.

Wharton, C., Rieman, J., Lewis, C., & Polson, P. (1994). The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. L. Mack (Eds.), *Usability inspection methods* (pp. 105-140). New York: John Wiley & Sons.

Appendix A – Benchmark tasks of the study

Inspectors were given a set of three benchmark tasks for the study, along with a set of instructions.

General guidelines

Please remember that the purpose of these tasks is not to test your skills or knowledge but rather to evaluate the Usability Problem Inspection tool. As a result, the focus of this study is not whether you can effectively identify usability problems of the interface but rather, the tests will help determine if there are any usability problems with the Usability Problem Inspection tool and whether the tool is effective enough to help find and report usability issues in *any* interface.

As you perform any of the tasks, please try and SPEAK ALOUD the thoughts behind your actions. Anything you say will help the evaluator understand usability problems that the Usability Problem Inspection tool may have. For example, you may experience confusion at a place, which may most likely be because of a usability problem. If you speak aloud your confusion, the evaluator will find it easier to note down this problem.

When you inspect for a generic user task, you may encounter usability problems with the interface. You may take down notes if you wish. You may re-perform a user task or any stages of it or 'play' with other similar user tasks. Feel free to play around with the interface to gain familiarity.

The user tasks suggested for *an Inspection* are such that hopefully you will use all the functionality of the Usability Problem Inspection tool. However, you may at any time, choose your own user tasks or similar user tasks. You need not adhere rigidly to the example user tasks given.

At any time, the evaluator may either prompt you to report a problem, or skip a category. Again, this will be because of the evaluator's wish to test a particular function of the Usability Problem Inspection tool, NOT to correct your Inspection.

Please do not start until the Evaluator asks you to. Your Login is of the following format:
Login: **Inspector<*UserID*>**
Password: **inspection**

Eg. If your UserID is #1, your login is 'Inspector1'.

Task 1:

The purpose of this task is to test the Usability Problem Inspection tool in 'Simple' Usage mode. You are to perform an inspection of the 'Internet Movie Database (IMDB) – http://www.imdb.com' website. Your task is to use the Usability Problem Inspection tool to find and report usability problems with the IMDB site.

Please note that a thorough Usability Inspection could take several hours. Appropriate benchmark user tasks would need to be devised to test all functionality of the interface. Since the focus of this experiment is to test the Usability Problem Inspection tool rather than the IMDB site, you may consider a simple generic user task of *finding all movies of a particular foreign language of a particular genre that released during a particular year*. In particular, you may consider the following user task – "Finding French *mystery* movies that released during years 1966-1967".

The idea is not necessarily that you have to do this task, but it's just something to keep in mind while you do your usability inspection with the Usability Problem Inspector tool, in case that might help you focus.

Create a new Inspection instance and perform an inspection in *'Simple'* usage mode with NO *'Filter'* or *'Abstraction'*.

Task 2:

The purpose of this task is to help test the Usability Problem Inspection tool in 'Advanced' usage mode. While 'Simple' usage mode allows the Inspector (YOU) to consider one user task at a time while looking for usability problems, the 'Advanced' mode will allow the Inspector to consider many user tasks at a time.

Again, you are to perform an Inspection for the IMDB site. This time you will consider many user tasks at a time.

Please perform the Inspection with at least 2 user tasks in consideration. You may use the following 2 user tasks and also add any of your own for consideration during the Inspection. Again, you can use these tasks in any way you like to help you with usability inspection, and you do not necessarily have to perform these tasks.

Task 1: Find all movies that a particular actor has acted in perhaps for a particular year. (For example - "Find movies that Al Pacino has acted in for the years 1995-2000")

Task 2: Find movie posters for a particular movie. (For example – "Find movie posters for the movie 'Insomnia' ")

Start off by creating a new Inspection. Add **at least** 2 tasks to the Inspection and select *'Advanced'* usage mode with NO *'Filter'* or *'Abstraction'*. Perform the inspection with this *limited* task set in consideration. (An exhaustive set of tasks may take too long).

Task 3:

The purpose of this task is to test the functionality of the *'Filter'* in the Usability Problem Inspection tool. Sometimes, the Inspector may wish to perform a small Usability

Inspection that only focuses on a subset of problems. In those cases, he can apply a filter so that the Inspection Wizard only returns categories that are relevant to his subset.

For the current task, you may assume that you are interested in checking affordances of Buttons/Labels on the IMDB main page (http://www.imdb.com).

Create a new Inspection instance and apply the Search Filter for the keyword ‘affordance’. You may ‘invent’ a simple task to do this or perform a ‘General Inspection’. Find and report problems (if any).

Note: When a Filter is applied for an Inspection, the Inspection Wizard will only search through categories that relate to the **filter word**. Other categories will be **skipped** by the Inspection Wizard.

Appendix B – Feedback form distributed to inspectors for the study

Feedback form for the Usability Problem Inspection tool

How would you rate your usability knowledge, on a scale of 1-5?

Scale:	1 None	2	3 Good	4	5 Excellent

How would you rate your knowledge of the User Action Framework (UAF) on a scale of 1-5?

Scale:	1 None	2	3 Good	4	5 Excellent

Have you taken the 'Usability Engineering' course (CS5714)?

☐ Yes ☐ No

Have you ever participated in a usability evaluation of software?

☐ Yes ☐ No

Did you have any problems with the Usability Problem Inspection tool?

Do you have any positive/negative comments about the Usability Problem Inspection tool?

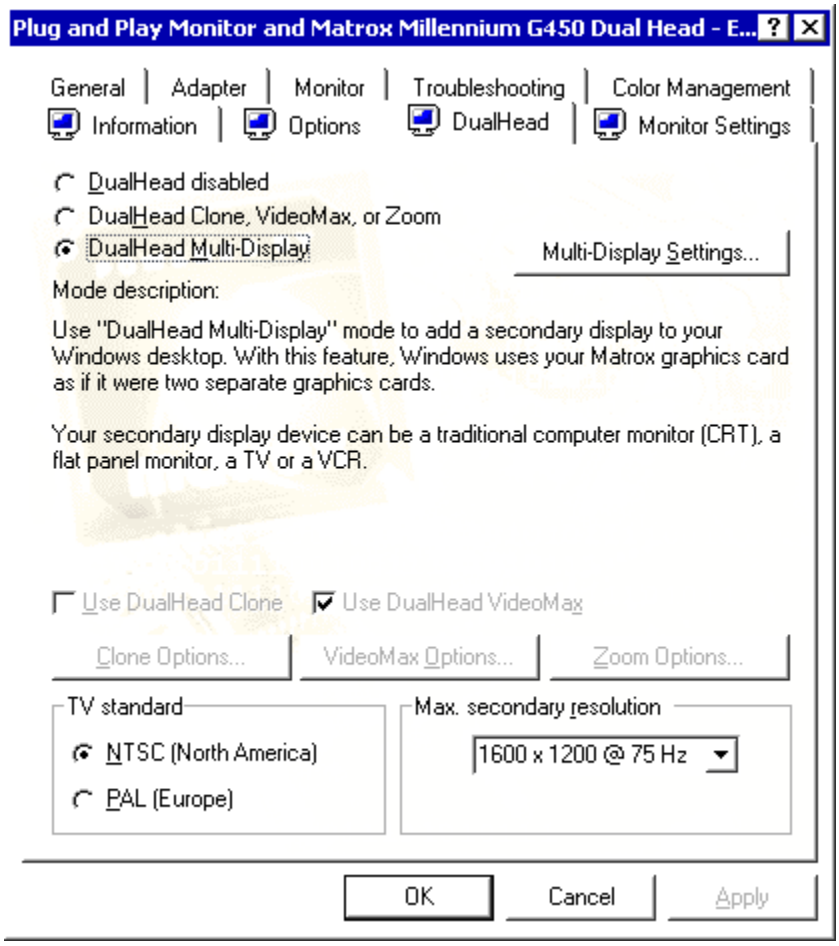
Were you comfortable with the experimental setup? Did you have any problems? Suggestions?

Appendix C – Equipment Manual for Laboratory testing

This equipment manual was written by Colaso Vikrant, Mahajan Reenal & Narayanan Kodiyalam.

Dual Monitor Setup

Enabling Dual-Monitor display	**Disabling Dual-Monitor Display**
Start> Settings> Control Panel> Display> Settings> Advanced> Dual-Head Click on *Dual Monitor MultiDisplay*	Start> Settings> Control Panel> Display> Settings> Advanced> Dual-Head Click on *Dual Monitor Disabled*



VNC (Virtual Network Computing) Setup

VNC allows the Evaluator sitting in room McBryde 102 B to view and *manipulate* the 'Desktop' or 'Screen' view of the Participant sitting in room McBryde 102 A. This way, the Evaluator can remotely and non-intrusively observe the Participant's screen. The Evaluator can also *take control* of the Participant's computing environment by manipulating the Participant's mouse and cursor.

1. Run WinVNC(App Mode) in the machine in 102 A. (This is the server)

Start>Programs>VNC>Run WinVNC(App Mode)

This will start the WinVNC server and produce an icon in the system tray

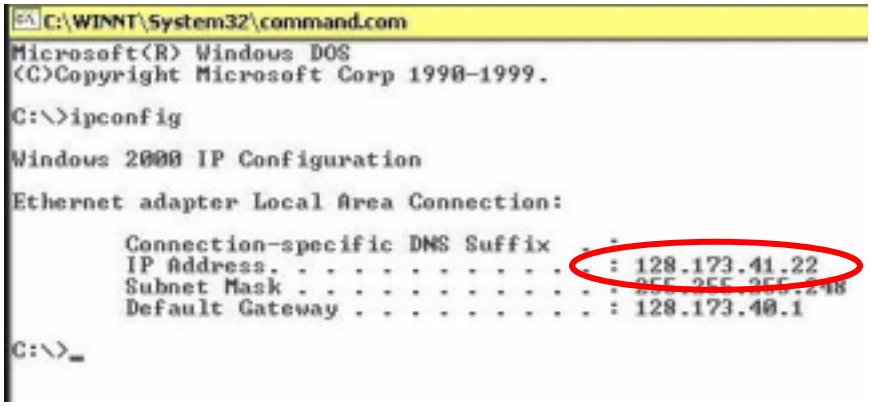
2. Run VNCViewer in Listen mode in 102 B. (This is the viewer)

Start>Programs>VNC>Run VNCViewer(Listen Mode)

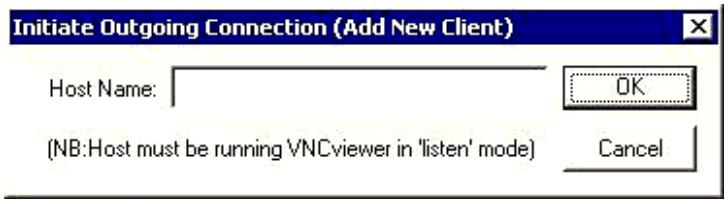
This will start the WinVNC Viewer and produce an icon in the system tray

The order in which you start the Viewer and Server does not matter.

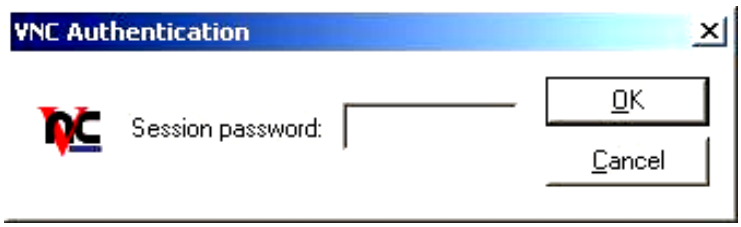
3
. In the machine in **102 A:** *Start>Run>Type "command"*. This will open the Command Prompt. Now *Type "ipconfig"*. Note down the IP Address. Then type *"exit"* to close the Command Prompt.



4. Now Double-click on the VNCViewer icon in the system tray of the machine in **102B** and enter the IP Address of the machine in 102 A.



5. The VNCViewer now asks you the password of the machine in 102 A so that it can connect to it. The password is currently set to "********".



Alternatively, you can also directly grant access to the Evaluator machine in 102 B from the machine in 102 A. For this, the VNC Listener should already be running in 102 B. Note down the IP Address of the machine in 102 B in a similar way as above. Then right-click on the VNC icon in the system tray of the machine in 102 A and select "Add a New Client". Enter the IP address of the Evaluator machine (102 B machine) here.

Microphone Setup

The Microphone Setup in 102 A allows a dual sound stream to be recorded. One sound input comes from the participant microphone in 102 A and the other from the evaluator microphone in 102 B. In the event that either of the microphones are not working, follow the instructions given below :

1. The Microphone Preamplifier (dbx 760x) in 102 B should be connected to the power supply.

   The Microphone Preamplifier has two sets of controls. The controls on the left are for the evaluator and the duplicate set of controls on the right are for the participant.

2. The evaluator's microphone in 102 B should be connected to the balanced input on the left in the Microphone Preamplifier.

3. The participant's microphone in 102 A should be connected to the balanced input on the right in the Microphone Preamplifier.

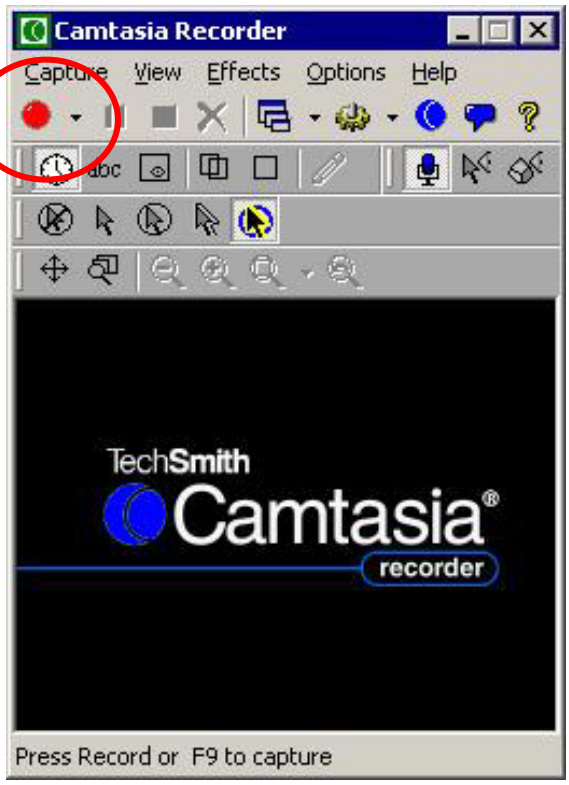
4. The unbalanced outputs from the evaluator and the participant should be connected to a connector which is then connected to the microphone input on the evaluator's machine in 102 B.

5. Adjust the volume levels on the Microphone Preamplifier.

Recording using Camtasia

1. *Start> Programs> Camtasia> Camtasia Recorder*

2. Press the record button.



3. When done, press the square STOP button.

4. Save the file. The file will be saved with a "*.avi*" extension.

Playback

The created file can be best played with "Real Player" which allows the Evaluator to scan back and forth through the file. Alternatively, it can be played on Windows Media Player or the Camtasia Player.

Communicating with the Participant in 102 A from 102 B

Use the Intercom to communicate one-way with the participant in 102 B. The evaluator can hear the participant in 102 A through the speakers in 102 B. Hence if the evaluator wishes to speak to the participant he can use the Intercom. This can be turned on by pressing the 'TALK' button. If the evaluator wishes to keep the communication on continuously, he can do so by pressing the TALK and LOCK buttons simultaneously.

Recording the VNC Listener view using Camtasia.

After starting the VNC viewer, the evaluator will have a view of the Participant's screen. He can then start Camtasia and by capturing the VNC Listener Window a screen capture video can be made. With both Microphones on, the video will record both the evaluator and participant voices simultaneously.

Appendix D – Usability Problem Sheet

Prob#	Task# where problem occurred	Description of Problem & Effects	Suggestions/Remedy	Interface Location	Freq	Inspector #						
						1	2	3	4	5	6	7
1	1	Date format entry problem. Lost the format after clicking in the input box and could not remember what it was. Also, found this entry unnecessary, as the system should derive it. No format checking.	Keep format always visible or auto-enter it.	Add project	4		✓	✓	✓		✓	
2	1	"Name" label unclear. Inspectors entered their own names instead of "Project Name".	Change label to 'Project Name'	Add project	2		✓		✓			
3	2	Version numbers should be unique, new version should not allow for creation of an already existing version.		Add version	1				✓			
4	1	Confused by the report because an extra '1>' appeared in the task name.	Do not produce numbering in simple mode.	Confirmation page (Simple mode)	1				✓			
5	2	Problem understanding why the Task List was produced. Felt that even tasks for which a problem has just been reported should also be shown else users might believe their reporting was unsuccessful.	Show *disabled* ticked checkboxes for tasks that have just been reported.	Confirmation page.	1	✓						
6	All	Unsure what the difference between 'Pinpoint' and 'Report' buttons is. Did not know how to proceed, which button to click.	Change pinpoint to 'Search this category', Provide help.	Inspection Wizard	6	✓	✓	✓		✓	✓	✓
7	All	No way to terminate the inspection.	Need a 'Inspection over' button.	Inspection Wizard	4	✓			✓		✓	✓
8	1	Difficult to associate which buttons correspond to 'Unsure', 'Yes' and 'No'	Try putting them in a box.	Inspection Wizard	1		✓					
9	2	Mental model of how 'pinpoint' works does not conform to developer's mental model. Cannot predict where the Wizard will go next.	Warning should be provided before jumping from a leaf node.	Inspection Wizard	3	✓		✓			✓	

10	2	Difficult to quickly switch to ‘Simple’ mode from ‘Advanced’ mode. Difficult to change to another task from current task.	There should be a way to quickly change the task and mode.	Inspection Wizard	2		✓		✓			
11	2	Confusion over the selecting of tasks. Thought that you had to select tasks even when pinpointing.	Associate the ‘Report’ button with the task list. If no tasks are selected, disable the report button.	Inspection Wizard (Advanced mode)	7	✓	✓	✓	✓	✓	✓	✓
12	2	Task list box was confusing; inspector did not notice it at first.		Inspection Wizard (Advanced mode)	2	✓				✓		
13	3	Make it clearer when a filter is active.	Highlight the ‘filter keyword’, show keyword in the header	Inspection Wizard, Tree Visualization	1					✓		
14	All	Inspector resized the UAF Tree frame generating more room for the tree.		Inspection Wizard, Tree Visualization	2	✓	✓					
15	1, 2, 3	Input boxes (gray colored) look disabled.	Convert to standard type.	Overall	2		✓		✓			
16	2	Input boxes too small to write detailed descriptions.	Increase the size of the boxes.	Report filling page	1		✓					
17	1, 2	Already checked checkboxes for the ‘filter’ made it seem like a filter was active. Inspectors unchecked the boxes to disable the filter.	Do not check the boxes if no filter text exists.	Setup Inspection	2		✓					✓
18	All	Filter data lost on clicking in the ‘Filter’ input box	Do not erase information typed here.	Setup Inspection	1							✓
19	All	Not clear what the ‘description’ box below the ‘select task’ is.	Add label ‘Description’	Setup Inspection	3					✓	✓	✓
20	All	Newly added ‘task’ should automatically get selected.		Setup Inspection	1					✓		
21	2	On selecting ‘Advanced mode’, inspector is confused which task to select. What if the inspector only wishes to select some tasks for an Advanced Inspection?	Tell the user explicitly that all tasks will be considered.	Setup Inspection	2	✓			✓			
22	All	Previous selections are lost on refreshing. Defaults get restored.		Setup Inspection	3			✓	✓		✓	
23	All	Message in red seemed like an ‘Error’ message		Setup Inspection	1				✓			
24	3	Confused with what *Category Title, Category Description* and *Keyword* mean exactly.	Prefix them with *UAF*	Setup Inspection (filter)	1					✓		

25	3	Confused with the Filter icon.	Show the icon on the page where the filter is applied.	Tree Visualization	3	✓	✓			✓	
26	All	Faster tool tips required		Tree Visualization	3	✓			✓		✓	
27	All	Confused with the green ‘no usability problem’ tick mark.	Show the icon near the ‘Skip category’.	Tree Visualization	3	✓			✓			✓
28	All	UAF Tree does not refresh (or hide) when current inspection is abandoned and a new one begins.		Tree Visualization	2		✓		✓			
29	2	Difficult to find previously reported problems when the tree collapses automatically.		Tree Visualization	1		✓					
30	1	Clicked on ‘Continue Inspection’ instead of ‘Create New Inspection’.	Label as ‘Continue *Previous* Inspection’	Welcome page	4	✓	✓			✓	✓
31	1	Definitions of Project, Version missing, unclear.	Update the definitions.	Welcome page	2		✓		✓			
32	1	On adding a new 'Project' or 'Version', the newly added one should be selected in the main page. Inspector is confused and thinks he is unsuccessful in adding it.		Welcome page	4	✓			✓	✓		✓
33	1	Failure to figure out that buttons were disabled, active ‘Add’ button missed among so many disabled buttons, tried clicking on disabled buttons.	Make ‘Add’ more noticeable, provide a prompt.	Welcome page	2				✓			✓

Appendix E – Cost-Importance Analysis of usability problems

Problems are cited here by the 'Problem#' used in Appendix D.

Fixed?	Prob#	Freq	Importance to fix	Cost to fix in hours	Importance/Cost * 1000	Priority to fix
✓	1	4	4	0.25	16000	1
✓	16	1	3	0.25	12000	2
✓	30	4	3	0.25	12000	3
✓	2	2	2	0.25	8000	4
✓	17	2	2	0.25	8000	5
✓	18	1	2	0.25	8000	6
✓	24	1	2	0.25	8000	7
✓	31	2	2	0.25	8000	8
✓	5	1	3	0.5	6000	9
✓	19	3	1	0.25	4000	10
✓	25	3	1	0.25	4000	11
	6	6	5	2.5	2000	12
✓	15	2	1	0.5	2000	13
	7	4	4	2.5	1600	14
✓	20	1	3	2	1500	15
	26	3	3	2	1500	16
✓	32	4	3	2	1500	17
	11	7	4	3	1333	18
	13	1	3	2.5	1200	19
	21	2	5	4.5	1111	20
	3	1	1	1	1000	21
✓	4	1	1	1	1000	22
	8	1	1	1	1000	23
	9	3	3	3	1000	24
	27	3	3	3	1000	25
	33	2	3	3	1000	26
	22	3	2	3	667	27
	10	2	2	4	500	28
	28	2	2	4	500	29
	12	2	1	2.5	400	30
	14	2	1	2.5	400	31
	23	1	1	4	250	32
	29	1	1	7.5	133	33

Vita

Vikrant Colaso was born on 18th Dec, 1979 in Bombay, India. He graduated from Fr. Conceicao Rodrigues College of Engineering, Mumbai (Bombay) University in 2001 with a Bachelor's degree in Computer Engineering.

He then pursued a Masters degree in Computer Science at Virginia Tech. During this two year academic phase, he worked as a teaching assistant for Virginia Tech. He started work at Freddie Mac, McLean, Virginia in April, 2003 as a Programmer Analyst. He completed and defended his thesis while still working at Freddie Mac.