Problem-Resolution Dissemination

by

Kevin Quan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
In
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2006

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The current problem-solving paradigm for software developers revolves around using a search engine to find knowledge about the problem and its solutions. This approach provides the developer with search results that are only restricted by the context of the keywords they used to search. *Problem-Resolution Dissemination* (PRD) is a system and method for collecting, filtering, storing and distributing knowledge that users discover and access when solving a problem. The method involves an agent running on a user's (Alice's) browsing client which is enabled when Alice is solving a problem. After Alice indicates that she has solved the problem, the agent will collect all web pages visited when solving the problem and filter out the pages that are not relevant. Pointers to the remaining pages (URIs) are tagged with Alice's identity and stored in the central repository. When another user (Bob) attempts to solve the same problem, the above repository is queried based on Bob's social context. This social context is defined by Bob as a group of other users who have one of three trust levels: team, peer or community. The results are displayed by ranking them within each of the above contexts. In the event that no results are relevant to the Bob, he has the option of following traditional problem solving approaches. When Bob has solved his problem, the web pages he visited are added to the repository and made available to future users. In this manner, PRD incorporates relationships and previous experiences to improve the relevancy of results and thus efficiency in problem solving.

Acknowledgements

Several people have provided me with invaluable advice, motivation, and inspiration required to complete this research. I would like to thank them for all the help provided:

Pauline Shou for always supporting me and waiting patiently while I perform my research and write my thesis.

Mohammad Ahmad Munawar for being a sounding board to bounce ideas off of, and for general graduate student advice.

Valentina Popescu and Craig Salter for providing feedback on the prototype.

Frank Jania for forcing me to reason analytically about the aspects of PRD which are novel and innovative.

Marin Litiou for seeing the value in this idea and providing the leadership required for IBM to recognize that value.

Paul Ward for teaching me the finer points of English writing, helping me to refine the idea, and providing key insight at just the right moments.

Dedication

This thesis is dedicated to my parents who taught me the value of education, and supported me through the first 25 years of my life. Without them and their hard work, I would not have had the opportunity nor the means to reach this stage of my academic career.

Table of Contents

List of Tables

List of Figures

1 Introduction

A software engineer's mandate is simple: to engineer software with a certain level of quality. The difficulties are, as always, in the details. Creating and maintaining high-quality software, regardless of size, is a challenge in many aspects; features and people need to be managed, bugs need to be fixed, and documentation needs to be written. Amidst these individual tasks, problems and errors are bound to reveal themselves; a new member of the team will have a configuration error, or a bug will return an arcane error message. The first line of support to solve these problems is the user who encountered the error and is responsible for the task. This user may or may not be a software engineer nor an expert in the problem area.

Many users who regularly deal with computers will have been exposed to situations where they have to assess the situation and determine how to proceed. In this scenario, they must learn about the causes of the situation and the effects of potential paths of action. For the general case, these situations are alleviated through the use of documentation and training; the user is able to learn how best to proceed by relying on the experience of other users that have passed this scenario.

There are instances, however, where neither documentation nor training can prepare the user to solve an impasse. In these cases, the user must rely on their problem-solving skills in order to learn about the situation, adapt their knowledge, and solve the problem. There are a variety of methods to accomplish this, but the predominant way to independently solve a problem is through the use of a search engine:

1. The user determines a likely set of keywords that when searched will return results relevant to their problem.
2. A search is performed and the user uses their cognitive skills to analyze the relevance of the search results against the problem they are facing.
3. One or more solutions may be attempted before the problem is verified to be solved.

Under this generic workflow, there are several points of ambiguity that arise which impede the user's task. While a search engine generally provides sufficient accuracy, the onus is on the user to read each result and make a conscious decision as to whether the link is relevant to their problem or not. This requires time and active effort. The effectiveness of this task is directly related to the time spent in solving the problem, as proper identification of search results will result in less time following dead ends.

Secondly, it would be incorrect to assume that in all cases the most-relevant result for the problem at hand is ranked as the first result. This could be from the simple fact that the user did not construct an adequate search query, or the search engine may not have specific information about the search query, or there is no way to properly specify the required search context. The effect is that it forces the user to continue parsing the results until they gather enough knowledge to solve the problem. This may be on the first page, the fourth page, or the fiftieth page of results.

The costs encountered and effort given by a single user is amplified by scale. Due to the large number of people worldwide, multiple users will encounter similar problems. When they use this traditional approach to find a solution, there will be a great deal of inefficiency from each user repeating the same steps from the beginning.

This inefficiency is not trivial. The previous points described problems in hypothesizing which results are relevant. Once this has been accomplished, there is additional effort and time required from the user to perform experimentation for each result, such as the following workflow describes:

1. The user identified that a search result may be relevant, and spends time to comprehend the information within the result.
2. If the information contained with the search result is promising, a solution based on that information is attempted.
3. If the attempted solution was not successful, the work must be reverted.
4. The user restarts the process with a different search result.
5. The user may refine their search query in order to retrieve more relevant results.

Due to the need for constant experimentation, current problem solving is in itself a problem. This problem is further exacerbated when the relevant result for a problem is not ranked prominently by the search engine. A better approach would involve the following concepts:

1) Reducing the amount of experimentation required by returning a better ranking of results for the problem at hand.
2) Returning a better ranking of results for the problem at hand by providing the user with a greater ability to define search contexts.
3) Increasing user trust of the ranking in order to reduce the decision time for choosing relevant results.

Our approach attempts to address the redundancy and experimentation required in problem solving (specifically web searching) by collecting and ranking sources of knowledge used during the problem solving process and sharing this information with future users who attempt to solve the same problem. This addresses the above points as follows:

1) Experimentation is reduced as the results are more relevant due to their use by previous users in solving the same problem.
2) Results are more relevant as users can specify a social context which sorts and ranks results based on who has previously solved the problem, and who the user performing the query trusts.
3) Comprehension time for deciding whether a result is relevant is reduced because the user is leveraging previous experiences of users that they trust.

Our approach is described with users being defined as software developers. The problems typically encountered by these users are errors that surface as error messages.

This document is organized as follows: Section 2 will describe traditional approaches to problem solving as well as outlining the deficiencies these approaches have compared to our approach. Section 3 will describe our approach, known as *Problem-Resolution Dissemination* (PRD). This includes an architectural view of the system as well as the methods used to collect and disseminate knowledge. Section 4 will describe in detail the prototype we created based on the design. Section 5 discusses how our approach can be validated. Finally, Section 6 concludes the discussion, Section 7 discusses future work in this area, and Section 8 is a glossary of abbreviations used.

2 Background

Developers (and, more generally, users) have attempted to gain knowledge to understand and solve problems using a variety of resources. This section will detail the most commonly used sources of information, as well as the deficiencies these approaches have when compared to our proposed approach.

Search engines are one of the most widely-used interfaces to query for information. There are a variety of search-engine types that range from generic search engines, to search engines on specific topics, to search engines that aggregate their information on specific criteria. Search engines are discussed in Section 2.1.

In addition to search engines, developers can search for information within Expert-Edited Directories, as discussed in Section 2.2; Wikis, which are discussed in Section 2.3; and documentation, provided by software vendors, as discussed in Section 2.4.

The last category of information is through recommendation services. This can be done through social-computing services, as described in Section 2.6.1 or in conjunction with search engines, as described in Section 2.6.2.

Section 2.7 describes feedback agents, which, while not an alternative solution to the problem at hand, utilize a similar approach to gather and use information for the benefit of the end user.

In this section, the term "domain" will be used in two different contexts; the first describes a subject field or area such as the use of the word domain in *domain-specific search engines*. The second use of "domain" is in terms of an Internet name domain; for example, *google.com* is Google's domain name. To avoid ambiguity, the term *URI domain* will be used to refer the latter definition of domain.

2.1 Search Engines

Several types of search engines will be discussed. The most-commonly used, Generic Search Engines will be discussed in Section 2.1.1. This is followed with Domain-Specific Search Engines in Section 2.1.2, Meta-searching in Section 2.1.3, and finally Collaborative Searching in Section 2.1.4.

2.1.1 Generic Search Engines

Using a search engine is a common way for users to find information on the Internet. However, before a search engine can be used, its information repository must be populated. This is done by agents of search engines, commonly known as robots, which visit webpages on the *World Wide Web* (WWW) in an expansive fashion by following web links. During this process, commonly known as a crawl, information pertaining to the webpages such as keywords and associations are indexed and stored. This information is then made available to the user by an interface that allows keyword queries.[1] This process repeats itself periodically in order to maintain up-to-date information.

For example, a search engine may crawl and index every page on *Wikipedia* (an online encyclopedia). Then, when a user searches for the keyword *Elephant*, the search engine will return the pages (which correspond to *Wikipedia* encyclopedia entries) that are related to elephants along with any other pages that it has crawled that are related to elephants. These pages are ranked based on a relevancy algorithm by the search engine.

The are numerous corporations offering search engines. However, there are 5 major search engines being used by the general public in the USA.[2] In the rest of this section, Google will be used as an example to represent a search engine due to its familiarity, with a market share of 43.7% of US searches as of July 2006.[3]

While the major search engines may have differing algorithms to determine relevancy of search results, some general remarks can be made about their behaviour in areas in which PRD is advantageous.

1. Generic search engines are domain-agnostic. This is in the implicit nature of this type of search engine as their aim is to cater to the general public rather than perform domain-specific search (which will be discussed in Section 2.1.2). Due to this jack-of-all-trades nature, these generic search engines do not capture any domain-specific context aside from what is entered as keywords by the user.

2. All of the major search engines are machine learned. The process to index and learn about the content of the pages is not performed by humans, but by computer algorithms. This is necessary due to the sheer number of pages on the WWW; for example, Google has indexed over 8 billion pages as of September 2005.[4] The magnitude of this task is further increased as pages must be re-indexed frequently in order to detect changes in the content.

   Due to being machine learned, there is only a slight social aspect to search engine results. This social aspect is due to the linking of one hypertext page from another using a hyperlink. Generally, the owner of the linking page must make a conscious effort to create a link to the destination page, which in turn exhibits his faith in the linked page. This is the basis of Google's PageRank algorithm.[5] However, the social context of this action is unreliable as the context of why the owner created the link is not necessarily known.

3. Boolean/logical operators such as *AND* or *OR* have been used in the major search engines since at least the year 2000[6], and many are beginning to include more, specialized contexts, such as location or movies.[7] However, social context such as team and peer groups have not been implemented in generic search engines. Research search engines, such as Referral Web[8], have investigated the idea of incorporating social elements in order to search for people who possess contain knowledge. This is tangential to our approach, where we leverage social context in order to prune search results.

4. All search engines have slightly differing algorithms that give different results for search terms. Due to this, it is difficult to say which search engine result is correct for a particular instance. Instead, users rely on their confidence that a particular search engine is generally "correct," ignoring the possibility that another domain-agnostic search engine may provide more-relevant results for a particular query. This sociological phenomenon occurs as their habitual search engine generally returns good-enough results, and as such there is no motivation to consider alternatives.

5. Search engines capture *large-scale relevance*. If a problem is encountered, solved and documented by a single person, the solution is not easily found unless there is some sort of uniquely identifying sequence of characters on the documentation page. Conversely, if 10,000 people perform the same actions, the solution will be more easily found.

   A solution to a specific problem in a certain context may only be documented by a small number of people, while a specific problem in a general context may by documented by a large number of people. It is likely in this scenario that the former solution would be overshadowed by the latter solution. Users searching for the former would have a very difficult time finding it.

Search engines, being domain-agnostic and lacking social context, have tradeoffs that users have become accustomed to. However, the goal of a user who uses search engines is to find the most-relevant information for their query. Thus, if the shortfalls of search engines can be addressed, the utility of the search results can be improved. A summary of the shortfalls are as follows:

1. Generic search engines are domain-agnostic
2. The information repository is machine learned, and thus lacks a usable social aspect.
3. Search engines incorporate context information, yet not relationship contexts at present.
4. User's loyalty to a specific brand of search engines may impact correctness of search results.
5. Uncommon problems are difficult to find due to search-engine results ranking.

2.1.2 Domain-specific Searching

A subset of search engines that generally provides better search results for a given problem is domain-specific searching. Domain-specific searching requires the information repository that serves as the basis for the search result to be restricted, either through manual or automated means, to the domain in which the user is searching.

This filtering of irrelevant information is by no means an easy task. A manual approach to filtering requires a tradeoff between accuracy of the filtering and time taken. For example, selecting the relevant links from Google's repository of 8 billion pages, with an estimation of a 10 second think time per page, would require more than 900,000 days.

Even if this task is distributed over a city of approximately 1 million people, they would have to drop everything they were doing, and work non-stop for the majority of the day in order to filter the pages for a single domain of expertise.

By demanding less accuracy, a user can significantly shorten the time needed to retrieve results from a domain. If the user has previously identified a website that contains the information they require, tools such as Google's `site:` operator[6] or Yahoo Search Builder[9] can restrict a search to a particular URI domain. However, while this approach presents an acceptable tradeoff of time for functionality, it requires an active effort on the part of the user to be knowledgeable about the particular websites they wish to search.

Suppose the above criteria of filtering the information repository for the search at hand is satisfied; it is possible that there are communities and websites that correspond to the domain in which a user is searching, thereby facilitating domain-specific searches. However, in this case the user is still restricted by the search interface in describing the context of their search. Like the search engine superset, domain-specific search engines generally use a keyword interface to provide a search key. As previously discussed, this does not at present provide the means to include the context of the team or peer group a user is working.

There has been research in providing more-relevant results by gathering domain-related information with respect to the user's search. Hullen *et al*. proposed WebPlan in order to create a search agent which aids the user by planning a set of queries to find the knowledge the user is looking for.[10] However this approach requires the user to execute multiple queries in order to provide context to narrow the information repository to a state where the relevant results can be obtained; something that our proposed approach can generally provide on the first query for common problems.

In summary, while domain-specific searching would improve the relevancy of results for a user's search query, there are significant barriers to providing knowledge as efficiently as our proposed approach.

1. Domain-specific search requires knowledge of repositories that are relevant to the search, which generally requires significant time to identify.
2. Interface to domain-specific search does not improve beyond a search engine's keyword query interface and context capabilities.

2.1.3 Meta-searching

Meta-search, also known as multi-search, is a search engine which sends a user requests to multiple search engines (and/or expert-edited directories), aggregates the results and presents the final aggregated result list to the user. In effect a user is searching multiple search engines at once.[11] Meta-searching can also be thought of as the query interface to multiple, heterogeneous databases that have been combined through research performed in the *web-integration* area.

The interesting application of meta-searching with respect to PRD is if a meta-search engine searches both generic and domain-specific search engines. However the deficiencies of this approach are similar to those listed in Section 2.1.1 for Generic Search Engines and Section 2.1.2 for Domain-specific Searching.

2.1.4 Collaborative Searching

Collaborative searching is the process of combining search queries from a selection of different users in order to improve the search results of that particular query. At a high level, it is similar to our proposed approach.

Walkerdine & Rodden propose a system of sharing searches between users by providing an alternative interface for searching (versus the conventional web browser). Their approach proposes that future searches on a saved query may query not only "the repositories he or she is attached to interrogate, but also the ones the original owner of the query is attached to."[12] This is similar to our proposed approach if we assume that users are using a variety of search engines as their repositories; a user will view results from their preferred search engine as well as key results from the search engines preferred by others that have performed the same query.

There are several drawbacks to the approach that Walkerdine & Rodden suggest:

1) The user must use a *Local Query Management System* to instantiate queries, rather than conventional means,
2) The recommendation system, only available at the query construction level, is a manual process,
3) The user context is only at the query-construction level rather than associated with the display and ranking of results,
4) There is no concept of social groups to filter the display and ranking of results.

Gnasa has released an abstract detailing an approach to improve collaboration in web searches by using a peer-to-peer architecture and personalized ranking list in 2004.[13] This has been followed up in 2005 with a description of a system to share information through a peer-to-peer architecture.[14] In their approach, Gnasa *et al*. store individual search history within *Peer Search Memory* data structures on each client. Subsequent queries on the same topic cause requests to be made to peers for information on the chosen topic. The peer group is restricted by autonomously building a semantic layer on top of the peers known as a *Virtual Knowledge Community*.

While the approach taken by Gnasa *et al*. is similar to our own, there are key differences that affect the effectiveness and efficiency of the solutions. First, the use of a peer-to-peer architecture includes these user-facing requests:

1) A unique (new) client in order to incorporate the necessary communication and organizational requirements for the peer-to-peer architecture.
2) Communication time to poll the members within the *Virtual Knowledge Community* for relevant results.

3) Accuracy of results relies on present (available) peers.

Secondly, the creation of *Virtual Knowledge Communities* through data-mining means do not capture social relationships explicitly known or wanted by users. These include:

1) The ability to prioritize results from certain individuals.
2) The ability to exclude certain individuals or groups for certain queries.
3) The ability to create custom, personalized communities.

Lastly, the underlying infrastructure of their approach is sufficiently different that it imposes certain requirements on the client machine. These include:

1) Online overheard required to maintain *Peer Search Memory* data structure.
2) Online overheard to create semantic *Virtual Knowledge Communities*.
3) Communication cost to respond to peer requests and to query peers.
4) Overhead to maintain peer-to-peer architecture (joins/leaves).

Our approach offloads some of this functionality to the user, resulting in a manual process. However, it is with the intention that the user controls result in a more flexible and adaptive environment, which benefits the user in terms of search-result clarity and relevance.

2.2 Expert-Edited Directories

An expert-edited directory (EED) is a collection of hyperlinks ordered in a hierarchical manner by a group of editors. These editors are domain experts and perform several tasks, including[15]:

1. ensuring the quality of the content on a website before a site can be included in the directory;
2. ensuring that links within the directory are categorized correctly;
3. removing dead websites from the directory.

Examples of expert-edited directories including Yahoo! Directory[16] and Open Directory Project (ODP)[17].

Expert-edited directories incorporate a greater social aspect to an information repository at the expense of the size of the repository. An EED is created and maintained by humans. Humans submit suggestions of websites to add to the directory, and human experts who are designated to edit a single or group of categories determine which links should be included within the directory. In contrast to search engines, there is no machine that navigates the WWW and indexes pages.

It is important to note the distinction between websites and pages. A website consists of numerous pages. However, it is generally the case that a website is submitted to a directory, not individual pages. If a page is submitted, it is generally a navigational apex

from which the other pages within the website can be found. This is a drawback compared to search engines as the relevancy of a website to a problem, compared to a page, is much more coarse. It is arguable that the granularity level of a web page may still be too coarse. However, in terms of relative scale, our proposed approach is more focused than EEDs due to our use of web pages.

EEDs improve upon the generic nature of search engines by allowing editors to provide their personal social criteria to links. Editors determine relevancy and value of each link and serve as the gatekeeper for their categories. However, it is important to note that this can also be a hindrance on the information repository as editors will knowingly, or not, incorporate their existing biases into their governing functions.

An additional social aspect is the ability of the owner (or submitter) of the pages to include their own description of the page.[18] However this information can be misused by the author to promote their website in ways that their content cannot support. The editorial system should be the check needed to prevent this exploit.

An EED's user interface is primarily categorical which restricts the ability to search for relevant information. A user must navigate to the appropriate category (of which there may be many) and then manually scan the hyperlinks for relevancy to the problem at hand. While generally EEDs have a search interface as well, this search interface only searches description information stored by the directory (i.e., information provided by the owner or submitter of the link). Thus, a directory search is not as intelligent as a generic search in addition to being potentially biased.

Like search engines, EEDs are also prone to *large-scale relevance*. For more common topics, a large number of links within the category will be a roadblock to user comprehension, and editors will attempt to limit the hyperlinks to those that can provide the most value to the greatest number of people. Thus, documentation for less-common problems may be removed from the page due to its smaller audience.

The disadvantages of EEDs for the problem at hand can be summarized in the following points:

1. An entry in an EED is an entire website, which is less granular than individual search results returned by search engines.
2. Completeness of the information repository compared to search engines is restricted by the operational capacity of human editors.
3. Biases potentially introduced into repository by editors.
4. Inefficient query system (categorical) and access to relevant hyperlinks.
5. Listing in directories may be biased to websites that provide information to largest number of people.

2.3 Wikis

Wikis are collaborative tools on the WWW that allow users to add and edit articles on the wiki with minimal effort.[19] Wikis enable people to document their knowledge in a

written form hosted at a central repository. Perhaps the most prominent usage of wikis on the WWW is *Wikipedia*, an online collaborative encyclopedia.[20] *Wikipedia* is also an excellent example of how an information repository can be created and maintained through the use of a wiki.

Corporations have begun adopting the use of wikis as documentation tools such as eBay with eBay Wiki[21] and Microsoft with MSDN Wiki[22]. eBay Wiki is a community where users create and edit articles about eBay in specific areas. Examples range from the basics of how to sell something on eBay, to specific facts about eBay's policies. The eBay Wiki provides first-line information to users that are trying to learn how to use eBay.

The MSDN Wiki is along the same lines in that it provides domain-specific information. The MSDN Wiki is more relevant to this discussion as it is a means for troubleshooters to converge and document their problem resolutions about specific technologies. This knowledge is then transferable to users who encounter similar problems in the future.

While knowledge sharing with wikis is similar to PRD, there are some areas where wikis are not as powerful for this particular task:

1. In order for information to be entered into a wiki, a conscious effort must be made by the developer to document clearly and logically the steps involved in solving the problem. However, the steps cannot be verified to be complete until the problem is solved as details or specifics may change.

   This poses a problem as a developer will wish to move on to other tasks, or may have forgotten some details once the problem has been solved. Thus, it is both bothersome and a disincentive to have to document information into a wiki after a problem is solved. However, without this time and manual effort, the required information will not exist.

2. Contribution to a wiki is limited if a solution is already documented. Users do not feel the need to spend the time and effort to append or edit a solution if one already exists. This may result in incorrect information propagating for a period of time until it is fixed in addition to being discovered.

3. Alternatively, a wiki article with all solutions is potentially lengthy. Users will be overwhelmed with information. The solution is pruning, which may remove vital information for certain people depending on the team context in which they are working.

4. Ease of editing and reducing vandalism are opposing goals for a wiki. In some configurations, where anyone can edit an article[19], vandalism and errors may be introduced into the information repository with any edit. Vandalism can be limited by enforcing authentication on the user; however, this will dissuade user adoption and make it more difficult for users to enter information. Vandalism and

errors are present in articles until they are both discovered and edited. The presence of editors/moderators may stem this issue; however, it may cause problems as discussed under Expert-Edited Directories.

Wikis are an increasingly popular method to collect and share tribal information about certain topics. However, it requires active user participation to be truly effective. In summary, the deficiencies of Wikis in comparison to PRD are:

1. Entering information in wikis requires time and effort
2. Users are hesitant to spend time and effort if a (partial) solution exists
3. Information on wikis can be overwhelming and without context
4. Vandalism and errors are easily introduced.

2.4 Documentation

Documentation refers to text provided by the developer of specific software which details the behaviour of the software, how a user can affect this behaviour, and what a user can do if they run into problems. It has become increasingly popular to make documentation publicly available on the WWW in order to make it more accessible (i.e., if a manual is lost, a copy can be found on the Internet).

There are a variety of problems with documentation. It is significantly less powerful as a source of problem resolution than the other sources due to its lack of breadth. However, it has been included in this survey since documentation from a software developer should be the most accurate source of information for a problem being caused by their software product.

Some of these problems include:

1. Documentation of a specific product does not effectively capture information about problems that occur during integration of the product with other software packages. While major product combinations may be discussed, it cannot be discounted that third-party software packages may reverse engineer (or utilize open) interfaces to integrate with a specific product. Problems caused by these latter combinations may not be discussed within the provided documentation.

2. Documentation is a single source of information. While documentation may be more accurate than other sources, it cannot capture the breadth of solutions available on the WWW, nor the user experience incorporated into a wiki. Problems with integration described in the previous point are better solved using a search engine or wiki approach.

3. Documentation does not capture user experience. Documentation is generally released by the software developer. It is not feasible for the software developer to release incremental additions to their documentation every time a user successfully solves and documents a problem resolution.

These problems are significant and detrimental to the value of documentation. Perhaps the final nail in the coffin is that the documentation information repository is generally a subset of a search engine's information repository. A search with the proper keywords in a search engine will reveal the information contained in the documentation as well as additional information.

2.5 Asking For Help

A common approach to solving problems is to ask someone for help (defined as an expert in this sub-section). The motivation to take this course of action is the assumption that the person has more experience, is more knowledgeable, or can offer insight into the problem resolution. This process can take two forms:

1) it can be a physical process whereby the user approaches someone in their immediate environment or via a phone call, or
2) it can be done electronically by sending an email or posting a question on an open forum.

In the electronic case, presently asking a question is differentiated from being able to search for an archived thread asking the same question using a search engine. An example of the electronic case is *Experts Exchange*.[23]

In both cases, there are several areas where PRD is an improvement:

1. Asking for help, while potentially reaching a large number of people in the electronic medium, still only reaches a subset of all the people working in a particular area. In the physical medium, only the people who a user can contact are available for advice. In an electronic medium, only the people who participate in the websites where the question was asked, and have viewed the current state of the question, can share their knowledge.

2. Advice from others may address their interpretation of the problem that they believe the user is asking about, which is not necessarily the question the user is asking. Alternatively, responses to a question may not be clear enough to be understood. Thus in both cases there will be a lag time until the subsequent communication clears up the question and/or response. All the knowledge needed to resolve the problem may not be present once the problem is identified.

3. Asking someone passes an implicit cost to the expert, to first comprehend the question and second to construct and explain the solution. Depending on the resources of the expert, this affects the quality of response measured by accuracy and completeness of the resolution.

4. Repeat queries about the same problem require repeated effort to explain. Even if the problem resolution can be duplicated easily, there is a cost on the expert to read and comprehend the solution again. This disadvantage is most prominent in the physical realm.

Asking for help captures the relationship context that PRD incorporates; however, there are disadvantages in the infrastructure of the current Ask-For-Help system which promote inefficiencies in access to information. These are summarized in the following list:

1. Asking for help only reaches a subset of the people who are solving the problem.
2. Communication introduces a time lag in solving the problem.
3. Asking for help conveys a cost to the expert.
4. Demand on an expert's time increases linearly with the amount of requests.

2.6 Recommendation Systems

Recently, there have been a collection of products released on the World Wide Web that provide recommendation services, specifically based on what website a user is visiting. A recommendation service provides supplementary information to the user based on the context the service can glean from a user's behaviour, and the system state. Supplementary information can be in the form of links to web pages, comments and ratings from other users, or information that the service predicts the user will be interested in next.

2.6.1 Social Bookmarking

Social bookmarking is the evolution of bookmarking. Bookmarking is the practice of saving specific URIs for later retrieval. Originally, this was primarily done within the web browser environment; a user would use specific functionality within the browser to bookmark a URI and its title into a hierarchical store. Bookmarks can later be retrieved within a separate user interface and viewed within the web browser by retrieving the content from the given URI.

Social bookmarking changes this paradigm slightly by shifting the storage mechanism to a central server instead of the user's client. By aggregating the bookmarks of a variety of users, a social-bookmarking service can gain insight into a society's bookmarking patterns. Examples of social-bookmarking services include Yahoo!'s Del.icio.us[24] and IBM's dogear[25].

Most social-bookmarking services also have the concept of tags. Tags are similar in concept to categories or keywords. When a user submits a bookmark to the service, they tag the bookmark with terms that relate to the URI. Over a large number of users, these tags can then cluster bookmarks on a similar topic together.

Shadows is a social-bookmarking service developed by Pluck which adds recommendations capabilities.[26] Users visiting the *Shadow Page* of a link see discussions and ratings of the link entered by other users. But perhaps the most crucial element is the ability to navigate to other links similar to the viewed *Shadow Page* based on the tags. Implicitly, the social-bookmarking service recommends related pages on a topic based on the clusters of links.

There are some drawbacks to using social-bookmarking services for recommendation:

1. Tag granularity and accuracy is at the discretion of users.
2. The granularity of recommendation is limited by tags.
3. There is no push mechanism to automatically present relevant information to the user based on the query.
4. Here is no concept of tiered confidence levels in user ratings.

2.6.2 Search-Engine Recommendations

Google's Personalized Search performs a function similar to recommendations. Over multiple searches, Personalized Search reorders the search results for a user based on what a user has clicked on in the past. The purpose of this is to surface links that are most relevant to the user higher in the search results.[27]

The key similarity between this product and our approach is that it tracks the frequency of links that are followed based on a specific query. As a link is followed more frequently for a query, the confidence and relevance of that link increases.

Additionally, the collection of which links a user is viewing is tracked implicitly as long as the user follows links directly from the Google search engine. This poses two restrictions: 1) the user must use Google as the search engine, and 2) links can only be tracked a single step away from the search engine. This means that if a user follows a chain of links from the search results, only the first link in the chain will be captured by Google Personalized Search.

In addition, Google Personalized Search is lacking the social contexts that are present in our approach. Thus, there is a significant lack of ability to share the updated search-results ranking with team members or peers.[28]

In summary, the disadvantages of Google Personalized Search compared with our approach is as follows:

1. Users must use Google as their search engine
2. Visited links are only tracked a single step away from the search engine.
3. No social element exists.

2.7 Feedback Agents

Feedback agents are described in this section as an agent which instruments or profiles an application or system in order to collect operational information about the application or system. This information is then sent back to a central server provided that the user has given permission for the agent to do so.

The purpose of this agent is so that the agent provider will have a large set of actual data about the operational characteristic of the application or system, and will be able to data mine this information for areas of improvement.

A common example of a feedback agent is a crash reporter.[29] An example of a crash reporter is Mozilla's Quality Feedback Agent which they license from SupportSoft. This agent is used to send information about crashes in Mozilla back to Mozilla developers for the purposes of trouble shooting and problem determination. The end goal is to fix bugs which may have led to the crash.[30]

The similarity between feedback agents and our proposed approach is that both collect information from the end user and submit the information to a central server for aggregation. Due to the fact that the end goals are substantially different, there are several strong differences between feedback agents and PRD. These are itemized as follows:

1. Feedback agents generally collect indirect feedback based on observations of the user. PRD can be considered as direct feedback as there is an associated task and user context with the information being collected. Additionally, the feedback mechanism described in Section 3.5.2.3 is an instance of direct feedback. Further comparisons between indirect and direct feedback is performed by Hilbert & Redmiles.[31]

2. Information submission through feedback agents can be done anonymously. Thus there may be no user context associated with any of the information. This anonymity is preferable as it increases user adoption and minimizes privacy concerns.

3. Feedback agents do not generally allow the end user to see the results of their or their peers' information submissions. This is due to the fact that the goal of information collection in feedback agents is to produce information for the agent provider, not the end users; and because anonymous operational characteristics are generally of little value to the end user.

4. There is no concept of social relationships or social contexts in feedback agents, since no user context is submitted with the information.

5. Due to the fact that feedback agents generally instrument and/or profile the application or system they are collecting information for, there is additional overhead required to collect this data. PRD requires no additional overhead provided that the user's browsing environment is typically configured.

The most-relevant comparison between feedback agents and PRD is the last point in the above list. Feedback agents consume resources in order to gather information. PRD does not require this additional overhead as it assumes that the user is typical and enables a browser's history feature for a period of time exceeding the time it takes to complete a problem resolution.

3 Problem-Resolution Dissemination

Problem-Resolution Dissemination (also referred to as PRD) is an approach to include social-context information when searching the World Wide Web. PRD automatically collects the pages visited by a human during their quest to solve a problem, aggregates this information over all the humans that have attempted to solve this problem, and presents this aggregate of information to subsequent users who attempt to solve this problem.

Section 3.1 will introduce PRD at a high level, Section 3.2 will lay out the fundamental architecture of PRD and Section 3.3 will introduce the social constructs used within PRD. Section 3.4 discusses changes made to a user's workflow when using PRD which leads into the query techniques being used, as described in Section 3.5.

3.1 PRD Overview

Problem-Resolution Dissemination incorporates social context into searching the Web by building an information repository based on pages that users have visited and linking those pages with social relationships defined by users. The benefit of including social relationships within the search is to produce more-relevant results when the goals of the people in the relationship are aligned.

Social relationships are divided into three classes of contexts:

1) a *Team* context consists of people who are working on your immediate team,
2) a *Peer* context consists of people you respect who are working in the same field, and
3) a *Community* context which consists of all the people who have solved the same problem that you are attempting to solve.

Problem-Resolution Dissemination is meant to coexist together with conventional search engines in the problem-solving process. During a problem-solving process, it is envisioned that a user will draw upon the information repositories of both PRD and search engines in a proportion decided upon by the user.

The information within PRD's information repository is gathered automatically from users who have previously solved the same, or a nearly identical, problem. This information is filtered and presented to the user, upon instantiating a query, and grouped based on the user's specified relationship classes. The greatest relevancy is given to the *Team* context, followed by *Peer* and finally the *Community* context.

From a technical point of view, PRD is an innovation by minimizing the impact on a user's problem-solving workflow. In order to initiate a PRD process, the user must notify the PRD Controller and initiate a search query. The latter follows the same pattern as a user would when initiating a search query on a search engine; that is, to enter a string of text into a search box. Both of these steps can be combined into a single atomic function.

Information is populated into PRD's information repository based on the pages that a user visits during the problem-resolution process. This may include results from search engines, pages that a user visited when following a hyperlink, and pages included in the results from PRD. No instrumentation is required of the user's operating environment, and there is no additional overhead in keeping track of the user's behaviour based on expected user configurations.

One additional step in a user's workflow is to require the user to notify the PRD Controller that the PRD process is finished. Once this is done, PRD will retrieve the pages a user visited in learning and solving the problem, and submit this information to the central server. PRD will also solicit optional feedback from the user to further improve the accuracy of its information repository.

The experiences of this user is then incorporated into the data for subsequent users to retrieve.

3.2 Architecture

PRD is designed in a typical client-server architecture. The responsibility of the client components are to collect information and display information to the user. The server architecture performs the majority of the computations in addition to data storage. The architecture of PRD is displayed, at a high level, in Figure 1.

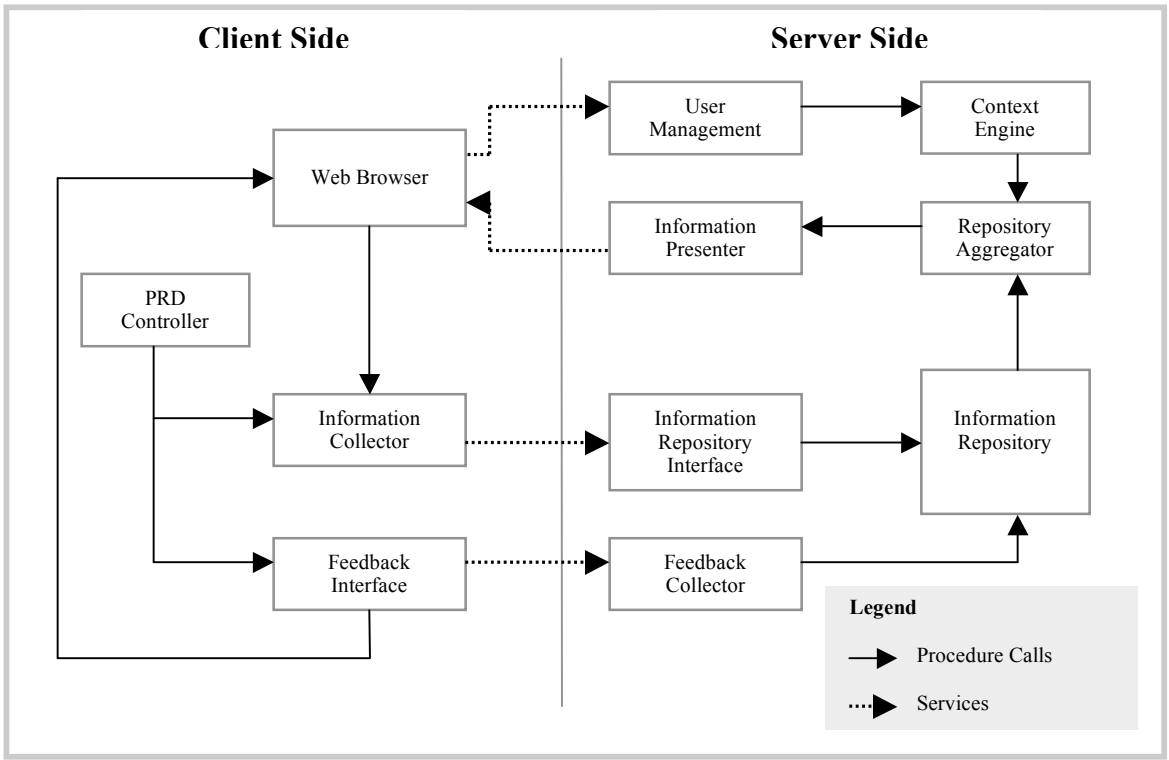

Figure 1. High-level architectural view of PRD

The components within the architecture are described within **Table 1**. These components will be referenced and discussed in the following sections. A reference to a component will be italicized (e.g., *Repository Aggregator*).

Table 1. Description of the components within the PRD architecture.

Component	Function
Context Engine	An engine that stores context based on the particular user.
Feedback Collector	A component on the central server which collects user feedback and adds it to the information repository.
Feedback Interface	This component activates the feedback interface after problem resolution.
Information Collector	This component collects the pages visited by the user, filters any irrelevant results, and sends a list of pages to the central server upon problem resolution.
Information Presenter	A user-interface component which presents results from the information repository to the user. This component also has an interface to search engines.
Information Repository Interface	An interface located on a central server which accepts information from clients
Information Repository	A data structure on the central server which stores the information provided by the client
PRD Controller	This component accepts user input to start and stop a PRD process. This component also stores authentication information
Repository Aggregator	A component which performs operations on the information repository in preparation for presentation to the user
User Management	A service on the central server which provides user-creation, management and authentication capabilities
Web Browser	The normal browser of a user (i.e., Mozilla, Firefox, Internet Explorer, Opera, etc.).

3.3 Social Context

PRD enforces a structure on the social relationships that are to be used within PRD. A social relationship must fall under three possible classes, which are described in detail in this section.

There is a distinction between a *class* and a *context*. A *context* is a set of relationships following certain guidelines (e.g., all these relationships are people within my team working a particular project). However a *class* is a collection of *contexts* where the general guidelines are the same, but the specifics are different. For example, a user can have two team contexts, one context with people working with him on project #1, and another context with people working with him on project #2. Both of these *contexts* will fall under the Team *class*.

Figure 2 shows a representation of how the various contexts and classes coexist. A class is the collection of all available contexts of the same type.

Section 3.3.1 will go into further detail about the *Team* class, Section 3.3.2 will discuss the *Peer* class, while Section 3.3.3 will discuss the *Community* class.

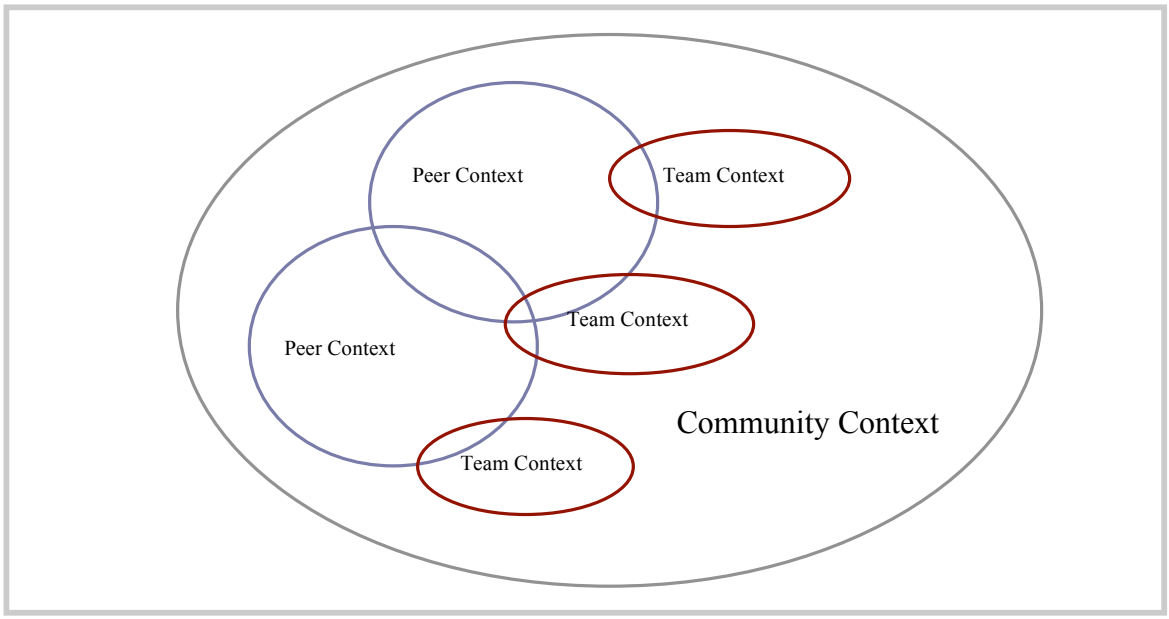

Figure 2. Example of how relationship contexts coexist can with each other.

3.3.1 Team Class

The *Team* class is meant for the group of people with whom a user has a daily working relationship. Generally, a user is collaborating with the people in this class and they all share a common goal. In a corporate environment, an example of a *Team* class would be the members of a team that are developing a piece of software. There is the possibility that a user may have multiple *Team* contexts, with each context referring to a potentially overlapping set of people who are working on a different project.

A team context is relevant because a team shares a common goal and approach to their work. When they encounter problems it will be very similar in both environmental and situational terms to the previous experience of the team members. Typically when this happens, a user must consult a team member who has previously solved the problem, thus running into the problems discussed in Section 2.5.

An alternative is for the user to solve the problem himself by searching for a solution. However, the information stored within a search engine's repository is ranked based on the experiences of an entire world; whose situational characteristics that drive whether they find a page relevant or not can greatly vary from that of a team environment.

Thus, it is expected that the results of a query against the PRD information repository applied with a *Team* context will be the most relevant to the problem at hand, primarily due to the closely connected tasks a team environment fosters.

3.3.2 Peer Class

The *Peer* class is differentiated from the *Team* class by the immediate goal of the people in the relationship. Individuals within a *Team* relationship are working on a single project, and thus share the goal of completing that project. However, those in a *Peer* relationship may have differing goals while working the same general area. If they are working on the same project, then that sort of relationship should be described within the *Team* context.

A *Peer* context is designed to allow an individual to maintain a group of relationships with people who are interested in the same field. For example, an individual working daily with DB2 may have a *Peer* context consisting of people who are DB2 application developers, DB2 support technicians, developers on the DB2 team and the marketing team for DB2. The specific interest of this group of people in DB2 is orthogonal; however, they all share an interest in the general area of DB2. In this way, the concept of a *Peer* context is very similar to an abstraction of a community which shares a common point of interest. However, unlike the *Team* context where external directories (such as LDAP) may be used to identify relationships, the individuals with whom a user chooses to associate have to be manually identified.

The manual creation of relationships is an important defining factor of this class. The reasoning is to introduce an implicit confidence ranking of certain people within the community from a user's perspective. These are the people that the user feels are the "experts" in the field. Due to this confidence, the people that are within a *Peer* context are more influential in a user's information-seeking process and the referral of pages from this *Peer* group is then considered more relevant than those within the larger *Community* context.

Similar to the *Team* context, there may be multiple *Peer* contexts for each user, based on the various areas of their interest.

3.3.3 Community Class

The *Community* class is different from the *Peer* and *Team* classes in that there is only a single *Community* context. The *Community* context is a direct representation of a community of people who share a single interest: the solution to a particular problem. User input is not required to create the *Community* context as all users that have participated in a problem resolution for a specific problem are automatically included.

While there is only a single *Community* context, the membership within this context changes based on the query at hand. The knowledge shared within this context results from previous resolutions to this problem, which only certain people have solved. This is different than the dynamic nature of *Team* and *Peer* classes, as a user may switch between multiple *Team* contexts whose members work in the same areas with different goals. An example would be a user who has a *Team* context for providing technical support on WebSphere and another *Team* context for developing web applications in WebSphere.

The *Community* context is an innovation beyond communities on the World Wide Web (such as forums) in that the data and individuals associated with the community are automatically gathered based on previous information collection. The forums analogy requires a user to manually search the World Wide Web to find relevant information; that is, to pull the information to themselves. The PRD approach pushes the knowledge to the user. The *Community* context also implicitly includes everyone a user may have specified within their *Team* and *Peer* contexts.

3.4 Workflow changes

PRD attempts to minimize modifications made to a user's problem-solving workflow. This is an important feature as users are hesitant to adopt better alternatives should they not be convinced, in advance, that it adds value beyond the additional approach, or it requires increased inconvenience. As PRD is not an established tool, the impact of the latter must be minimized. For this section, we assume that most users, when encountering a problem, utilize a search engine to perform a preliminary search of background information on the problem and its potential resolutions.

Section 3.4.1 will discuss the current workflow taken by users when solving problems using search engines and Section 3.4.2 will discuss the changes PRD makes to this workflow. Finally, the workflow introduced by PRD to collect information from the user to form *knowledge-sets* will be discussed in Section 3.4.3.

3.4.1 Existing User Workflow

Before discussing the workflow changes, the reference workflow used by users to solve a generic problem must be presented. This workflow only includes the user interactions, and is shown in Figure 3.

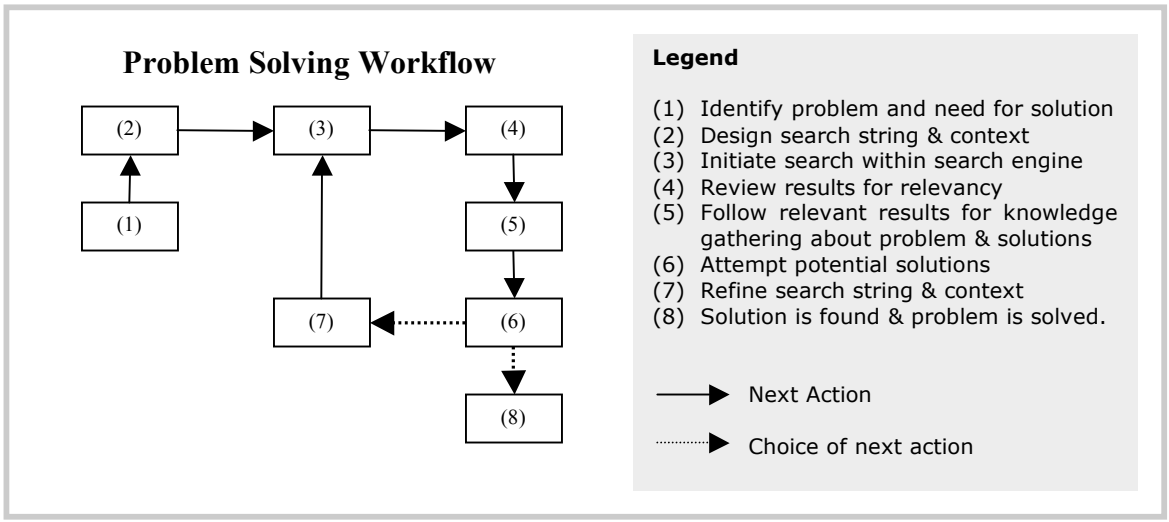

Figure 3. Workflow for solving a generic problem using a search engine.

It is important to note that the iterative process required in problem solving using a search engine is due to two factors:

1) The ability of the user to construct a precise search query, and
2) The relevance of the links returned versus the search query.

An increase in quality in either of these factors will shorten the number of cycles required by the user to find the knowledge they require.

3.4.2 PRD User Workflow

PRD modifies the user workflow shown in Figure 3 by modifying step (3) and adding 2 steps, one before step (3) and one after step (8). The modified workflow, with changes in bold, is shown in Figure 4.

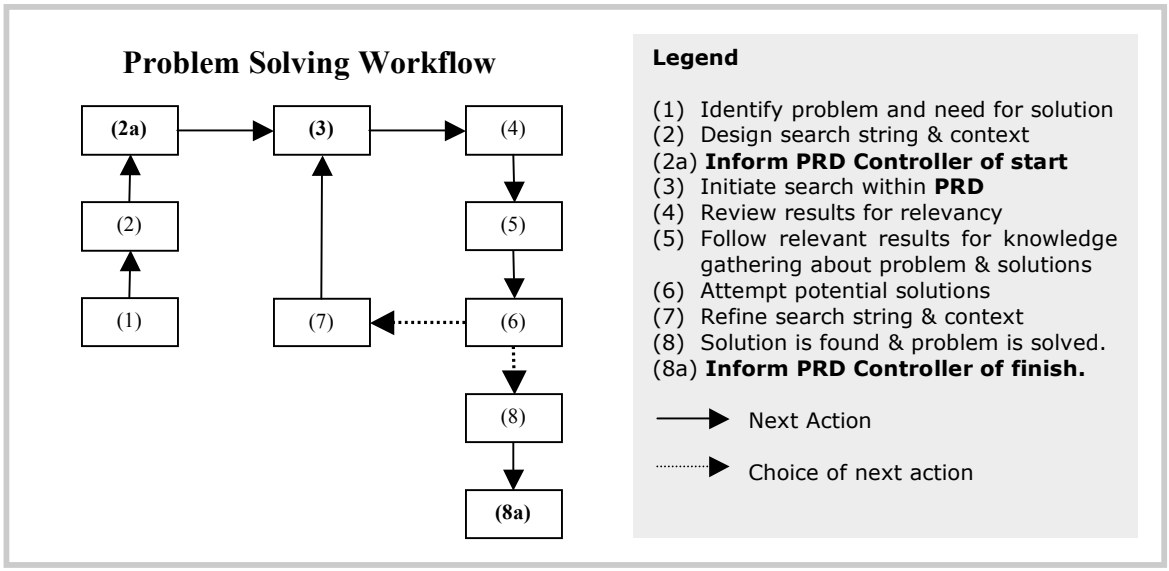

Figure 4. Workflow for solving a generic problem with PRD.

The modified step, shown in step (3), is a substitution of the search engine interface with the PRD interface. This is necessary in order to funnel the user query into the PRD system and show query results from the PRD information repository. The impact of this change is minimized by using an identical query interface as a search engine; that is, to use a search string as the query.

PRD uses traditional search engines in parallel with the pages found within the PRD information repository. This is to introduce new relevant links into the PRD repository, in addition to providing a fallback mechanism that is more familiar to the user.

The two additional steps are minor additions. However, they require an effort by the user, and as such are shown in separate steps. The purpose of these steps is to start and stop the automated collection of information for the PRD information repository.

Efficiency can be increased by coupling steps (2a) and (3) together. Initiating a search within PRD can be considered an initiation of the PRD process. As such, in the PRD prototype, there is no explicit step for the user to notify the PRD controller of a "start"; the user simply initiates a PRD search query.

Likewise, there has been investigation done into further minimizing the impact of step (8a), informing the PRD controller that the PRD process has finished. This has been motivated by the fact that a user, once a problem has been solved, may forget to terminate the PRD process in their enthusiasm to move on to the next challenge. Several ideas have been considered, but none have provided a reliable mechanism to discover that a user has finished solving a problem. This is considered further work. The ideas are as follows:

- Periodically ask the user whether they have finished. Periodicity is governed by an exponential process.
- Display a graphical reminder within the PRD interface or the IDE interface if the PRD process is active.
- Monitor the user workload within the browser and terminate the process given a sufficiently low workload.

Finally, upon completion of the PRD process, an optional feedback workflow is initiated. The feedback consist of a ternary rating system on the scale of (useful, neutral, not useful) for all non-original links visited by the user; that is, pages that have been seeded into the PRD information repository by a previous user. This process is discussed in further detail in Section 3.5.2.2.

3.4.3 PRD Information Collection Workflow

One key feature in PRD is that the information a user utilizes to solve a problem is automatically captured and collected. This transparency in information collection allows the PRD information repository to be built quickly and with minimal user effort. Unfortunately, there is a tradeoff between the ease in which information is collected and a user's knowledge of what is being submitted. If information collection is completely transparent, without the need for user intervention, the information submitted may infringe upon a user's privacy. Conversely, if every piece of information submitted required the user's active agreement, the process would be a deterrent to user adoption. A balanced solution for this tradeoff is discussed in Section 3.4.3.1.

PRD makes the assumption that a user will visit web pages in order to gain insight into the problem at hand, as well as knowledge to solve the problem. We also assume that these web pages will be viewed within a recent web browser with common browser settings. That is, we assume that the browser will have history enabled and set to a time limit that exceeds the time needed to solve the problem. The reason for this is because PRD queries the history store of a browser in order to retrieve the web pages visited during the problem solving process.

In common, recent web browsers, whenever a user visits a webpage certain information about that web page is stored within the browser's history. These characteristics will at least include the page URI, the timestamp when the page was accessed, and the title of the page. The history is then accessible through the browser's user interface. It is also stored within the local file system.

When a user initiates a PRD process, PRD saves the current timestamp within its state. As the user proceeds to solve the problem, PRD is not active (aside for monitoring for a PRD process finished event) and does not contribute any overhead by running a collection process. The overhead in information collection is implicit in the population of the history store by the web browser.

When the user signals that the PRD process is completed, PRD parses the history store to retrieve all pages that were visited between the starting timestamp and when the process was stopped. This set of URIs, which we will call a *knowledge-set*, is the set of information needed to learn and solve the problem from this user. After, potentially, some filtering is performed on the set, as discussed in subsequent sections, the set will be sent to the PRD information repository.

An overview of the operations performed by the PRD information collection process on the client side with respect to the web-browser history is shown in Figure 5. These actions are always performed on a copy of the actual browser history. No changes are made to the browser history.

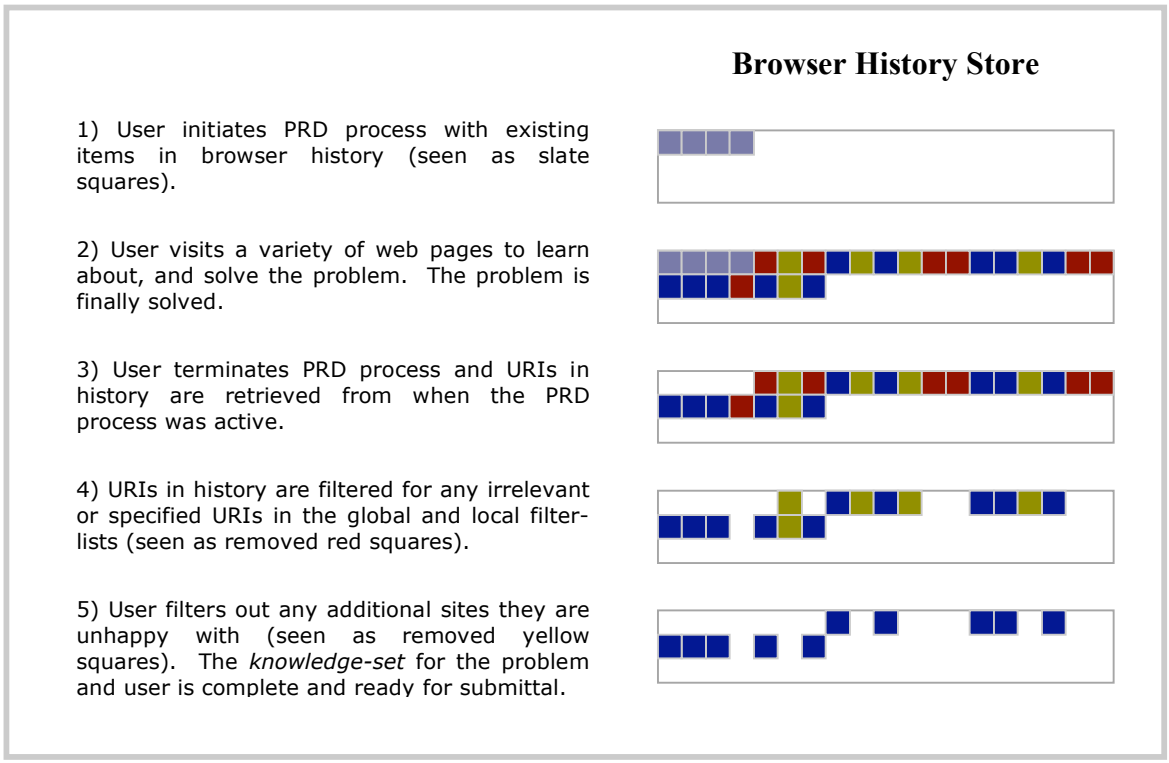

Figure 5. Example of actions on a copy of browser history.

Section 3.4.3.1 will discuss the filtering of pages that may have user-sensitive data, while Section 3.4.3.2 will discuss the framework used to filter irrelevant URIs.

3.4.3.1 Privacy Concerns

Users with an understanding of PRD will have full knowledge that all links they submit will appear under the *Community* context for the particular problem they have solved.

Due to the transparent nature of information collection, users may feel wary about letting PRD have autonomous control to submit a user's web-browser history to a central server. In addition, as the history sent must be tagged with identity metadata in order to be of use for social contexts, the particular visitor to a website can be identified. An example of this may be banking sites, a hobby web site unrelated to the problem at hand, or portions of the dark web.

While this concern may not be justified in some cases (for example the URLs visited within a banking site will generally not contain any session information that can identify the user), it is a concern as it may incite fear, uncertainty, and doubt within a user, thereby reducing their confidence in PRD.

There are three approaches used to solve this problem, one social and two technical. From a social point of view, when a user is solving a technical problem, it is unlikely that they will randomly switch their mental workspace and begin working on another problem (e.g., paying a bank bill). This will minimize the occurrence of this problem; however, it does not alleviate a user's fear that this problem may occur.

The first technical means to solve this problem is to implement a user-defined filter list. This list will include a list of pages, domains, and regular expressions that a user can define never to be sent as part of the collection process. This solves the occurrence of the problem under the assumption that a user will take the time and effort to create this list. However, again, it does not alleviate a user's fear that a page may slip through the filter.

The second technical solution is to display the set of pages to be submitted to the user before the list is actually delivered to the repository. The user is then allowed to prevent certain pages from being sent. This solution provides the user with security and control of what pages can be submitted, as well as a means to satisfy legal requirements for their operating region.

It is recommended that both technical solutions be implemented.

3.4.3.2 Irrelevant Information

A second concern in the data being collected is relevancy of the information. There is considerable difficulty in defining the relevancy of the individual pages visited by the user. It is somewhat simpler to define classes of pages that are not relevant to the search query. These can take many forms, such as advertisements or pages that are not accessible to the general public. The following is a list of common pages that are not relevant:

- Advertisements
- Email sites
- Image files
- PRD system pages
- Search-engine interfaces

These pages will very likely be present within a user's *knowledge-set*. For example, if a user initiates a search-engine query in parallel with the PRD process, the pages for the search engine will be included in a user's set of pages. However, these pages add no value to the search process, as it does not contain any direct information to the problem at hand.

There are also certain categories of pages that may or may not be irrelevant depending on the deployment of PRD. These are as follows:

- Members-only sites
- Intranet URIs

In a worldwide, Internet deployment of PRD, where the members of PRD have no common affiliation, these pages are considered irrelevant. The reasoning behind this is that the information stored in the pages with these restrictions is only available to users who have access permissions. Since, there is not enough information to assume that all users of PRD have sufficient access permissions, inclusion of these sources will reduce the relevancy of the overall system for those that do not have permission.

The exception to this is if it is possible to prove that all users of PRD will have access to the information contained in these sources. A scenario where this can occur is if PRD is deployed within a corporate Intranet. In this case, pages from the Intranet are viable sources of information and should be included.

To solve the problem of irrelevant information, a technical solution is implemented. Two filter lists are created, a server-side filter list and a local filter list. The server-side filter list contains a list of pages, domains and regular expressions that will be used globally by all users of PRD to filter out irrelevant pages. This list represents the technical solution to solve the problem discussed above.

The local filter list, which can be shared within teams (although PRD has not been designed to support this), acts as an additional mechanism to prevent pages from being included within the *knowledge-set*. This additional solution is to provide *Team* or *Peer* contexts with the ability to manually filter out certain websites which have a reputation of providing irrelevant content. An example may be including a regular expression for sites within a corporation's Intranet as including these URIs within a *knowledge-set* would not be beneficial for users outside the corporation.

3.5 PRD Query Handling

Once a user's *knowledge-set* has been identified, the set of URIs is sent to a central server along with the problem the user solved. Thus, the *knowledge-set* is tied to two things, a user and a problem, and is stored within the PRD information repository. Additionally, a problem signature is extracted from the problem to uniquely identify this problem. This problem signature may already exist in the information repository, and if so, this *knowledge-set* expands the pages related to this problem.

When a query is received by PRD, a problem signature is extracted from the query and matched against the data within PRD's information repository. If data about the problem exists, additional queries are created which take into account a user's *Team* and *Peer* contexts. Section 3.5.1 will discuss the generation of problem signatures, while Section 3.5.2 will discuss the ranking algorithms used on the results returned by a query.

3.5.1 Generating Problem Signatures

If similar problems are not grouped together, individually each variant of a problem will have a smaller set of results, which implies a smaller set of relevant results. By drawing together similar problems, the ranking algorithms used will be more effective as outliers are reduced. Problem signatures are vital to group similar problems together.

However, extracting problem signatures from user-constructed search strings is a difficult task. There may be no discernable pattern in the string generation and thus a dependency on structure is unreliable. In order to create problem signatures, we will make some assumptions of a user's behaviour.

First, it is reasonable to assume that when a user encounters an error, they will use the error message as part of the search string. When users attempt to solve errors in web browsers, it is assumed that they copy and paste the entire error string into the search engine query interface. This is a viable assumption since search interfaces generally are not visually large enough to contain an error string and associated stack trace (or other error information). Thus, there is a psychological barrier to including too much information in the search string.

The use of an error message is also aided by the fact that when problems are posted into community forums and discussion groups, the entire error message as well as other trouble-shooting information is included. Thus, users are familiar with using an error message in order to precisely match information on the World Wide Web.

Secondly, users are able to balance between including relevant information into a query and entering too much information. For example, if a user encounters an error with a stack trace, it is assumed that the user will copy the error message and search with that string, rather than the error message and the entire stack trace.

Due to working predominantly with IBM technologies, some further assumptions can be made upon the structure of error messages. Each error message will have a symptom code embedded within it, defined in regular expression from in Figure 6.

```
\\s+([A-Z]{1}[A-Z[0-9]]+[0-9]+[A-Z]{1})\\s+
```

Figure 6. Regular expression definition of a symptom within an IBM error message.

Problem signatures are created for error messages based on extracting a match for the symptom code. This approach can be generalized to extracting an error code from an

error message. In the future, standardized formats such as IBM's Common Base Event log format can be leveraged to extract a normalized problem signature.[32]

3.5.2 Results Ranking

Once the pages within the PRD information repository that correspond to the problem have been identified, a process of ranking the results takes place. There are several ways to perform this ranking, with the most basic being a frequency ranking. If a page has been included in the *knowledge-sets* of several users, that link is thought to have more relevancy than a page which has only been cited by one user. Thus, the preliminary ranking performed is based on frequency.

However, frequency ranking is not fool proof and this section will introduce several ways to improve the ranking through the use of a series of stacked ranking algorithms. The frequency-ranking algorithm is further discussed in Section 3.5.2.1, and the stacked modifiers follows, with the Last-Page Rank modifier in Section 3.5.2.2 and the feedback modifier in Section 3.5.2.3. Finally, additional modifiers are suggested for future work in Section 3.5.2.4.

The order in which these algorithms are stacked is shown in Figure 7.

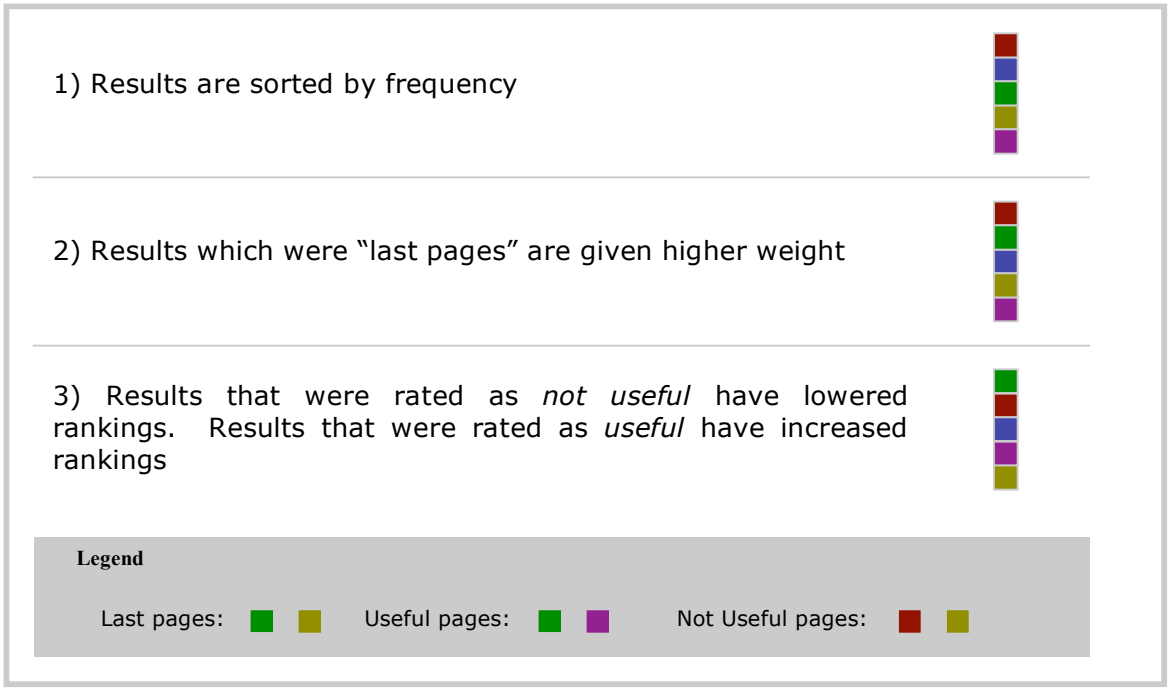

Figure 7. Example of ranking algorithms' stacking order

3.5.2.1 Frequency Ranking

A frequency ranking is a simple starting point to perform relevancy ranking; however, there are some problems with this approach. A simple example that produces an unwanted result is if a particular page is a highly-ranked result on a search-engine query for a particular problem. Everyone who attempts to solve this problem may click this link; however, the resultant page may have nothing whatsoever to do with the problem.

Ignoring the fact that users can filter this page, the PRD information collection process will repeatedly associate this irrelevant page with the problem, thus decreasing the accuracy of our ranking.

Frequency ranking makes the assumption that the more often a page is visited, the more relevant a page is. This leverages the fact that users who find pages (i.e., those that are not already within the PRD information repository) on search engines are more likely to view documents that are higher up in a search engine's result ranking, thus seeding more-relevant data within the PRD repository. This is generally a valid assumption for two reasons. First, users will manually review the results returned by a search engine for relevancy based on the snapshot of the page's content within the results. Secondly, a user implicitly believes that the higher up a page is in a search engine ranking, the more relevant it is to the query (although this is not always true in practice).

However, there are significant problems with this approach. First, users may click multiple links of varied significance during the resolution process. While it is a goal that over a large number of users, the pages with lower significance are clicked fewer times, this is not always the case. One reason is that there may be some social block which affects user behaviour. For example, perhaps the unwanted link is always returned as the first result in a search for the query, and users intuitively end up clicking that first search result. Another reason is that it cannot be assumed that there will in fact be a large number of users, especially in *Team* contexts.

Another problem with a frequency-ranking approach is that of a *self-fulfilling prophecy*. As users see that certain pages are higher in the ranking, they will follow these links and thus, regardless of the relevancy, elevate the ranking of these documents.

Frequency ranking is a good starting point for ranking; however, some modification of the results are required in order to improve the accuracy of the attempt to return relevant results.

3.5.2.2 Last-Page Rank Modifier

As a user iteratively solves a problem, they attempt potential solutions before finally discovering the actual solution. During this process, they must continue their learning process, which exhibits itself from an information-collection perspective as viewing web pages.

Once the problem-resolution process has completed, it is very likely that the last page visited by the user contains the key information that they were missing in order to solve the problem. Since there is important information on this last page, it stands to reason that this last document should be at the same level or more relevant than the other pages viewed by the user.

A Last-Page rank modifier is added to the last page to capture this assumption. However, the weight of the last page must be greater than the weights of the other individual pages (currently weighted equally), in order to differentiate it from the other pages.

The algorithm to rank and apply the Last-Page rank modifier is shown in Algorithms 1 and 2. Algorithm 1 selects all the relevant pages of the problem being queried from the PRD information repository and assigns a default weight of 1 to every page. The last page for each solved problem instance is given an increased weight as defined by the parameter LP. Finally, the results are aggregated by Algorithm 2 and the weights of each page converted into ranking for each page.

Algorithm 1. Calculate the rank of results by including Last Page modifier

Define $p_{i,j}$ as the j^{th} page used in solving the problem for the i^{th} time and $0 < j < n(i)$

Define $w_{i,j}$ as the weight for page $P_{i,j}$.

Define $LP > 1$ as the weight modifier for last pages

```
function calculate_ranking() {
  select all pages p_{i,j} used in solving previous instances          1
  foreach i and j { w_{i,j} = 1 }                                      2
  foreach i { w_{i,n(i)} = LP * w_{i,n(i)} }                           3
  group(p,w)                                                           4
}
```

Algorithm 2. Aggregate pages from multiple resolution instances.

Define $p_{i,j}$ as the j^{th} page used in solving the problem for the i^{th} time and $0 < j < n(i)$
Define $w_{i,j}$ as the weight for page $P_{i,j}$.
Define r_u as the u^{th} result for the query and $0 < u < m$
Define k_i as the rank for r_i

```
function group(p,w) {
  while p contains elements {                                          1
    consume all p_{i,j} of the same page and store page in r_u         2
    k_i = □w_{i,j} associated with consumed p_{i,j} in this iteration  3
  }                                                                    4
  foreach k_i { k_i = k_i^-1 }                                         5
  sort order of r_i by magnitude of k_i                                6
}
```

3.5.2.3 Feedback Rank Modifiers

One approach to removing unwanted URIs from future results is to utilize a user's cognitive abilities. When a PRD process is terminated by the user, PRD asks the user to provide optional feedback on the pages they used to solve the problem. Feedback is only requested if the user had visited documents that other users have previously viewed during their most recent problem-resolution process. The data gleaned from this process is then used to modify the ranking of a link result.

The feedback is based on a ternary system; the user can rate a link as *useful*, *neutral* or *not useful*. The outcome for this ranking is twofold. The positive feedback enables less popular (i.e., less viewed), but more-relevant links, to quickly ascend to a higher position in the results. The negative feedback diminishes the effect of a positive frequency ranking on pages that are not as relevant to the query.

A drawback to the acceptance of this approach is that it requires an active effort on the part of the user. However, as the effect of this action is beneficial to a user's relationship groups and in particular their team, it is hoped that they will actively seek to improve the accuracy of the results through feedback.

Once a user enters their feedback, the PRD information repository is updated with these modifiers. Subsequent queries of the repository with the same problem will show a modified rank including this user's feedback. The integration of users' feedback on the rank of the results is a modification of Algorithm 2. The modified algorithm is shown in Algorithm 3.

Algorithm 3. Modified `group(p,w)` algorithm to apply Feedback modifiers

Define $p_{i,j}$ as the j^{th} page used in solving the problem for the i^{th} time & $0 < j < n(i)$
Define $w_{i,j}$ as the weight for page $P_{i,j}$.
Define r_u as the u^{th} result for the query and $0 < u < m$
Define k_i as the rank for r_i
Define $U > 1$ and $NU < 1$

```
function group(p,w) {
  while p contains elements {                                          1
    consume all p_{i,j} of the same page and store page in r_u         2
    k_i = □w_{i,j} associated with consumed p_{i,j} in this iteration  3
  }                                                                    4
  foreach r {                                                          5
    if r_u has useful ratings                                          6
      do { k_i = k_i * U } for every useful rating                     7
    if r_u has not useful ratings                                      8
      do { k_i = k_i * NU } for every not useful rating                9
  }                                                                   10
  foreach k { k_i = k_i^-1 }                                          11
  sort order of r_u by magnitude of k_i                               12
}
```

The modifications in this algorithm are within the lines ranging from 5 to 10. The additions check for any *useful* and *not useful* feedback for each page and increase or decrease the weight of each page according to specified parameters. Note that a page may have both *useful* and *not useful* feedback.

Both `U` and `NU`, the parameters used to modify *useful* and *not useful* feedback are linear operators. The reasoning for this is because the feedback modifiers should be most accurate at the *Team* class level. Thus, an alternative of using an exponentially decreasing modifier would not be adequate for the potentially low number of feedback responses in a team environment.

3.5.2.4 Other Rank Modifiers

Several other modifiers have been considered during the research performed on this topic. However, at this point it is infeasible to implement these due to lack of resources and/or architectural restrictions. The investigation of these modifiers and their integration points within PRD is future work. A brief explanation of the possible modifiers are listed below:

- A user-defined list of higher-weighted domains (e.g., product documentation or a knowledge base)
- How recently a particular URI was added to the repository. By promoting more recent URIs, feedback will be attained for the link and the relevancy of the URI can be better computer. This is especially useful for URIs from domains that are not common for a particular problem.

- Whether the page spawned a fork in the user's knowledge-seeking path. This requires monitoring of the sequence of pages followed by the user and referral information for each page.
- The document type (e.g., a webpage is more accessible within a web browser, and may be easier to comprehend than a Powerpoint presentation).

4 Prototype

A prototype has been built to validate the design and illustrate the functionality of our proposed approach. This prototype implements the detailed architecture as described in the previous sections. The sections previously discussed as future work have not been implemented.

The prototype has been implemented with Apache, PHP, MySQL and as an Eclipse plugin. Apache, PHP and MySQL were used due to familiarity with the technologies as well as for fast prototyping capabilities. An Eclipse plugin has been created to encapsulate much of the PRD functionality and to increase adoption of the prototype. Eclipse is a commonly-used, open-source IDE used by the target audience of this approach. Table 2 describes the mapping of the architectural components to software components in the prototype.

Table 2. Software representation of the architectural components within the prototype

Component	Software Representation
Context Engine	PHP
Feedback Collector	PHP
Feedback Interface	Eclipse plugin/Web browser
Information Collector	Eclipse plugin
Information Presenter	Web browser
Information Repository Interface	PHP
Information Repository	MySQL
PRD Controller	Eclipse plugin
Repository Aggregator	PHP/MySQL
User Management	PHP/MySQL
Web Browser	Web browser

Section 4.1 will describe the mappings between the architectural constructs described in Section 3.2 with the prototype.

4.1 Architectural Mappings

The following sections will describe, in detail, the software representation of the components.

4.1.1 Eclipse Plugin

The PRD controller functionality has been built into an Eclipse plugin. Eclipse is an open-source IDE whose goals include "providing an extensible development platform and application frameworks for building software".[33] The intent is that Eclipse is an example of an IDE widely used by the target audience of PRD; that is, programmers and software engineers. These are the same users who work within team environments and encounter the class of problems for which PRD is attempting to streamline resolutions.

4.1.1.1 Manual User Controls within Eclipse Plugin

The created Eclipse plugin includes functional mappings to the start and stop actions of a PRD process. These events are activated by the user through two alternatives: 1) a button press on the toolbar, or 2) a selection within a PRD menu item. Figure 8 shows a screenshot from the prototype of the control mechanism. The section labeled **A** corresponds to alternative 1, while the section labeled **B** corresponds to alternative 2.

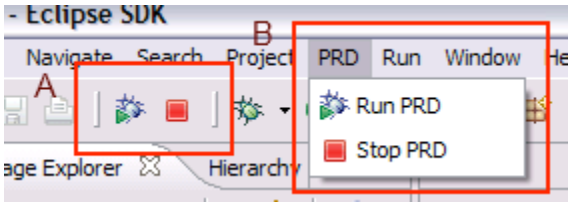

Figure 8. Screenshot of the PRD control mechanism within the Eclipse plugin.

When a user initiates a PRD process they are presented with a dialog box within their working environment (i.e., Eclipse). This dialog box is used to query the user for a search string, or more typically an error message. Figure 9 shows a screen shot of this dialog box. Once the user presses **OK**, a browser will spawn, external to Eclipse, and load the PRD client.

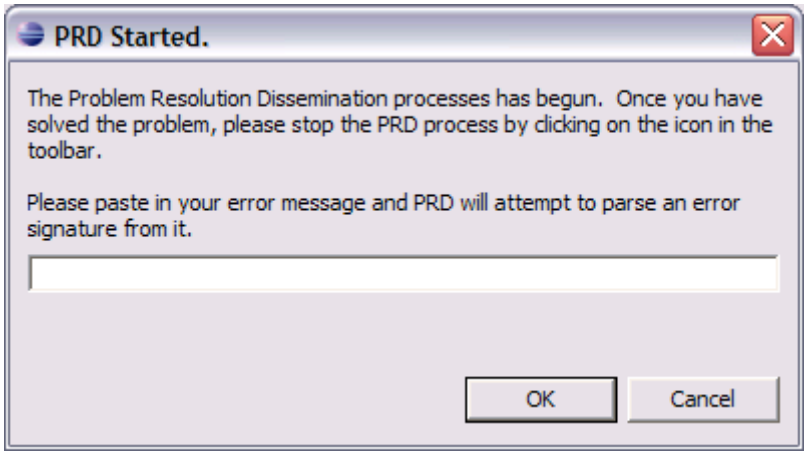

Figure 9. Screenshot of error query interface within the Eclipse plugin.

Finally, the stop control presents a question box to the user asking if they are willing to provide additional feedback on their resolution process. This is the initiation of the process described in Section 3.5.2.3. If the user selects **No**, the process will not be initiated. A screenshot of the question box is shown in Figure 10. If the user presses **Yes**, a browser will spawn, external to Eclipse, and load the feedback for the recently finished PRD process.

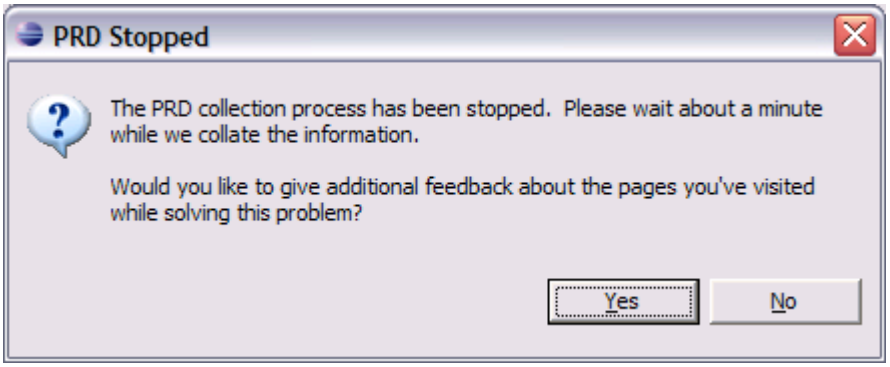

Figure 10. Screenshot of feedback initiation query within the Eclipse plugin.

All of these user controls have been built using the Eclipse API.

4.1.1.2 Information-Collection Mechanisms

The Eclipse plugin also implements the information-collection mechanisms of PRD. Currently these mechanisms have been implemented under the condition that the user uses a Mozilla-based browser or Internet Explorer. For the latter, Internet Explorer 6.0 must be used as this was the only version tested. If Firefox is used as the Mozilla-based browser, a version within the 1.x family of Firefox must be used. This is because Firefox is moving from a Mork-format database to a SQL Lite database in version 2. Similarly, the Mozilla family is performing the same change in its 2.x family.[34]

Information collection is initiated when the user starts PRD as shown in Figure 8. However, the only action performed at this stage is to store the timestamp of when the PRD process was initiated. The next action performed for information collection occurs when the user has manually terminated PRD. At this point, the browser history is found (as specified by the user in a configuration file) and parsed. The objective of this parse is to find all pages visited since the initial timestamp.

During the parsing process, in addition to the URI visited, PRD also stores the title of the page visited if available. This is to provide a more meaningful visual representation of the URI for future users.

In the future, additional history parsers can be written to support other popular browsers such as Opera. This task is trivial in nature provided that the browser-history data structure is known.

4.1.1.3 Information Transport to Server

Once the visited URIs for the PRD session have been parsed, they are sent to the central server. This process is currently done through a simple approach. A URL is constructed, with the visited URIs (and titles) encoded in its query string. This URL is then opened using Java networking. Authentication information is also passed which will be discussed in Section 4.4.3.2.

In the future, this implementation will be refactored to utilize more of a services-oriented approach.

4.1.1.4 Configuration File

A configuration file is used by the plugin to store certain state information that may vary on individual clients. This data includes location of the browser history as well as login information for the PRD client. While it is possible that some of these configuration parameters are discoverable from the user's operating environment, a configuration file is used for ease of implementation.

4.2 PRD Client

The goal of the PRD Client is to integrate a web-form equivalent of the interface used by a user to perform a query on their search engine, with the results of the search coming from the PRD information repository.

Figure 11 shows the client screen displayed once the user has entered the error into the dialog box shown in Figure 9. The error entered by the user may be sufficiently generic that there may be several errors that match the query text. This situation is handled by asking the user to choose the specific error they have encountered.

A possibility, not shown in Figure 11, is if the error message entered by the user matches existing ones in the database, but is of an entirely different nature. In this scenario, it seems that the proper course of action would be to create a new type of error for the entered error message.

The implementation of this feature is trivial; however, it introduces the possibility of multiple problems resulting from an identical error message. Typically, this would be expected due to the different environments in which a problem can occur; however, in PRD these scenarios are captured through the use of separate *Team* contexts rather than separate problems. Thus, the ability to spawn a separate error from an error message that is already in the PRD information repository has not been included.

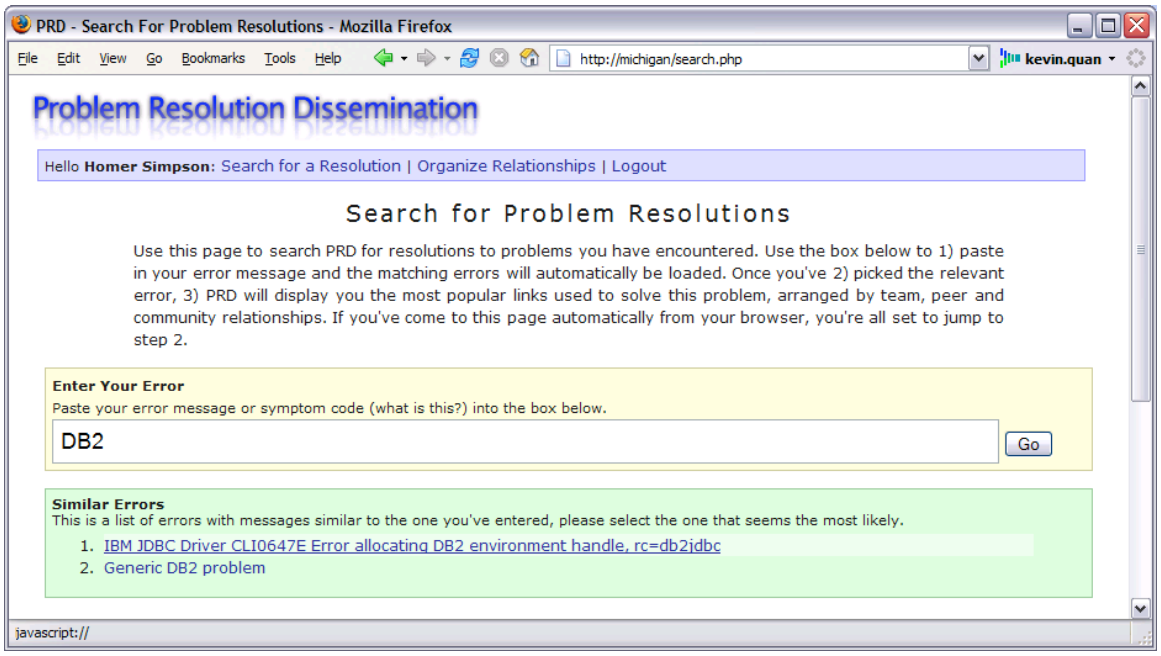

Figure 11. Screenshot of user interface to pick a specific problem in the PRD client.

Once the user has selected the correct error message, the error signature is preloaded into the Web search form (which hooks into a search engine of the user's preference) and the results from the PRD repository are loaded using AJAX calls. An example of this is seen in Figure 12.

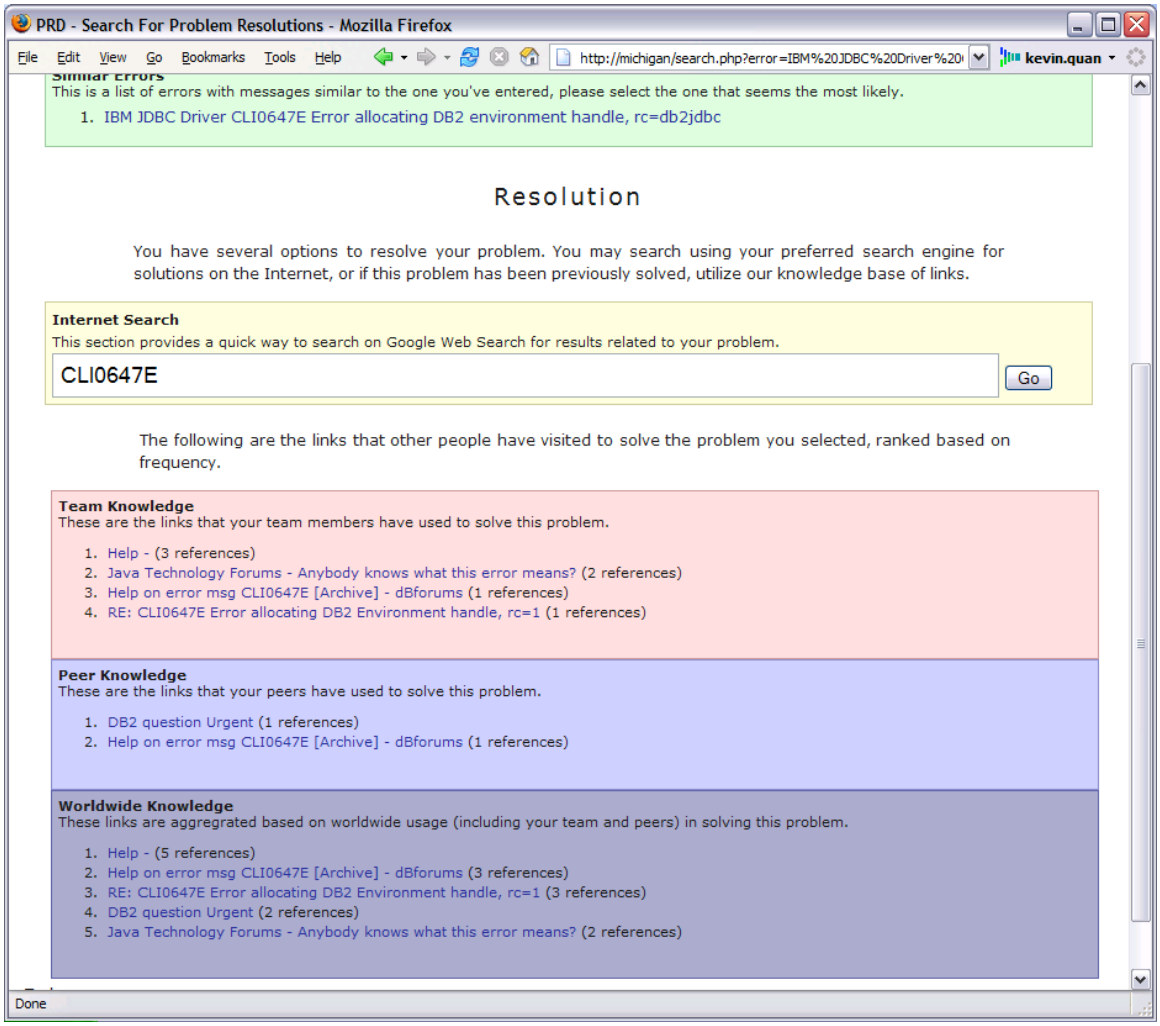

Figure 12. Screenshot of results from the PRD repository within the client.

As illustrated in Figure 12, the results from the *Team*, *Peer* and *Community* contexts are separated. The *Team* context has greatest priority and is thus closer to the top of the screen. This is followed by the *Peer* context and finally the *Worldwide Knowledge*. In the prototype, the *Community* context is referred to as *Worldwide Knowledge* as the latter term is less ambiguous and better describes the intent of that context.

The results in each context are ranked based on the frequency of selection and the results are limited to the top 5 (when more than 5 exist). Not shown in this screenshot is any modification of the ranked order based on the feedback mechanisms discussed in Section 3.5.2.3.

Future work in this area will include the ability to click on a team member who has solved the problem and see the list of links they used to solve the problem. Additionally, more-personalized user-interface functionality will be included, such as the ability to show/hide certain contexts by default, and to change the number of results shown in each context.

4.3 Database Structure

The PRD information repository is physically stored within a MySQL database. When a query is initiated by the user within PRD, the database is queried. The grouping of results within social contexts are performed using `JOIN`s and frequency ranking is performed using `GROUP BY` and `ORDER BY` operators. The schema of the database is shown in Figure 13.

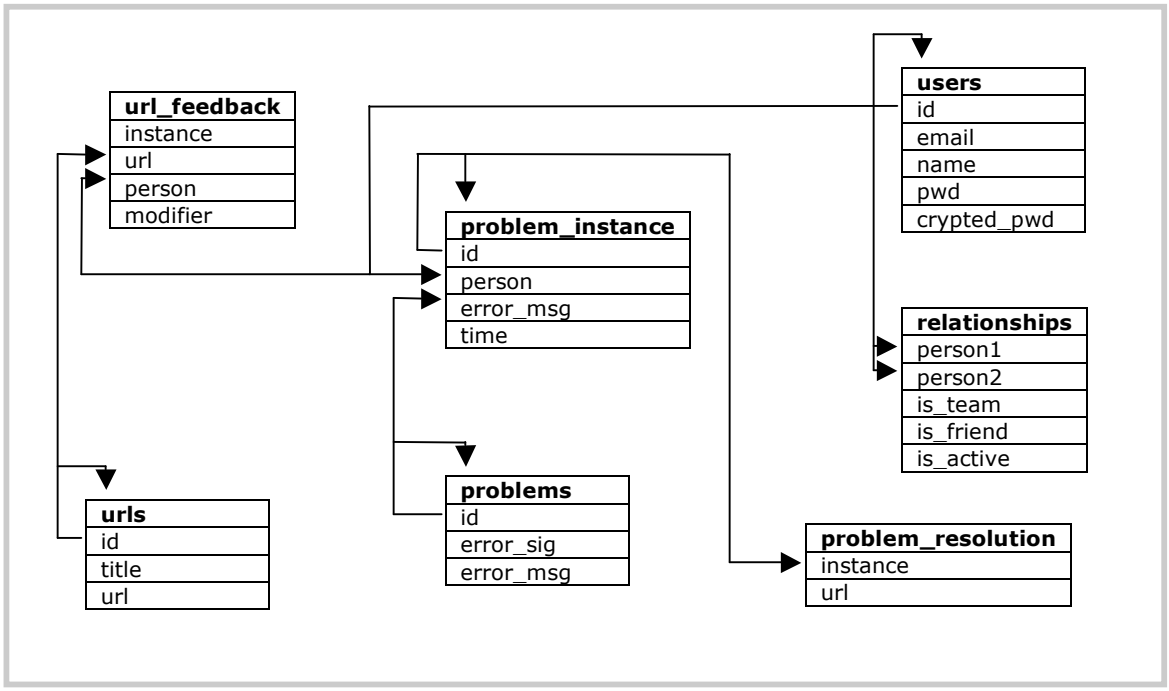

Figure 13. Schema of PRD information repository.

Each of the tables are described in further detail in the following subsections.

4.3.1.1 Users Table

The `users` table stores the user data for PRD. For the prototype, the amount of information stored is limited to the minimum required. Table 3 describes the fields in further detail.

Table 3. Description of fields within the `user` table.

Field	Type	Description
`id`	`integer`	The primary key for this table.
`email`	`varchar(64)`	The email address for the user, which is used as a unique username of the user.
`name`	`varchar(64)`	The nicely formatted full name of the user as entered by the user.
`pwd`	`varchar(24)`	The plain text password of the user
`crypted_pwd`	`varchar(32)`	The hashed password of the user in order to facilitate authentication.

It is acknowledged that storing the plain-text password is bad practice. It is only included within the prototype to facilitate the implementation and testing. This field would not be included within a production system.

4.3.1.2 Relationships Table

The relationships table, whose columns are described in Table 4, stores pairs of relationships and whether they belong to *Team*, *Peer*, or *Community* context. A *Team* context is defined if `is_team` and `is_active` are set to `t` while `is_friend` is set to `f`. A *Peer* context is defined if `is_friend` and `is_active` are set to `t` while `is_team` is set to `f`. Finally, a *Community* context is set if `is_active` is set to `f`. This somewhat-unintuitive system is implemented to facilitate the sharing of structure within queries, and as such is a candidate for refactoring within the implementation of a product.

Additionally, `person1` is always the user that is logged in. This intention is made explicit as relationships are not always mutual.

The primary key for this table is { `person1`, `person2` }, while `person1` and `person2` are both foreign keys from the `users` table.

Table 4. Description of fields within the `relationships` table.

Field	Type	Description
`person1`	`integer`	The current logged-in user.
`person2`	`integer`	Another user on PRD.
`is_team`	`enum('t','f')`	Boolean flag that specifies whether the relationship is a team relationship.
`is_friend`	`enum('t','f')`	Boolean flag that specifies whether the relationship is a peer relationship.
`is_active`	`enum('t','f')`	Boolean flag that specifies whether the relationship defined is active or not. If the relationship is not active, the relationship pair is defined as a community relationship. The utility of this flag is to remove the need for tuple deletions.

4.3.1.3 URLs Table

The `urls` table stores URI entities. Table 5 describes the fields in further detail.

Table 5. Description of the fields within the `urls` table.

Field	Type	Description
`id`	`integer`	The primary key.
`title`	`varchar(255)`	The title of the page referenced by the URL stored in this tuple.
`url`	`varchar(255)`	The actual URI.

4.3.1.4 Problems Table

The `problems` table is used to store problem entities. A problem entity is received from the user in the form of an error message. As users may deviate from a common form for copying error messages, an error signature is extracted from the submitted error message. Currently in the prototype, the error signature is of the form defined in Figure 6. Table 6 describes the fields in further detail.

Table 6. Description of fields within the `problems` table.

Field	Type	Description
`id`	`integer`	The primary key.
`error_sig`	`varchar(255)`	The signature of this particular problem.
`error_msg`	`text`	The user submitted error message text.

4.3.1.5 Problem-Instance Table

The `problem_instance` table stores a particular instance of a problem encountered by a user. This corresponds to one PRD process being initiated. The field `person` is a foreign key to the `users` table, and the field `error_msg` is a foreign key to the `problems` table. Table 7 describes the fields in further detail.

Table 7. Description of fields within the `problem_instance` table.

Field	Type	Description
`id`	`integer`	The primary key.
`person`	`integer`	The user that encountered this problem
`error_msg`	`integer`	The problem that was encountered.
`time`	`timestamp`	When this instance of the problem was encountered.

4.3.1.6 Problem-Resolution Table

The `problem_resolution` table stores the URIs visited during the resolution process. The field `instance` is a foreign key of the `problem_instance` table while the field `url` is a foreign key of the `urls` table. There is no primary key as the tuples do not have to be unique. Table 8 describes the fields in further detail.

Table 8. Description of fields with the `problem_resolution` table.

Field	Type	Description
`instance`	`integer`	The problem instance being solved.
`url`	`integer`	The URI visited.

4.3.1.7 URL-Feedback Table

The `url_feedback` table stores feedback given by a user about the URIs used to solve a particular problem. Based on the ternary system described in Section 3.5.2.3, the `modifier` field is given a value in the set { `-1,0,1` }. The field `instance` is a foreign key to the `problem_instance` table, the field `url` is a foreign key to the `urls` table, and the field `person` is a foreign key to the `users` table. The primary key is { `instance`, `url`, `person` }. Table 9 describes the fields in further detail.

Table 9. Description for fields in the `url_feedback` table.

Field	Type	Description
`instance`	`integer`	The problem that was encountered.
`url`	`integer`	The URL that was used to solve the problem.
`person`	`Integer`	The user that encountered this problem.
`modifier`	`integer`	The modifier based on the feedback given.

4.4 User Management

Central to PRD is the concept of an individual user using the system. This is implicitly required in order to provide the social mechanisms. Section 4.4.1 describes how users are created, Section 4.4.2 describes how relationships are formed, and finally Section 4.4.3 describes the authentication used to securely identify users.

4.4.1 User Creation

A user can be created by utilizing a web form on the PRD client. A user is uniquely identified by their email address. Confirmation of an identity is performed by a user-selected password. A screenshot of the web form used to create an account in PRD is shown in Figure 14.

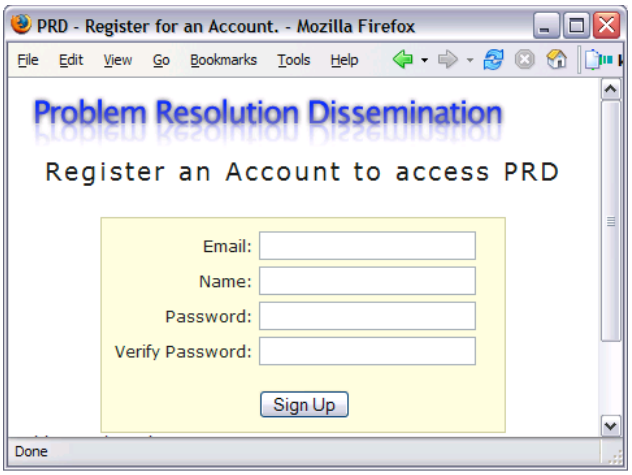

Figure 14. Screenshot of user creation web form in the PRD client.

This action must be performed before the Eclipse plugin can be used, as the user information is required in the configuration file. An alternative that can be used to add a larger number of users quickly (for example, an entire team or department) is to construct

SQL statements and insert the users into the database directly. A second alternative, which would require significant work on the authentication model, is to link user management with a directory such as a company's LDAP directory. The benefit of this is discussed in Section 4.4.2.

4.4.2 Relationships Management

A user's social context can be managed through the PRD client's graphical mechanisms. Users specify contacts to add at a *Team* or *Peer* context level by entering their email address. It is expected that users will use their work (i.e., professional) email with PRD and as such there will be no ambiguity with users having multiple email addresses. Additionally an AJAX-enabled auto-completion mechanism is used to help users fill out emails.

Note that an email address serves as a unique identifier for a user, and can be generalized to other mechanisms (e.g., a username or a telephone number). However, it is assumed that a user's professional email is their most widely-known, and unique form of identification. Thus, in an attempt to be user-friendly, the prototype uses an email address.

There are two main concerns with this approach. First, adding people to a user's context takes time, even if their email addresses are known. This problem can be solved by linking the *Team* context to a company's LDAP directory, which would automate the addition of a large number of existing relationships. However, this would only be an addition to the base relationship-management system as there may be temporary relationships (e.g., per-project relationships) that are not captured by such a directory.

The second concern is from a privacy perspective as all users of the system can potentially see email addresses for the users of PRD. If PRD is deployed on a worldwide scale, as opposed to an Intranet, this would have to be redesigned to alleviate the privacy considerations.

Once a relationship has been established between the logged in user and another, an AJAX-enabled drag-and-drop interface is used to move a relationship between the various contexts. Currently the prototype supports a single *Team* context and a single *Peer* context, and has not implemented the ability to have multiple contexts in each class. Figure 15 shows an example of the relationship-management section of the PRD client.

Figure 15. Screenshot of relationship-management screen within the PRD client.

4.4.3 Authentication

Authentication is required for all queries and information submissions. The requirements for security are slightly different in both of these scenarios; however, the ability to concretely identify an individual is required in both cases.

When a query is initiated, the identity of the person performing the query must be known in order to apply that person's relationship contexts to the query. While the ability to identify the user is required, the need for security in this case is weaker (i.e., properly identify who the user is). Security of the identity protects a person's relationship information from attackers. For the general case of users, this may not be of great importance; however, this may be a key issue if a secret project is being worked on by a secret team.

During information submissions, the ability to securely identify the user is important. Identification is necessary in order to attribute the information being collected to a particular user, thus enabling others who have relationships to this user to benefit from the submitted information. Security is also a necessity in order to prevent the addition of invalid information to the repository. This does not have a great deal of effect on the *Community* context; however, it may be of great impact at the *Team* context level.

4.4.3.1 Security Model – PRD Client

As the PRD client is viewed within a web browser, the security model being used relies heavily on the web-browser infrastructure. A user is authenticated through a username

(email) and password combination. This information is stored locally by utilizing cookies. Cookies are sent with each HTTP request to the server, and verified on the server side.

Data stored locally within a cookie is in plain text; thus, if a user's real password is stored within the cookie, it is at risk. This liability is lessened by storing a cryptographic hash of the password locally instead of the real password. Similar to above, the user and hashed password is sent to the server on each request; however the server now checks the password against a runtime-computed hash of the real password. In the future this will be changed as the real password will no longer be stored within the database. Since the same key is used, an identical hashed password will pass the authentication test.

There is currently no security model on AJAX requests within the PRD client. This is a specific design consideration in order to facilitate future API functionality (e.g., different types of clients). However, if authentication is required, it is fairly simple to implement. For example, the username and hashed password may be sent (along with the remaining parameters) in each AJAX request. The server-side processes would have to be modified to authenticate the user.

Currently the PRD client does not operate over SSL. In the future, this may further enhance the security if this is implemented.

4.4.3.2 Security Discussions - Eclipse Plugin

As the Eclipse plugin performs the information submission, authentication of the user at that stage is also required. This is done by including the username and hashed password within the information submission. This is a first step towards protecting the user's identity. However, a detailed threat modeling of the plugin and the security required to protect it is not in the scope of this research.

It would be inconvenient for the user to require them to enter their username and password each time they perform a problem-resolution. This process is streamlined by requiring the user to enter their user information for PRD within the plugin configuration file.

The password in the configuration file is in plain text as it is laborious to require the user to hash their own password with a particular key. However, this does present a security risk in the event that an attacker has access to the machine's file system.

5 Validation

In the following sections, techniques to validate the PRD approach will be discussed. The PRD approach is compared against the use of search engines. Search engines were chosen as the basis for comparison against PRD as it is the most similar of the approaches discussed in Section 2 (Background). In fact, PRD can be thought of as an evolution in the way users can search for information to solve a problem.

The following section is arranged as follows. Section 5.1 discusses how to measure the effectiveness of PRD over a single problem-resolution, Section 5.2 compares the efficacy of PRD versus search engines as the number of problem-resolution requests increases, while Section 5.3 discusses the integration of this measurement over a larger experimental scope.

5.1 Comparing Efficiency Over a Single Attempt

In this section, the future social benefits of a problem-resolution performed within PRD are ignored. Instead, a comparison is made between a single, atomic problem-resolution performed using PRD and performed using a search engine. Note that while future social benefits are ignored, the presence of past social-context information being available within the PRD information repository is not precluded.

The criteria in which the processes will be examined are the following:

- The time it takes to find a solution,
- The work required to find a solution.

In this section, we use the term *work* to describe both actions that a user must take and the cognitive processing that the user must perform in order to determine whether links are relevant or not.

The comparison will be made with the following assumptions:

- The problem is solved in both cases,
- The user is familiar with the usage of both PRD and a search engine.
- The user using PRD submits information that was collected and related to the problem they are solving.

An individual PRD process can be classified as one of two types. The process can either be a problem-resolution attempt of a problem that has no existing data within the PRD information repository (i.e., first attempt), or it can be an attempt to solve a problem that has existing data within the PRD information repository (i.e., subsequent attempts). These two cases will be discussed separately in the following sections.

5.1.1 First Attempt

In the initial attempt at solving a problem, due to the fact that there is no data within the PRD information repository concerning the problem, the user gains no immediate benefit

from using PRD. In fact, the user workflow will degenerate to the previous approach of using a search engine.

While PRD integrates a web search engine within the process, the fact that PRD must be first checked and found to not contain any information involves more work and time consumed for the user. Thus, in the first attempt, the use of PRD costs more when compared to a search engine.

We can put a reasonable, quantifiable limit on this increased cost. Typically, it will be substantially less than the time and work required to solve the actual problem. As such, we will refer to this as an incremental cost of the first attempt. Note that this incremental cost may extend beyond the first attempt, but will be a decreasing function of the number of attempts, as described in Section 5.2.

5.1.2 Subsequent Attempts

In this section, we assume that in the initial attempt as well as all attempts prior to the current attempt, information was added into the PRD information repository regarding the problem being solved. Thus, the PRD information repository is not empty for the current problem resolution.

In these attempts, as was in the initial attempt, there are additional actions performed by the user beyond that of using a search engine. This work was described under the PRD User Workflow in Section 3.4.2. Note that this work is fixed and does not vary with the number of problem-resolutions performed for this problem.

The work that does vary in terms of the PRD process is the cognitive requirements from the user. As more information is populated within the PRD repository, the results displayed become more relevant; a user will have to follow and understand fewer links before solving the problem than in the search-engine approach. This results in an overall decrease in time and work.

In this section, there is no distinction between how many previous problem resolutions have been attempted prior to the current attempt. Note that the time and work required in a PRD attempt may still exceed that of one done in a search engine due to there being insufficient existing information within the PRD repository. Section 5.2 will examine this in more detail.

Table 10 defines several metrics to quantitatively measure the above argument. Note that while work in the form of actions such as clicks can be measured, the work a user performs in determining whether a link is worthwhile is more difficult, but should not be worse than in the use of a search engine. Both of these forms of work have a direct consequence in the form of time. Thus, they will be grouped together with the time metric.

Table 10. Metrics to evaluate effectiveness of PRD.

Metric Name	Description

$TotalTime_{old}$	The total time needed to solve a particular error using the traditional approach of performing web searches.
$TotalTime_{new:i}$	The total time needed to solve a particular problem using links returned by querying the PRD information repository. This assumes that `i` previous attempts have been made to solve the problem.
`NewStepsTime`	The time needed to perform all additional tasks that are not performed using search engines. Examples include starting and stopping the PRD process as well as providing feedback.
`ProblemSolved`	A boolean flag that is true if the problem was solved by the PRD approach.
$Time_{PRD:i}$	The time taken to solve a problem using PRD on the i^{th} attempt. This is defined as $TotalTime_{new:i}$ + NewStepsTime.

From the metrics described in Table 10, there is value in the PRD process in terms of efficiency if the expression shown in Figure 16 evaluates to true.

Figure 16. Boolean expression to verify efficiency increase in PRD approach.

$$ProblemSolved \;\&\&\; (TotalTime_{old} > (TotalTime_{new:i} + NewStepsTime))$$

Note that the expression in Figure 16 only evaluates the presence of efficiency improvement in the PRD approach over the traditional approach. The value of providing context-relevant results to future users is not captured. This benefit will be discussed in the next section.

5.2 Comparing Efficiency Over Multiple Attempts

The relevancy of the results returned by PRD increase as more and more people solve a problem. This is due to several factors:

- A larger number of people increase the network effect (i.e., frequency) on the most-relevant links.
- A larger number of people increase the usage of the feedback rank modifier (as well as other modifiers), which improve the relevancy accuracy.
- The coverage of links associated with the problem is greater as the number of people increases.

The time and work required decreases as the relevancy of PRD increases. This is intuitive as increased relevancy allows the user to find the needed information more quickly and thus solve the problem more quickly. The operational impact of this is that fewer links need to be followed and analyzed.

The time required to solve a problem with PRD in theory follows a decreasing curve dependent on the number of resolution attempts. Comparatively, the time required to solve a problem with the search engine approach is constant as there is no information

passed between each query. Figure 17 shows an abstract graphical representation of the relationship between the two approaches to problem resolution.

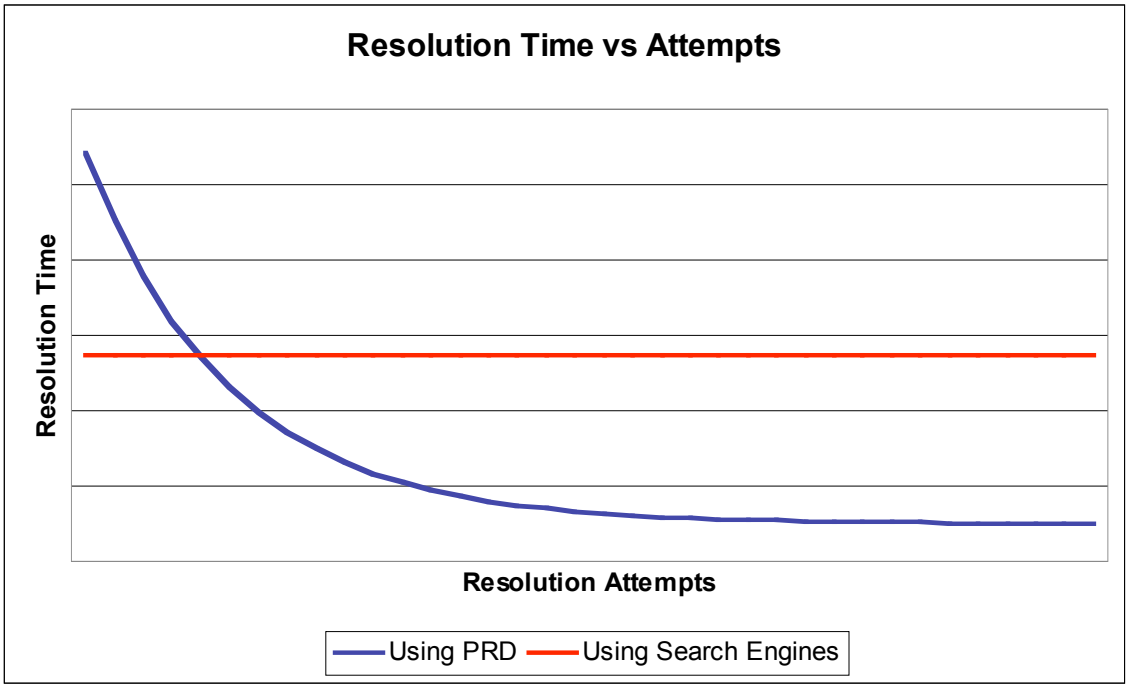

Figure 17. Theoretical representation of time required to resolve problem using PRD and search engines.

It is expected that the time required using PRD will decrease at an exponential rate as the first several attempts will discover the majority of the relevant links. Future attempts serve to refine the ranking of the links included within the PRD information repository.

In the above discussion, it was assumed that each resolution attempt has been performed by a unique user. This is not necessarily the case. A simple example is if a user has in the past solved a problem through PRD and then encountered the error again. Should they require the use of either a search engine or PRD, it can be assumed that they do not remember enough information to solve the problem themselves. They then benefit from the fact that they have previously submitted information into PRD and can retrieve it more readily. In the case of a search engine, perhaps the time required will not be constant as in the unique user case, but there will not be as dramatic of a decrease in time as the user is required to manually parse the returned search results.

The primary benefit of successive attempts in solving a problem with PRD is the decrease of time required to solve a problem; whether a different unique user or the same user is trying to solve it in the future. It is clear that in the long run, PRD should be more efficient that using a search engine. However, the point at which the cross over in efficiency occurs or how much more efficient PRD will be is unclear. This will be discussed in Section 5.3.

5.3 Real-World Validation of PRD

While the previous sections discussed theoretical improvements when comparing PRD with search engines, the true test of the approach is whether PRD is more efficient for solving a wide range of problems for a wide range of users in a real environment. To accomplish this, we would need to run experiments of users solving problems with both the PRD and the traditional approach.

First, it must be noted that a diverse group of users and a diverse group of problems are orthogonal in scope. A diverse group of users will solve the same problem with a wide range of speed, regardless of the approach. Similarly, a diverse group of problems being solved by similar users will exhibit varying resolution times based on the problem at hand. Thus, in order to properly compare apples to apples, both the types of user and types of problems being used in an evaluation must be controlled.

Secondly, although users in a group may be similar, the exact makeup of a user is never exact. A variety of factors will affect the problem resolution time, such as:

- Background
- Experience
- Thought process
- Mood

One way to simulate a controlled test is if the identical person resolved the same problem with both approaches. However, by solving the problem once, a user gains inherent experience about the problem, and thus the initial condition is no longer identical.

While these problems may be solved with further sociological and psychological analysis, it is beyond the scope and resources of a Masters thesis to design and implement such an experimental procedure. Furthermore, without concrete statistics from an experiment such as above, it is impossible to draw conclusions as to the effectiveness of PRD in generic real life scenarios.

6 Conclusion

This thesis has discussed an approach, named *Problem-Resolution Dissemination* (PRD), to reduce the time required to solve a problem. PRD focuses on the class of problems that a developer may encounter, which typically are errors that include a short message describing the problem. PRD increases a user's efficiency in solving these problems by collecting the information previous users have used to solve the problem and introducing display and ranking mechanisms that surface more-relevant information to the current user. The information stored within PRD is in the form of links to web pages.

The novelty in PRD is derived from its use of social relationships and contexts to provide more-relevant results. These aspects of PRD, as described in the previous sections, can be summarized in the following points:

- Browser history is aggregated based on social relationships.
- Browser history is associated with a task context (i.e., a problem).
- User identity is considered a metadata property of a web page.
- Information pertaining to a problem is grouped, ranked and displayed by:
    - task,
    - social relationships.
- Information pertaining to a problem is shared through a pull mechanism.

Additionally, PRD incorporates several features which distinguish it from other approaches in the same problem space. These are summarized in the following points:

- Reduce repetition in searches as well as in the cognitive process of users by sharing information sources for specific problems.
- Logical organization of social relationships in the form of *Team*, *Peer* and *Community* contexts.
- Information collection does not require additional instrumentation and does not contribute any overhead while the user is solving a problem.
- Gracefully degenerates to the traditional approach of using search engines should PRD not contain any information about the searched error.

PRD, along with several other approaches described in Section 2, attempt to assist users in solving problems. Using search engines is the most similar approach to PRD. However, unlike PRD, search engines have not investigated the use of social relationships as a means to provide context to a search, in addition to being a form of metadata to share search experiences with future users.

Expert-Edited Directories (EED) utilize a social aspect in order to enforce a taxonomy on web links. This is similar to PRD in that it attempts to increase the relevancy of links through social means. However, an EED's granularity is at a website level, rather than the web-page granularity utilized by PRD. Secondly, the social structure used by EEDs defines a single two-tiered relationship of editors and users, which does not have the flexibility of PRD's individually user-defined experts and team members.

A Wiki is an approach whereby multiple users can contribute information about a problem resolution. The main disadvantage with Wikis is that information submission is a manual process, which is more time consuming than PRD's automated approach. Software documentation also is fairly limited when compared to PRD as the documentation is only one source of information in the World Wide Web.

A further subset of approaches is through recommendation services. These services can be differentiated between manual and automated means. Asking for help is a manual process which requires additional communication time when compared to the response time of a query to PRD's information repository. Automated recommendation services such as social bookmarking and search-engine recommendations generally do not have the implicit task context provided by PRD or do not collect as much information as PRD. Finally PRD's collection mechanism is similar to those used by quality feedback agents; however, quality feedback agents do not generally attempt to assist users in solving their problems.

A prototype of PRD was built and described in Section 4. A method to validate our approach was discussed in Section 5; however the research required to validate PRD within a real environment is beyond the scope of this thesis. Instead it was evaluated analytically. We argued that in the long run PRD will decrease the time needed for users to resolve a problem when compared with search engines.

We believe that the use of PRD can provide a more efficient means to problem-resolution to users that normally use search engines. For the few initial users, the time required to solve a problem may equal or be slightly greater than using a search engine; however, the future personal and social benefit of collecting and providing their experiences to future users will outweigh those incremental costs.

7 Future Work

The prototype developed for PRD can be improved in several ways. This section covers several of the areas of improvement and outlines what needs to be done.

7.1 Extracting Problem Signatures

In extracting a problem signature from an error message, the current prototype makes an assumption based on the fact that PRD is used within an IBM environment. This assumption will not hold given the different software vendors and configurations in the world and as such the method of extracting a problem signature must be revisited.

The goal of a problem signature is to cluster user input of an error message such that all variations in the inputted error message of a specific error can be grouped. It is an unreasonable assumption to expect users to copy error messages in the same manner. There may be basic variations in the form of white space, to more complex distinctions such as including or not including a stack trace.

Additionally problems may have an identical solution but have different error messages. This may be due to environment differences (such as versions of software) or the cause itself may be inherently different. However, these errors should be clustered together if and only if the solutions are identical. The reasoning is that by grouping the solutions, a larger amount of data will increase the efficacy of the network effect.

One possible solution to address some of the above issues is to extract problem signatures from a standardized log format. Common Base Events (CBE) is a log standard proposed by IBM which may fit this need.

The main challenge in this area is being able to classify and cluster error messages together. Once the accuracy of this is sufficient, the process of creating a problem signature (or more generically, a unique key) is trivial.

7.2 PRD Workflow Improvements

The following sections describe improvements that pertain specifically to the user workflow within PRD.

7.2.1 Intelligent Termination of PRD Processes

In the user workflow of PRD, the user must manually tell the PRD Controller that they have completed the PRD process. This is potentially troublesome as the user may, in their excitement that the problem has been solved, forget to terminate the PRD process. This is unhealthy for our system as the information gained is either: 1) not saved and lost, or 2) contains extraneous information at the end, and 3) interferes with the Last-Page rank modifier.

In order to protect against these possibilities, algorithms can be used to assess the current progress made by the user to solve their problem. While sociological analysis is required

to produce an effective algorithm, the following lists some rough ideas that can be incorporated in future algorithm(s):

- Poll the user to determine if they have finished. The length of time to wait between each poll increases exponentially from an initial, preset timeout.
- Monitor the browser and IDE interactions by the user, and determine algorithmically whether the user is engaged in problem solving or has finished.
- Take periodic snapshots of the pages visited and attempt to programmatically determine when the user has finished based on aggregate statistics of pages (e.g., how long a user viewed a page or how often the user returned to the page).

7.2.2 Pre-population of Error Messages

The workflow of a user can be made more efficient by pre-populating the error message during the PRD start process. This can be done based on an assumption that the user will likely search for the solution of an error message that can be seen within their console or in the last dialog box.

The implementation of this feature seems feasible given that the PRD controller is implemented within the Eclipse environment.

Due to the inability to assess the accuracy of this feature for individual users, it is likely that this pre-population will be a configuration flag defaulted to on.

7.3 Other Ranking Modifiers

The prototype utilizes several of the rank modifiers described in Section 3.5.2. However, to improve accuracy, additional ranking methods can be incorporated. Examples of these are discussed in Section 3.5.2.4.

Future work in this area not only includes the implementation of other ranking modifiers. The weighting of each modifier with respect to each other must also be considered. Experimentation must be performed in order to determine which configuration of weights returns the most-relevant results for the greatest number of searches. The configuration of weights may potentially be different for different industries and environments.

7.4 Miscellaneous Improvements

The following are several miscellaneous improvements that can be made to the PRD prototype.

7.4.1 Firefox and Mozilla 2.0 History Parsing

The prototype was built near the tail end of the Firefox 1.x lifecycle and as such does not support the new history structure used by the Firefox 2.x and Mozilla 2.x releases. As the Mozilla family of browsers is a commonly-used web browser, information collection from these browsers should be supported.

This argument can be extended to other common web browsers such as Opera and Safari.

This future work will be accomplished by implementing an interface with functionality to parse browser history of these browser and store them within PRD's data structures. Additionally, PRD will have to be extended to know of the existence of these browser types.

7.4.2 Relationship Management Through Services

Currently PRD requires a user to manually enter their relationships into the system. This process can be improved, particularly for team situations if the relationship component is integrated with a directory service such as LDAP. By leveraging the information within LDAP, team relationships can be created and maintained through the directory service without requiring input from the user.

It is important to note that a directory service should not be a replacement of the relationship-management system, as a user may create transient relationships for particular projects and/or want particular relationships based on their confidence in certain people.

7.4.3 SSL

Using SSL will increase security and decrease the likelihood of an attack intercepting a user's login information in plain text. However, there will be a need to investigate whether the AJAX services can be used over SSL.

8 Glossary

AJAX: *Asynchronous Javascript and XML.* A mechanism to dynamically update or submit content to a server in a WWW environment without reloading the entire page.

CBE: *Common Base Event*. A standard for logging proposed by IBM.

EED: *Expert-Edited Directories*. A collection of hyperlinks ordered in a hierarchical structure by a group of editors who are experts in their governing area.

HTML: *HyperText Markup Language*. This markup language is the standard for creating web pages on the World Wide Web (WWW).[35]

HTTP: *HyperText Transport Protocol*. The protocol used to transfer information on the World Wide Web (WWW). [36]

IDE: *Integrated Development Environment*. A collection of integrated tools that assist a user in developing software. [37]

LDAP: *Lightweight Directory Access Protocol.* A standard to query directories over TCP/IP.[38]

ODP: *Open Directory Project*. An example of an expert-edited directory located at http://www.dmoz.org/

PHP: *PHP: Hypertext Preprocessor*. This recursive abbreviation names a popular server-side scripting language which can be embedded within HTML.[39]

PRD: *Problem-Resolution Dissemination.* The topic of this paper.

URI: *Uniform Resource Identifier.* A string which can uniquely identify a particular resource on the World Wide Web. [40]

WWW: *World Wide Web*. A network of computers that is navigated using the hypertext protocol (HTTP). [41]

9 References

[1] "How Search Engines Work," August 2006, http://searchenginewatch.com/showPage.html?page=2168031 .

[2] "comScore Media Matrix Search Engine Ratings.," August 2006, http://searchenginewatch.com/showPage.html?page=2156431 .

[3] "Google loses ground in search battle, AOL continues to freefall," August 2006, http://www.theregister.co.uk/2006/08/21/google_comscore_july/ .

[4] "How Many Pages in Google? Take a Guess," August 2006, http://www.nytimes.com/2005/09/27/technology/27search.html?ex=1285473600&en=690f19edefa7572e&ei=5090&partner=rssuserland&emc=rss .

[5] S. Brin, and L. Page "The Anatomy of a Large-Scale Hypertextual Web Search Engine," in

Computer Networks and ISDN Systems, vol. 30, pp. 107-117, 1998.

[6] B. J. Jansen, A. Spink, and T. Saracevic, "Real Life, Real Users, and Real Needs: A Study and Analysis of User Queries on the Web," in *Information Processing & Management*, vol. 2, no. 2, pp. 207-227, 2000.

[7] "Official Google Blog: Now Playing: Movie Trailers," August 2006, http://googleblog.blogspot.com/2006/08/now-playing-movie-trailers.html .

[8] H. Kautz, B. Selman, and M. Shah, "Referral Web: combining social networks and collaborative filtering," in *Communications of the ACM*, vol. 40, no. 3, pp. 63-65, 1997.

[9] "Yahoo Search Builder," August 2006, http://builder.search.yahoo.com/ .

[10] J. Hullen, R. Bergmann, and F. Weberskirch, "WebPlan: Dynamic Planning for Domain Specific Search in the Internet," in *Workshop Planen und Konfigurieren (PuK-99)*. 1999.

[11] "Meta-Search Engines," August 2006, http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/MetaSearch.html .

[12] J. Walkerdine and T. Rodden, "Sharing Searches: Developing Open Support for Collaborative Searching," in *Proceedings of IFIP INTERACT'01: Human-Computer Interaction*, pp. 140-147, 2001.

[13] M. Gnasa, "Sharing knowledge online (abstract only): a dream or reality?," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 602, 2004.

[14] M. Gnasa, S. Alda, N. Gul, and A. Cremers, "Personalized Peer Filtering for a Dynamic Information Push," in *ISMIS, Lecture Notes in Computer Science*, vol. 3488, pp. 650-659, 2005.

[15] "Open Directory Editorial Guidelines," August 2006, http://dmoz.org/guidelines/ .

[16] "Yahoo! Directory," August 2006, http://dir.yahoo.com/ .

[17] "Open Directory Project," August 2006, http://www.dmoz.org/ .

[18] "Submit a Site to Open Directory," August 2006, http://dmoz.org/cgi-bin/add.cgi?where=Computers/Internet/Searching/Directories .

[19] "Wiki: What is Wiki," August 2006, http://wiki.org/wiki.cgi?WhatIsWiki .

[20] "Wikipedia:About – Wikipedia, the free encyclopedia," August 2006, http://en.wikipedia.org/wiki/Wikipedia:About .

[21] "eBay Wiki (Beta)," August 2006, http://www.ebaywiki.com/ .

[22] "MDSN Wiki Home Page," August 2006, http://msdnwiki.microsoft.com/en-us/mtpswiki/default.aspx .

[23] "Experts Exchange, the #1 IT Professional Collaboration Network on the Web," August 2006, http://www.experts-exchange.com/ .

[24] "Del.icio.us," October 2006, http://del.icio.us/ .

[25] D. Millen, J. Feinberg, and B. Kerr, "Social Bookmarking in the Enterprise," in *ACM Queue*, vol. 3, no. 9, pp. 28-35, 2005.

[26] "Pluck Products: Shadows," October 2006, http://www.pluck.com/products/shadows.html .

[27] "Google Personalized Search," October 2006, http://www.google.com/psearch .

[28] "Google: Web Search Help Center – Can other people view my Search History?," October 2006, http://www.google.com/support/bin/answer.py?answer=26659&topic=9005 .

[29] "Crash reporter – Wikipedia, the free encyclopedia," November 2006, http://en.wikipedia.org/wiki/Crash_reporter .

[30] "Quality Feedback Agent", November 2006, http://www.mozilla.org/quality/qfa.html .

[31] D. Hilbert and D. Redmiles, "Separating the Wheat from the Chaff in Internet-mediated User Feedback Expection-Driven Event Monitoring," in *ACM SIGGroup Bulletin*, vol. 20, no. 1, pp. 35-40, 1999.

[32] B. Topol, D. Ogle, D. Pierson, J. Thoensen, J. Sweitzer, M. Chow, M. A. Hoffmann, P. Durham, R. Telford, S. Sheth and T. Studwell, "Automating problem determination: A first step toward self-healing computing systems,". IBM White Paper, 2003.

[33] "Eclipse.org home," September 2006, http://www.eclipse.org/ .

[34] "Mozilla2:Unified Storage – MozillaWiki," September 2006, http://wiki.mozilla.org/Mozilla2:Unified_Storage .

[35] "W3C HTML Home Page," September 2006, http://www.w3.org/MarkUp/ .

[36] "Hypertext Transfer Protocol," August 2006, http://en.wikipedia.org/wiki/Http_protocol .

[37] "Integrated development environment," September 2006, http://en.wikipedia.org/wiki/Integrated_development_environment .

[38] "Lightweight Directory Access Protocol," September 2006, http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol .

[39] "PHP: General Information," September 2006, http://ca3.php.net/manual/en/faq.general.php .

[40] "Web Naming and Addressing Overview (URIs, URLs, …)," September 2006, http://www.w3.org/Addressing/ .

[41] "Desktop Software Support: Netscape Navigator.," August 2006, http://www.hms.harvard.edu/it/swinfo/nav_wwwterms.html#www .