Structured Inspections of Search Interfaces: A Practitioners Guide

Max L. Wilson, m.c. schraefel
School of Electronics and Computer Science
University of Southampton
Southampton, UK
{mlw05r, mc}@ecs.soton.ac.uk

ABSTRACT
In this paper we present a practitioners guide on how to apply a new inspection framework that evaluates search interfaces for their support of different searcher types. Vast amounts of money are being invested into search, and so it is becoming increasingly important to identify problems in design early, while it is relatively cheap to rectify them. The inspection method presented here can be applied quickly to early prototypes, as well as existing systems, and goes beyond other inspection methods, like Cognitive Walkthroughs, to produces rich analyses, including the support provided for different search tactics and user types. The guide is presented as a detailed example, assessing a previously unevaluated search interface: the Tabulator, and so also provides design recommendations for improving it. We conclude with a summary of the benefits of the evaluation framework, and discuss our plans for future enhancements.

Author Keywords
Search, seeking, evaluation, inspection, prototype, design.

ACM Classification Keywords
H5.2 User Interfaces: Evaluation/Methodology, Prototyping. H3.3 Information Search and Retrieval: Search Process.

INTRODUCTION

Search pervades our digital environments. We search for files on our computers, numbers in our phones, resources on corporate intranets, information on the web, and products on websites, to name a few. Yet it has been shown that a number of scenarios, including vertical search (within-website search, such as for products at Walmart.com), would benefit from more interactive search methods that support alternative means to the simple keyword search model [16]. Even Google has added facets such as price and brand as filters to their product search[1]. While both industry and academia are producing novel means of supporting users in browsing, comparing, and evaluating information, the challenge remains in selecting the best features for an effective search interface.

As vast amounts of money are being invested into search, and it is widely accepted that the sooner usability problems are found, the quicker and cheaper they are to rectify, there is a clear demand for understanding the strengths and weaknesses of designs before too many resources are invested in developing them. While early design evaluations, such as Heuristic Evaluation and Cognitive Walkthroughs, have been shown to discover a high number of general usability problems, our own research has focused on evaluating search interfaces particularly for their support of known information seeking tactics. Using a similar methodology to Cognitive Walkthroughs, this inspection framework encourages evaluators to systematically question a user's ability to carry out 32 different tactics with each search feature of an interface. Unlike Cognitive Walkthroughs, however, the process produces three simple graphs that represent: 1) the amount of support for search provided by each feature, 2) the amount of support provided for each search tactic, and 3) the amount of support provided for different types of users, such as experts with clear goals and unsure novices.

While previous publications explain how this framework has been generated from of two models of information seeking, and validated [17], the aim of this paper is to provide a practitioners guide to applying the framework, in the same vein as the guides produced for Cognitive Walkthroughs [15] and GOMS [6]. Combined with online materials, described in the discussion section below, this guide will support evaluators working on new search software in understanding how new designs differ in their support for information seeking. Like the Streamlined Cognitive Walkthrough, presented by Spencer [11], the framework has a fixed scope and so is fast to apply, identifying weaknesses in support for different search tactics and different user types in a matter of hours. Among

[1] http://www.google.com/products - Google Product Search

the benefits of the approach is in being able to see for whom the interface might be difficult to use, and with the other graphs, where the interface may be poorly designed to cause any lack of support. Further, when applied to multiple designs, the analyses allow a direct comparison of their strengths and weaknesses.

In the paper below, we first present related work focusing on evaluation during the early phases the design. This is then followed by a brief overview of the inspection-based evaluation framework to highlight its unique benefits against other optional methods. Following this overview, the majority of the paper focuses on a detailed example of applying the framework to a search interface. As stated above, the focus of this paper is a practitioner's guide on how to apply the framework. The example application focuses on a new browser for linked data: the Tabulator, which has yet to receive usability evaluation [3]. A further contribution of the paper, therefore, is to present the analysis of the Tabulator, but the reason of presenting the analysis is as a guide on how to use the results produced by the framework. We conclude with design recommendations for the Tabulator, discuss the future developments planned for the framework, and recount the benefits of the approach.

RELATED WORK

It is widely accepted that usability assessments should be part of the entire design process, starting even before requirements are gathered [14], with techniques such as Ethnographic observation [4]. It has been known for some time now that large amounts of resources can be saved by finding usability problems towards the start of the development cycle. Bosert, for example, showed that early evaluation of usability can reduce the development life-cycle by up to 50% [5]. Further, Lederer and Pressad [9] showed that, while 63% of development projects over-run, the top four reasons were all related to unforeseen usability issues. Consequently, a number of techniques have been developed that allow early prototype designs of software to be assessed for their usability. While some methods, such as rapid low-level prototyping [12], allow early designs to be evaluated by participants, Cognitive Walkthroughs [15] and Heuristic Evaluations [10] allow interfaces to be evaluated by the designers alone.

Cognitive Walkthroughs [15] allow evaluators to systematically step through example scenarios of use, usually provided by Hierarchical Task Analysis [7], asking with each action four simple questions including: does the user understand that a certain function is available, and does the user receive feedback about their action. This approach has been shown identify around 80% of usability problems and can be applied to early designs, and without the need of study participants. More recently a Streamlined Cognitive Walkthrough [11] method was proposed, based upon the experiences of using the Cognitive Walkthrough method within real software development environments. The streamline method advocates, among several recommendations, focusing on key usage scenarios to save time, and for postponing the discussion of redesign until after the evaluation is complete.

While Cognitive Walkthroughs focus on the learnability of an interface, in terms of how easily a user might learn how to use it, Heuristic Evaluations focus on comparing an interface design with several recognized usability principles [10]. Although often considered to be fairly informal, the process is widely used to make sure that simple errors, based around known usability principles, do not hinder the design and development process. Heuristics include: consistency, feedback, providing short cuts, clear error messages, and using clear and natural language.

While both of these methods encourage evaluators to emulate predicted scenarios of use, they do not analyze the functionality of the interface for how it allows users to achieve their goals. Instead, these methods focus on whether the means provided to achieve goals are clearly and simply designed. One benefit of methods such as Heuristic Evaluations and Cognitive Walkthroughs, therefore, is that they are sufficiently generic that they can apply to all user interfaces.

Another method that focuses on evaluating interfaces for how they support users in achieving their goals is GOMS [6], standing for *Goals*, *Operations* (user actions), *Methods*, and *Selection rules*. The aim is to analyze software for a specific set of goals, the operations available, the methods used to achieve the goals, and the selection criteria for choosing different methods of achieving the same goal. Largely, however, this method requires working software to be evaluated, and requires a detailed understand of specific tasks and working practices. GOMS was used, for example, to estimate that a new call-center workstation was less efficient, by 3%, for the tasks carried out by the staff. The model found that the new software was too rigid and reduced staff utility of waiting time. While this sort of finding can't be achieved through Cognitive Walkthroughs, the method is very complicated, does not consider different types of users [13], and doesn't evaluate the functionality of the interface, only the usability of its functions. The framework presented here, however, is much more light-weight, can be applied at earlier stages of design, and assesses the functionality for different types of searchers.

With the intention of evaluating a more specific set of interfaces, in this case for search, we have designed an evaluation framework that can be applied in a similar way to Cognitive Walkthroughs, but is focused specifically on how usable the interface is for search. Like Cognitive Walkthroughs and Heuristic Evaluations, the framework a) can be applied by evaluators without the need for participants, b) can be done so with early prototype designs, and c) can be completed within a matter of hours, depending on the complexity and number of interfaces being evaluated. Finally, as the framework is designed for a more prescriptive set of interfaces, we have used

established models of information seeking to interpret a simple inspection approach into rich analyses. The next section provides an overview of this approach.

DESIGN OF EVALUATION FRAMEWORK

The evaluation framework promoted in this paper provides means to analyze search interfaces with three graphs by simply calculating the number of moves it takes a user to carry out known search tactics with each part of the search interfaces being evaluated. Before describing, in-depth, an example application of the framework and its analyses, we first give an overview of: the way it is applied, the way data is recorded, and the three graphs it produces for analysis. The purpose of such an overview is to provide context and understanding as we progress through the example.

Overview of Method

An overview of the process taken in applying the framework is to: count the number of actions a user must take in order to achieve each search tactic (from a fixed set of known search tactics), with each feature, of each interface in question. 'Each interface' may refer to very different interfaces that are being compared, or may refer to multiple potential designs of a single new system. 'Each feature' refers to the set of user interfaces that have been included to support the user. 'Search tactics' are known tactics that have been identified through information seeking research, which are described in more detail in previous work [18], and consist, for example, of tactics such as narrowing a search, broadening a search, saving a result, checking results found by others, and so on.

The result of applying this process is that the evaluator, who may well also be the person designing a new system, systematically asks the question 'how can the user perform this tactic with this feature', and with every tactic and with every feature of every interface being studied. Moves, as the metric being counted, are taken from Bates [1] and consist of a segmented set of both mental and physical actions, such as choosing a search term and entering it. We count the minimum set of actions required to achieve the tactic with the interface feature, as we cannot predict circumstantial or conditional cases, such as the size of a search term that will be important to the user, or how many search terms will be required. More discussion of using the minimum set of actions can be found in previous work [18].

Data Entry Table

To store and analyze the results, the move-count produced by the process method above is stored in a series of Excel tables (one per interface being studied) with tactics listed on the x-axis and features listed on the y-axis. The table can be seen in Table 1 in the analysis section. While zero moves represents no support for a tactic by a feature, the optimal support will be one move. Consequently, positive scores are inverted, so that a single move becomes the largest score. The formulae in the Excel workbook can then take summative values by both feature and tactic. Informally, these two summative values equate to the amount of support each feature provides during search, and the amount of support the interface provides for each tactic. Both are used in the generation of the graphs produced for analysis, discussed below.

Overview of Analyses

The metrics calculated above are converted into three graphs by the Excel Spreadsheet. The first graph, put simply, provides a comparison of the interface features. With a single browser being evaluated, the graph can identify strong or weak aspects of the interface. Further, when comparing multiple interfaces, the graph allows a direct comparison between the same features implemented in each, showing that, for example, interface A has a better keyword search implementation than interface B, assuming that they have different interactions rather than different retrieval engines.

The second graph produced compares the support provided for each search tactic. When studying a single interface, it shows which tactics are well or poorly supported. When comparing multiple interfaces, however, the graph allows a direct comparison between the support provided for the same tactic by each. This shows that, for example, interface A is better for narrowing search than interface B.

The third graph produced by the framework, presents a summarization of the support provided by each tactic, as the support provided for different types of users. Different types of users, according to the model used [2], are identified by their previous knowledge, goals, and familiarity with a system. The model captures 16 types of users ranging from those who can perform efficient known-item search to those who will be performing undirected and exploratory search [16]. When studying a single interface, this graph helps identify which types of users will be well or poorly supported. When comparing multiple interfaces, the graph tells us which interface is better or worse at supporting each type of user. Full detail of the mapping between user types and search tactics is described and validated in previous work [17].

EXAMPLE APPLICATION OF THE FRAMEWORK

The following example application of the framework servers two purposes. First, the detailed description of its application to a known browser provides a clear example of how to apply the framework. Second, the evaluation is of a new browser for the semantic web, which has been identified, by the authors, as having had no prior usability assessment. This evaluation seeks to confirm the view held by many that, although functional, the system is notably hard to use. In confirming this, however, the paper concludes by producing some design recommendations, which may generalize to general semantic web UI guidelines.

Step 1: For every interface

The first step of the framework's application process is to identify the browsers being evaluated. In this example application, we are describing the evaluation, in detail, of a single search interface: The Tabulator [3].

The Tabulator is a Semantic Web browser, designed to embody the interactions that have been envisioned by the future of the World Wide Web, by the Web's own creator: Tim Berners-Lee. Unlike the current web, made up primarily of web pages, the Semantic Web has been designed to be a Web of data that can be queried by either

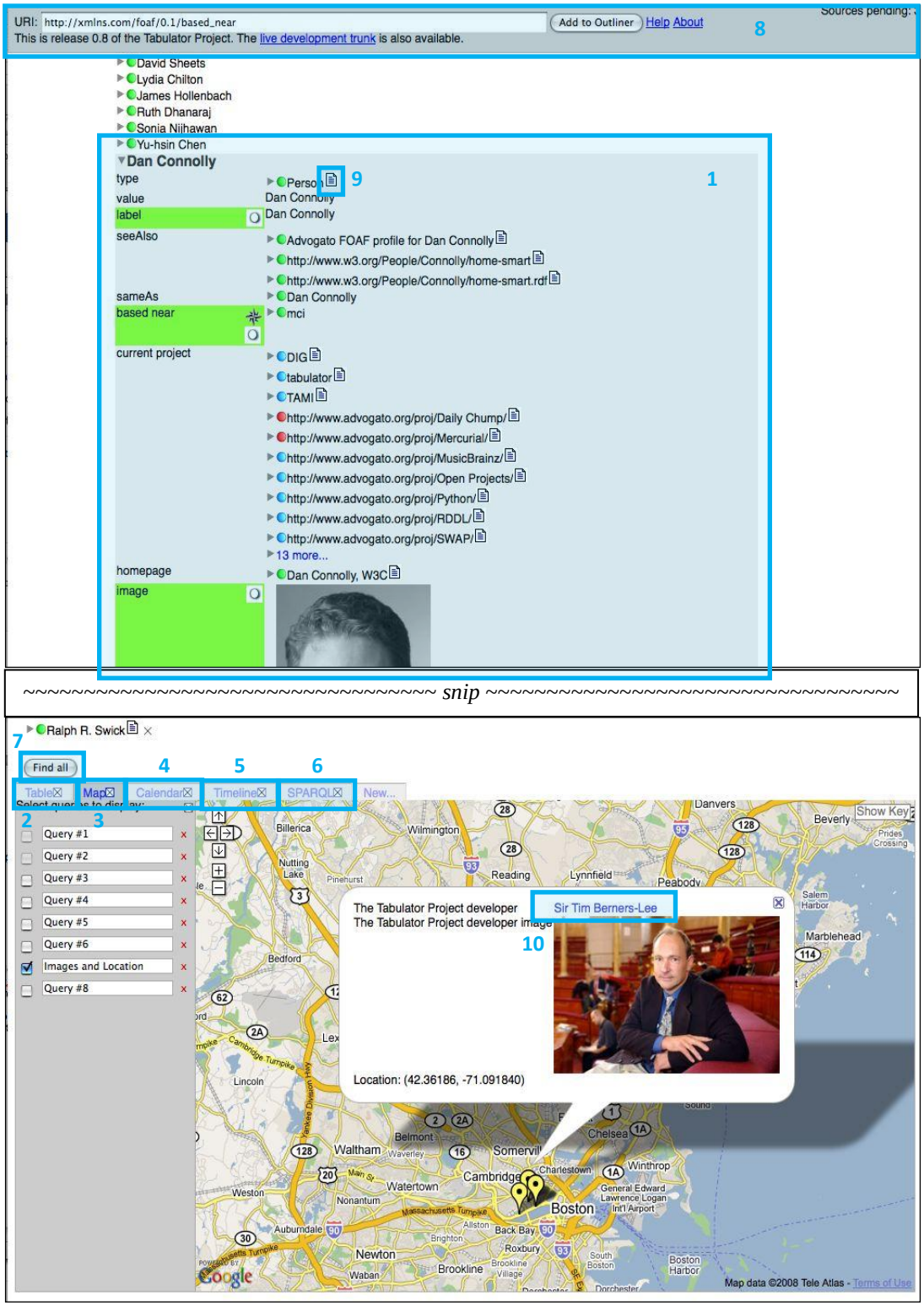

Figure 1: The Tabulator browser interface, with features highlighted in blue.

		Tactics																												
Tabulator	(Totals)	CHEC	WEIG	PATTE	CORR	RECO	BIBB	SELE	SURV	CUT	STRE	SCAF	CLEA	SPEC	EXHA	REDU	PARA	PINP	BLOC	SUPE	SUB	RELA	NEIGI	TRAC	VARY	FIX	REAR			
URI Bar	2.8702381		4	6	6	7					8	6		5							4	4	6				6	6		
Explorer	6.99761905	3	4	4	2			3	2		6	7		2	3	3	5	3			3	3	3	4	2			6	7	
RDF Popup	1.70833333			4		3	4		4		8	8													4					
Table View	3	3	6	6	6			3			6	6												2	2					
Map View	5.2	5	8	8	8			3		2	8	8			2	2			2		2	2	3	3						
Calendar View	4.7	5	8	8	8			3			8	8			2	2			2		2	2	3	3						
Timeline View	4.7	5	8	8	8			3			8	8			2	2			2		2	2	3	3						
SPARQL View	2.71803752	5	7	7	9			9		9	11	9		9	9	9	9	9		9	9	9				9	9	9		
Predicate Req	1.75			4			4				2													4					
Find All	0.5					2																							

Table 1: Input table showing the all values entered by carrying out the analysis of the Tabulator.

humans or computers in any combination as described by the data itself. That is, if the data on the Semantic Web has specific attributes and relationships, then either, and any number of them, can be used as constraints during search. The aim of search on such a web is not to find pages that contain desired information, but find the information itself. As a result of this difference, the Tabulator has also focused its design upon the ability to both find and analyze information. Consequently, the interface has two main halves, the search half, and below that, the analysis half. We now break down the Tabulator into its interface features for step 2.

Step 2: For each part of the interface

There are 8 main features of the Tabulator interface, and two less obvious feature, which have been highlighted in Figure 1. The foremost feature of the interface is the tree-based explorer (1). Using this explorer, the user can expand any one of the root nodes initially listed to see all of the attribute types associated with it, and one or more of their values (long lists are cut off and replaced with a 'more' button). The user can continue to navigate in this way as long as the values reached by expansion have further attributes to expand. As well as exploring in this way to find specific items of information, the user can also define a pattern and request, using the 'Find All' button, to see all such values. To assert such a pattern, the user can select the attributes and/or values in the explorer, so that they are highlighted in green. Alt-select allows the user to select multiple attributes or values for more complicated examples, as shown in Figure 1.

For example, a user might expand a 'developer team' node to see all of its attributes, such as its office location and its developers, and expand the details of one team member, and highlight: the name, date of birth, current living location and picture. Pressing 'Find All' will find these details for all the members of the team and pass them to the analysis features, described below. If, however, there is a team manager with these same details, he will also be found, as the user did not highlight 'developer' as a constraint. The user may add this constraint and select 'Find All' to pass the new findings to the analysis modules, as a new result set. Further, the user may decide that they want to see the whole team, regardless if they are missing either their date of birth, or home town, and may mark them as optional with the radio button seen within the green highlight.

There are 5 analysis modules available (2-6), that make up 5 separate features: the table view, the map view, the calendar view, the timeline view, and the SPARQL code view, which allows the user to directly edit a query in the SPARQL[2] language used to retrieve from the Semantic Web. The 'Find All' button passes sets of results to these views to be displayed. In the team example above, the table view would show four columns, with the team members' names, dates of birth, locations and pictures. As the query contains a location field, these can be displayed on the map view. Multiple result sets can be shown on the map view at once if required. Similarly, as the team member query above has a date field, the user can show their dates of birth in either the calendar or the timeline view, where result sets can be combined if required. The SPARQL viewer provides a query by example interface, allowing the user to edit the queries that produced existing result sets, and use them to create new queries, and thus new results sets.

The first unobvious feature of the interface is, in fact, the 'Find All' button (7), which servers to create results sets from the patterns defined in the explorer, and pass them to the analysis modules. As it is servers this separate function, and is not required to explore or to analyze, it is identified separately as a feature to be evaluated.

Another noticeable feature of the interface is the URI bar that is permanently visible at the top of the screen (8). Primarily, the URI bar is used to display the complete URI of the last item selected within the Explorer. This allows the user to both check the provenance of an item selected, and copy and save it if necessary. The URI Bar may also be used to add certain parts of the Semantic Web to the browser, as a new root node on the interface. This can be achieved by pasting a URI into the URI Bar and pressing 'Add to Outliner', where Outliner is the name used for the explorer.

The penultimate feature to identify in the Tabulator is the RDF Popup button (9). This allows the user to view the original source data, in the RDF format[3], of something found in the explorer. The final feature of the Tabulator to identify is that any item found in the analysis modules may be loaded as a new starting node in the explorer, by double

[2] http://www.w3.org/TR/rdf-sparql-query/

[3] http://www.w3.org/RDF/

clicking on it (10). So in the team member example, the user may wish to start exploring again from one particular member, or one particular location or date.

Step 3: For each tactic, count the Moves required

The final step of applying the framework requires the evaluator to count how many actions it takes for a user to carry out each known tactic, if the tactic is achievable with that feature. As it is not possible here to a) define each of the 32 tactics, as defined by Bates [1], b) explain how they can be carried out with each of the 9 interface features, we instead first define what constitutes an 'Move' and then give 5 examples of how many actions are required to achieve certain tactics with particular features.

Moves, as according to Bates [1], are segmented mental or physical actions. Familiar examples in keyword search would be: choosing a search term (mental), entering a search term (physical), pressing search (physical), scanning results (mental), choosing a result (mental), and opening it (physical). In the simple way just demonstrated, we seek to count the moves required to achieve each tactic with each of the 9 interface features. Repeated and circumstantial moves, such as scrolling, are not counted, as we cannot guarantee that they will be needed [18].

SPECIFY with the Explorer: 2 moves. Perhaps the most simple tactic-feature combination is to specify an information need with the explorer. In the simplest scenario, with circumstantial conditions such as appropriate starting point and a single constraint, only two moves are required: choose an attribute to expand (mental), and expand it (physical).

CHECK with the Explorer: 3 moves. The CHECK tactic refers to be the user correlating their current search status against their original goal. The user is able to check that their search is still on a productive path with three moves: review their selections in the explorer (characterized by green highlights or expanded lists), recount their intentions, compare the two. In this case, as a visualization of the users previous selections is produced as they explore, no physical moves are required to carry out this monitoring tactic, only the identification of two comparable items, and their comparison.

CHECK with the Table View: 3 moves. Like with the explorer, the user is able to CHECK their current state against their search goal with the Table view in 3 moves. The attributes selected to produce the result sets are clearly listed as table headers, and so the user can: view the table headers, recount their intention, and compare the two. Again, these are all mental.

CHECK with the Map View: 5 moves. Unlike the Table View and Explorer, a physical action is required before the user can check the direction of their search. As the user can only see the attributes of the result set in the information popup of a selected item on the map, the user is required to: choose an item on the map (mental), select it (physical), view the information popup (mental), recount their intentions (mental), and compare the two (mental).

RECORD with the 'Find All' button: 2 moves. Uniquely in the interface, the 'Find All' feature servers only one tactic: to record ones search for future reference. When the user presses the 'Find All' button, any matching results are stored as a result set that can be used by any of the analysis modules. It does not, itself, display any results, or allow the user to affect their search in anyway. Simply, the moves required are: identify the location of the 'Find All' button (mental) and press it (physical).

ANALYSING THE FRAMEWORKS RESULTS

We now present the analysis of the Tabulator, using the rich analyses produced by the inspection method, to a) demonstrate how the analyses can be used, and b) also present usability findings of an interface that has otherwise not been evaluated.

Graph 1: Interface Parts

Graph 1, shown in Figure 2, conveys the level of support during search that is being provided by each of the features in the interface, as defined in Step 2. This Graph alone both confirms some expectations and reveals some interesting insights. First, it is not surprising, perhaps, that the Outliner, or explorer, provides the broadest amount of support for search, compared to all the other features within the Tabulator. Second, it is probably not surprising that the different visualizations at the bottom of the interface make up the subsequently tall bars within the graph, as these provide the means to analyze the results further.

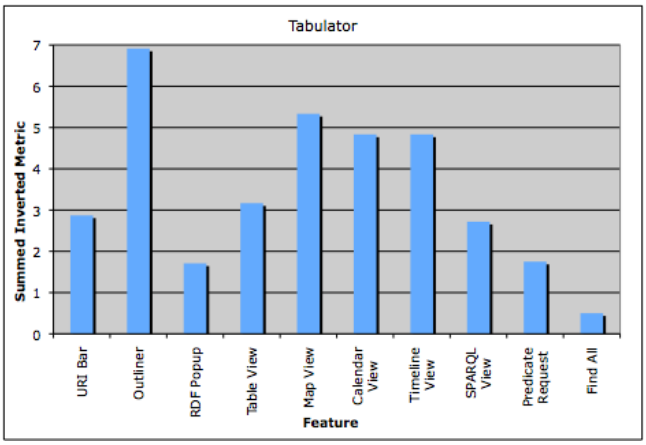

Figure 2: Graph 1 showing the support for search provided by each feature of the Tabulator.

One perhaps surprising result is that, while the table view may provide the most often used representation for analysis, the map, calendar, and timeline views more broadly support search. This prompts the question, which has probably not been asked as of yet: what about their design is different to the table view? Consulting the input table (Table 1) in more detail reveals that compared to the table view, the other views are interactive. With the map,

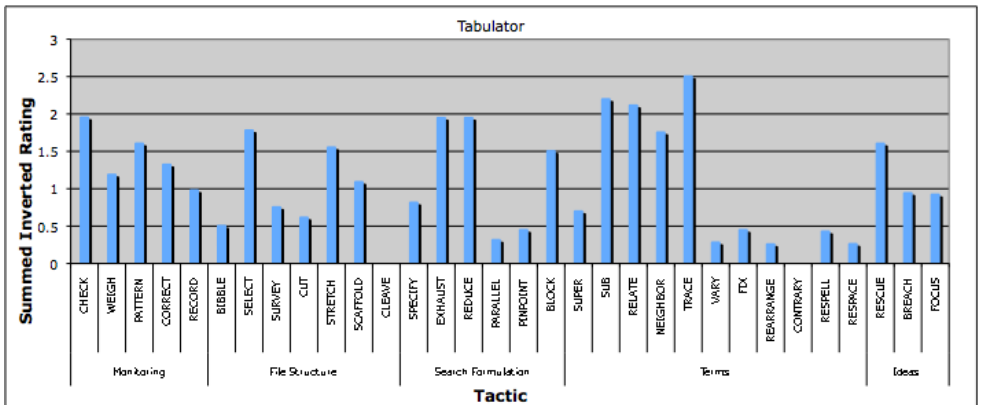

Figure 3: Graph 2 showing the support provided for each search tactic by the Tabulator.

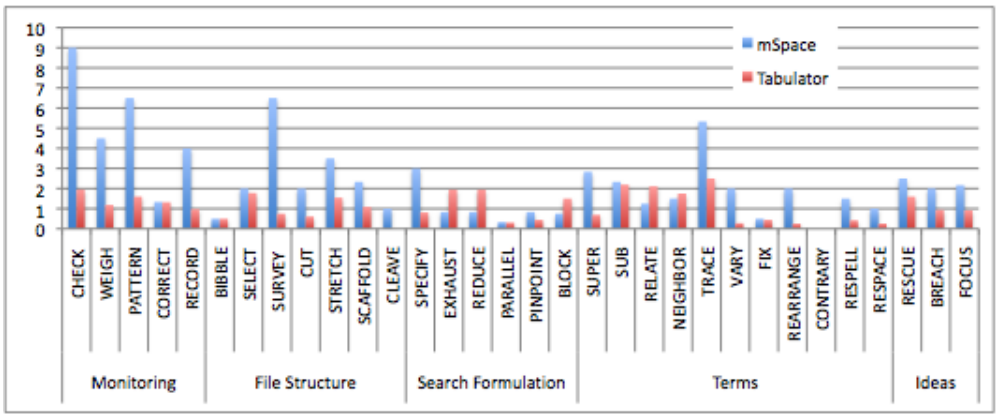

Figure 4: Graph 2, combined with results from a previous analysis of mSpace [18].

for example, the user is able to zoom in on specific groups of results, thus reducing the number of results found. There is currently no means within the table view to manipulate the results and so the subsequent question is, therefore, how could the table view be altered to permit further investigation.

Another perhaps surprising result is the support for search provided by the URI bar that is persistent at the top of the screen. Investigating the input table, shown in Table 1, reveals that, as this persistently shows the URI of the last item clicked on, that it can be used for a number of monitoring tactics. As it can also be used as an input to control the main explorer, the URI Bar can also be used for tactics such as expanding, narrowing, and restarting ones search.

Finally, although it appears only to serve as a means to fill the analysis views below, the 'Find All' button, in of itself, supports the tactic of recording ones search. If it merely populated the views, rather than creating query objects that can be compared or combined, then it would support any particular tactic at all.

Graph 2: Search Tactics

Graph 2, shown in Figure 3, conveys the opposing view to Graph 1, by representing the amount of support available for each search tactic. With Graph 2, therefore, we can identify certain tactics where the Tabulator poorly supports users during search. From first glance, it appears that the Tabulator provides quite a broad range of tactics, but comparison with results from another study [18], shown in Figure 4, shows that it actually has fairly low scores across the board. The purpose of Figure 4 is not to compare the two browsers, as they have different aims, but to demonstrate that equal support for many tactics is different from supporting them all well.

From Figure 3, we can see that there are two tactics that are entirely unsupported; although results from other papers show that these are often the hardest to support. CONTRARY, for example, is to find the opposite of something, which is inherently different from showing everything but something (BLOCK). While TRACE, consulting results to find new search constraints, is often well supported, the tabulator supports this better than actually defining or altering ones search constraints. Consulting the input table reveals that this is due to the many ways of visualizing results, but that the only way to specify ones searches is through the single explorer interface.

One key tactic is to SPECIFY ones constraints, and we can see that it has much more support, compared to some other tactics relating to refining search constraints. This supports the opinion held by many that the Tabulator can be hard for

a user to specify what they would like to find with the Tabulator interface.

It is also clear in the graph, that the first half of the term tactics receive much more support than those in the latter half. This is shows that it is easier to expand and narrow upon ones search than it is to specify variations within them. That is, a user is restricted to either specifying a specific value of a particular attribute, or that they would like any value of a particular attribute. It is difficult using the specify-then-analyze model of the Tabulator to explore variations in either phase, as the results of a user's actions are so distantly removed from the actions themselves.

Graph 3: User Types

Graph 3 is designed to convey how different types of users are supported. The 16 user types are made up of four dimensions of two options, as displayed in Table 2. Like the pattern created by the pairs of options in the table, Graph 3, shown in Figure 5 also has patterns. Further descriptions of these dimensions can be found in previous work [17] and from the originally published model [2].

Method of Search. The first and the second half of the graph, for example, are almost identical, indicating that the Tabulator is just as supportive for people who are scanning or searching, where the latter is characterized by searching for a known item. The second half of the graph is slightly higher, representing slightly better support for those who are searching.

Goal of Search. There is also a clear pattern across the different quarters of the graph, where the odd quarters are noticeably higher than the even quarters. Unlike many browsers, this means that users who are intending to learn more generally about a topic are better supported than those who are specifically aiming to retrieve a certain piece of information.

Mode of Search. The most prominent difference seen is between the odd and even eighths of the graph. This drop indicates that it is significantly harder to use for people who can specify exactly what they need, than it is for people who are likely to recognize the information they need when they see it. This emphasizes one of the results shown in Graph 2 and matches the opinion held by many that it is actually hard to use the Tabulator to find specific information, and that users are almost entirely dependant on what is presented to them as they explore. Ultimately, the user is required to begin at varying starting points, and to seek the information they can only navigate through links and associations. Most existing web browsers provide keyword search paradigms to search for and jump directly to the information they need, and allow navigation from there.

Resource Being Sought. The final pattern seen is between the odd and even sixteenths of the graph, which are slightly higher for the latter part of each pair. This indicates that it is slightly easier to find metadata than it is to find particular information objects. This is perhaps not surprising for a browser of the data-web, which promotes exploration of inter-object associations.

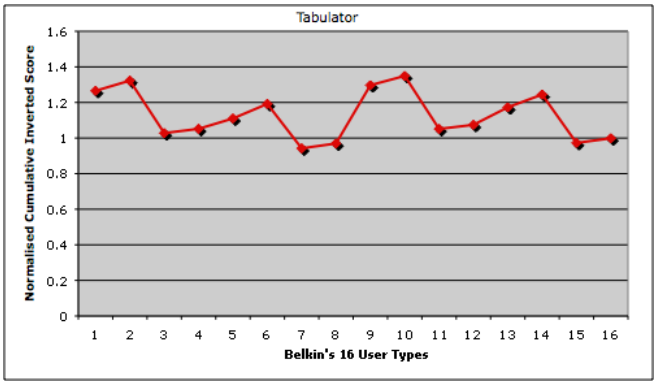

Figure 5: Graph 3 showing the support of 16 different user types, where peaks represent better support.

ISS	Method	Goal	Mode	Resource
1	Scan	Learn	Recognize	Information
2	Scan	Learn	Recognize	Meta-Information
3	Scan	Learn	Specify	Information
4	Scan	Learn	Specify	Meta-Information
5	Scan	Select	Recognize	Information
6	Scan	Select	Recognize	Meta-Information
7	Scan	Select	Specify	Information
8	Scan	Select	Specify	Meta-Information
9	Search	Learn	Recognize	Information
10	Search	Learn	Recognize	Meta-Information
11	Search	Learn	Specify	Information
12	Search	Learn	Specify	Meta-Information
13	Search	Select	Recognize	Information
14	Search	Select	Recognize	Meta-Information
15	Search	Select	Specify	Information
16	Search	Select	Specify	Meta-Information

Table 2: 16 user types defined by dimension [2].

DISCUSSION

In the following discussion, we first interpret the analysis performed on the Tabulator to produce some design recommendations. As the paper is focused primarily on how to apply the framework, we provide some key design recommendations, and then a) discuss how the framework can be used to support redesign, b) provide additional advice and guidelines for when applying the framework, and c) present our future directions for strengthening the framework.

Design Recommendations

Given the findings identified by systematically applying the inspection-based evaluation framework, the challenge remains to consider ways of potentially improving the Tabulator. Uniquely, the same methods that have identified problematic aspects of the Tabulator browser also provide clear direction and design requirements for future redesign. We know, first of all, that one of the key areas of focus should be in improving the users ability to *Specify* what they are looking for. This focus is further supported by the inherent difficulty experienced by those who are trying to retrieve specific information (*Select*).

Graph 3 is created from Graph 2, using a mapping that approximates the needs of each type of user to the tactics they may primarily need to apply. This mapping was validated in detail in previous work [18], where the tactics associated with each user type are clearly listed. Using this mapping, we can see explicitly the individual tactics that require better support. Primarily, these poorly supported tactics include: SPECIFY, FIX, REARRANGE and PARRALLEL. Each of these equate to the users ability to determine and vary their requirements easily. Currently, the primary method of expressing search constraints with the tabulator is through navigating within the explorer, which provides very little means for the user to quickly vary their constraints, as they must re-navigate to do so. Consequently, we recommend that the tabulator explore alternatives to navigation, as opposed to alternative navigation methods.

One of the clearest design recommendations produced by the comparison of interface features, is that the Table view could be easily extended to be interactive, perhaps allowing users to filter, reorder, or remove columns. While many more design recommendations could be extracted from the graphs, we now return to discuss how the framework can continue to be involved in the redesign of a system after evaluation.

When Considering New Design Ideas

While reviewing the findings can provide both clear design recommendations and general areas for improvement, one key challenge for designers is in knowing that any changes will solve them. Another advantage of this framework, however, is that because paper prototypes can be just as easily evaluated as implemented systems, speculative re-designs can included within the already performed evaluation. For example, an evaluator could duplicate Table 1 and insert it into the spreadsheet as a second interface. With even a paper prototype of an additional feature, or a refinement such as an interactive table view, the user can enter the new or refined data into the duplicate table, and quickly compare the old and new designs within the same three graphs. If two potential changes are being considered, then they could both be added to duplicate tables, and both added to another duplicate table, to see what their individual and combined interactions will add to the breadth of support for users.

Further Advice

As with all inspection-based evaluation techniques, the decisions, and consequently the data entered, are subject to some interpretation, especially when estimating mental user actions. This subjectivity is inherent in inspection methods, however, as the evaluators are estimating user behavior, rather than to recording it. Consequently, we recommend, as with Cognitive Walkthroughs, that judgments are made and discussed by multiple evaluators. This is especially important when evaluators are less familiar with an interface, like, for example, when the interface is new and has been designed by a different party.

While striving to record accurate sets of moves, internal consistency is also important, as different interfaces, or even different interface features, become less comparable if evaluator decisions vary over time. Practice, and pilot applications can help establish internal consistency, and statistical methods such as Cohen's Kappa, or Fleiss' Kappa can be used to assess the inter-rater reliability of multiple judges if splitting up analyses, as opposed to jointly discussing them necessary. In the simplest instance, re-checking a small number of early data entries, after completing an analysis, can help improve the reliability of that particular evaluation.

Future Work

While we are not working directly on the Tabulator, we are still extending and improving upon the power and demonstrated accuracy of the framework. Currently, our focus is on assessing the complexity that can be created by adding more interface features to a design. While the current framework focuses on improving the breadth of support given to searchers, the method proposed in [19] is to assess the cognitive load, or information overload, imposed on users through by the combination of features available. Combined with the existing framework described above, this allows evaluators to make trade-off decisions, if necessary, between enriching functionality and reducing interface complexity.

CONCLUSIONS

The aim of this paper was to provide a clear practitioners guide, by example, on how to apply this inspection-based evaluation framework when designing future search interfaces. The framework simultaneously allows evaluators to assess existing interfaces, to understand what gives them their strengths, and new potential designs. The framework is flexible and, quite importantly, can be applied to early designs and paper prototypes, to understand how they will support users during search. The systematic inspection-based evaluation framework has been recently validated against large-scale user studies, to show that it a) produces the same results, b) explains, through the three analyses, what caused them to appear, and c) can be applied in much less time than it takes to carry out a user study. Although we are not advocating that user studies are replaced by early evaluation methods, it is good practice to review designs carefully before they are subjected to expensive user studies are performed. Our own findings have identified a clear example of when, and why, a large user study found no significant differences between the search interfaces in question [18]. Further, the example showed where the browsers did differ, and may have helped to design an appropriate evaluation to highlight their differences.

In demonstrating the application of the framework, we have also presented a detailed evaluation of the Tabulator browser. Designed to support search of the Semantic Web,

the authors of the Tabulator published clearly that it has, until this time, not been assessed for usability problems. The results produced here a) confirm the opinion held by many that, although highly functional, it can be hard to use, b) identify what aspects of the design have caused these problems, and c) presents design recommendations and guidelines to improve the browser.

Finally, to complement this practitioner's guide, by detailed example application of the method, we have also published supporting materials online[4]. First, a template spreadsheet is available that has a) three empty templates, b) 3 pre-prepared graphs, and c) instructions for entering the tables and personalizing the graphs. Second, reference material is provided that includes a) definitions of the 32 tactics, b) definitions of the user-type dimensions, and c) the mapping between tactics and user types to be used when designing improvements for under-supported searchers. Finally, the spreadsheet used during the analysis presented above is also available as a reference example.

In conclusion, we have presented a practitioners guide, by providing a detailed example, on how to apply a recent inspection-based evaluation framework that has been specifically designed to evaluate the support for search provided by information seeking interfaces. In doing so, we have also systematically analyzed a new information seeking browser: the Tabulator, and provided specific design recommendations for improving it. The evaluation framework has been previously validated, but its strength lies in four main areas: 1) it can be applied to both established systems and low-level prototypes; 2) it can be applied to multiple search interfaces, regardless of the datasets shown within them; 3) it has a simple 3 step application process; and 4) it provides clear analyses that convey both *why* a search interface might be unsupportive, to *whom*, and *how* it might be improved.

REFERENCES

1. Bates, M. J. Information Search Tactics. *Journal of the American Society for Information Science*, 30, (1979), 205-214.
2. Belkin, N. J., Marchetti, P. G., and Cool, C. Braque: design of an interface to support user interaction in information retrieval. *Inf. Process. Manage. 29*, 3 (1993), 325-344.
3. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A. and Sheets, D. Tabulator: Exploring and analysing linked data on the semantic web. *In Proc. SWUI06*, (2006).
4. Blomberg, J., Giacomi, J., Mosher, A., and Swenton-Wall, P. Ethnographic field methods and their relation to design. In *Participatory Design: Principles and Practices*. Schuler, D. and Namioka, A. Eds. Erlbaum, Hillsdale, N.J. (1993), 123-156.
5. Bosert, J.L. *Quality Functional Deployment: A Practitioner's Approach*. New York: ASQC Quality Press. (1991).
6. Card, S.K., Newell, A., and Moran, T.P. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates, Inc. (1983).
7. Diaper, D. *Task Analysis for Human-Computer Interaction*, Prentice Hall PTR, Upper Saddle River, NJ, (1990).
8. John, B.E. and Kieras, D.E. Using GOMS for user interface design and evaluation: which technique?. *ACM Trans. Comput.-Hum. Interact. 3*, 4 (1996), 287-319.
9. Lederer, A.L. and Prasad, J. Nine management guidelines for better cost estimating. *Commun. ACM*, 35, 2 (1992), 51-59.
10. Nielsen, J. *Usability Engineering*. San Diego: Academic Press (1994), 115-148.
11. Spencer, R. The Streamlined Cognitive Walkthrough Method, Working Around Social Constraints Encountered in a Software Development Company. In *Proc CHI00*. ACM Press (2000). pp353-359.
12. Rettig, M. Prototyping for tiny fingers. *Commun. ACM 37*, 4 (1994), 21-27.
13. Rogers, Y., Sharp, H. and Preece, J. *Interaction Design*. John Wiley & Sons (2002), 454.
14. Shneiderman, B. and Plaisant, C. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In *Proc AVI06 Workshop on Beyond Time and Errors*. ACM Press (2006), 1-7.
15. Wharton, C., Rieman, J., Lewis, C., and Polson, P. The cognitive walkthrough method: a practitioner's guide. In *Usability inspection Methods*, J. Nielsen and R. L. Mack, Eds. John Wiley & Sons, (1994), 105-140.
16. White, R. W., Kules, B., Drucker, S. M., and schraefel, m. c. Introduction. *Commun. ACM 49*, 4 (2006), 36-39.
17. Wilson, M. L. A Transfer Report on the Development of a Framework to Evaluate Search Interfaces for their Support of Different User Types and Search Tactics. School of Electronics and Computer Science, University of Southampton (2008).
18. Wilson, M. L., schraefel, m.c., and White, R. W. Evaluating Advanced Search Interfaces using Established Information-Seeking Models. *Journal of the American Society for Information Science*, (to appear).
19. Wilson, M. L. and schraefel, m. c. Improving Exploratory Search Interfaces: Adding Value or Information Overload? In *Proc. HCIR08*, (2008).

[4] http://users.ecs.soton.ac.uk/mlw05r/searchinspection/