Improving Problem-Oriented Mailing List Archives with MCS

Robert S. Brewer
Collaborative Software Development Laboratory
Department of Information & Computer Sciences
University of Hawaii, Manoa
Honolulu, Hawaii 96822 USA
(808) 956-6920
rbrewer@lava.net

ABSTRACT
Developers often use electronic mailing lists when seeking assistance with a particular software application. The archives of these mailing lists provide a rich repository of problem-solving knowledge. Developers seeking a quick answer to a problem find these archives inconvenient, because they lack efficient searching mechanisms, and retain the structure of the original conversational threads which are rarely relevant to the knowledge seeker.

We present a system called MCS which improves mailing list archives through a process called *condensation*. Condensation involves several tasks: extracting only messages of longer-term relevance, adding metadata to those messages to improve searching, and potentially editing the content of the messages when appropriate to clarify. The condensation process is performed by a human editor (assisted by a tool), rather than by an artificial intelligence (AI) system.

We describe the design and implementation of MCS, and compare it to related systems. We also present our experiences condensing a 1428 message mailing list archive to an archive containing only 177 messages (an 88% reduction). The condensation required only 1.5 minutes of editor effort per message. The condensed archive was adopted by the users of the mailing list.

Keywords
Knowledge condensation, mailing lists, archives, collective memory

1 INTRODUCTION

Modern software development is a complicated task. Over the course of a project a developer may use: a design tool, an editor, a compiler, a debugger, a regression test system, and a packaging tool. The developer may need to use third party libraries or application programming interfaces (APIs) from a variety of sources. On top of all these software development related tools, developers may be responsible for the installation and maintenance of their computing environment: operating system, hardware drivers, word processing, electronic mail, etc. In this kind of complicated technical environment, problems and questions inevitably arise.

Electronic mailing lists have become a common means for users to exchange information and help each other to solve problems. They can be administered by the producer of the product, but they are often run by a user of the product who wants to create a community for the product's users. These mailing lists often become an essential information source for the product, providing up-to-the-minute information and wise advice from experienced users.

As useful as mailing lists are, they have problems that limit their usefulness. A popular mailing list can have tens or hundreds of new messages daily, but keeping up with that level of traffic is prohibitive for most subscribers. While there is a lot of valuable knowledge available, it can be buried among a seemingly endless stream of beginner questions, off-topic discussions, and sometimes unsolicited advertising. The amount of traffic leads many subscribers to delete or file away messages from the list without reading them, simply due to time constraints. When a subscriber has a question, they frequently send it to the list blindly, without knowing whether the answer was just posted recently. This further adds to the information glut.

Luckily for developers, there is another way to find solutions to problems: the mailing list archives. Most mailing lists maintain an archive of all the messages sent to the list, and usually provide some searching capability. With the rise of the browser, most archives are made available via a web page with a search form, such as the Sun archive of the "jserv-interest" list for the discussion of Sun's Java Web Server (http://archives.java.sun.com/archives/jserv-interest.html). These searchable archives provide a way for developers with problems to see if their problem has already been discussed, and possibly even solved.

Unfortunately, mailing list archives are poorly equipped to support problem-solving queries. All the irrelevant information that has been sent to the list is immortalized in the archive, making it difficult to find the useful information. Searchable archives also face the problem that any particular

ICSE 2000, Limerick, Ireland

query may return an enormous number of hits. For example, a developer looking for help on how to redirect a web client to different web page with the Java Web Server might do a search for "redirect" on the jserv-interest mailing list archive. As of this writing, that search returns 175 articles, many of which are irrelevant to the developer's goal. The search hits are displayed in chronological order, and this arbitrary ordering doesn't help the developers find the solutions they are looking for. Another problem with conventional archives is that a particular question may have been asked and answered many times with varying levels of accuracy and clarity. A developer might find a message proposing a solution only to miss the follow up message which explains how that solution is flawed.

We have developed a method for improving the archives of these kinds of problem-oriented mailing lists which we call *condensation*. Condensation involves several tasks: extracting only the messages of longer-term relevance, adding metadata to those messages to improve searching and browsing, and even editing the content of the messages when appropriate to clarify or provide context. The condensation process is performed by a human editor (assisted by a tool), rather than an AI system.

2 CONDENSATION AS A SOLUTION

The goal of condensation is to take the voluminous data stream generated by a mailing list and extract the information which would be useful to future users of the archive. As an analogy, newspapers provide a daily report on current events but are limited by short deadlines, a broad subscriber base, and other considerations. These considerations prevent them from analyzing which events are accurate or relevant over the long term. A story published one day might be amended or retracted the next, depending on how events unfold. However, a book describing world events will tend to have a longer deadline which permits more reflection and analysis: a hoax which might occupy weeks of headlines in a newspaper will probably be little more than a footnote in a book (unless the book is about newspaper hoaxes). The book can also have an index to enable readers to jump directly to the information they are interested in. It is this refinement of information that we refer to as condensation.

There are a variety of ways that the information could be condensed, depending on the intended use of the archive. Our goal is to provide a searchable archive of information that allows developers to quickly find solutions to specific problems. Since the goal is to help developers find solutions as efficiently as possible, there is little point in preserving the conversational nature of the mailing list data stream. Users with problems are looking for solutions, not conversation. Condensation requires omitting unimportant, contradictory, or inaccurate messages, removing unimportant portions of messages, inserting new text into messages when required for clarification, and adding new messages to the database from scratch when that is the best way to explain something.

For this narrow focus on problem solving, we can categorize each message as either a problem or a solution. Each message is annotated with keywords which are actually relevant to the subject of the message. The result of this process is a condensed archive, an archive which does not suffer from the problems of an unabridged searchable one. Since only truly useful information will be put into the archive, the amount of data to be searched is smaller which improves the odds of a search being accurate. Developers also benefit by having a set of standardized keywords which insure that messages using different terms but discussing the same topic will be retrieved by a single search.

3 MCS: A SYSTEM FOR CONDENSATION

To demonstrate the improvements possible through condensation, we have constructed a new software system for condensing mailing list archives. We have named this system (for lack of imagination) the Mailinglist Condensation System or MCS (http://csdl.ics.hawaii.edu/Research/MCS/MCS.html). MCS has two main parts: one which is dedicated to taking the raw material from the mailing list and condensing it, and another which stores the condensed messages and allows developers to access them.

One way to perform the condensation would be to implement an AI system that reads the messages and then decides what information to keep, what to throw away, and what keywords to assign to each. In order to perform this task adequately, the system would need superb natural language processing capabilities and an in-depth knowledge of the mailing list domain. Such a system is currently at or beyond the state of the art, and would at any rate require a substantial investment of resources to complete.

A practical alternative to an AI system is the employment of human editors for condensation, along with extensive tool support to lower editing overhead to an acceptable level. Humans are quite good at examining textual information and determining what is useful and what is not, while computers are good at queries across structured data [3]. This alternative also exploits the presence of mailing list gurus: subscribers who read all messages sent to the list and who are domain experts. Therefore, in MCS, humans do the editing using the MCS editing program which makes the process as efficient as possible. Only the editors need to use the editing portion; the interface of the end-user portion is simpler and geared towards ease of use. If AI systems become available which can replace some or all of the manual effort required by the editor, they can be added to MCS.

Requirements

MCS was designed to help users of problem-solving mailing lists by improving the usability of the list archives. Making archives more useful not only helps the archive users, it also helps to improve the quality of the mailing list itself, because people are less likely to re-request information which is easily available via the archive. To achieve this goal, the

user community must adopt MCS in preference to the many existing systems for generating and maintaining searchable mailing list archives. To encourage users to adopt the system, the design of MCS takes into account two issues: an explicit domain focus, and the existing list community.

Most mailing list archive search engines are designed to work with any mailing list. Because they must work with any mailing list, conventional search engines are limited to keyword searches and simple search results presentation. The idea of MCS is the exact opposite: mailing list archives can be enhanced by tailoring the search engine to a particular mailing list domain. By embracing the details of a particular kind of mailing list MCS provides greater utility and efficiency for archive users.

Because MCS receives its input from a mailing list, it is crucial that MCS be designed with the social structure of a user-supported mailing list in mind. Specifically, the mailing list and its community should not be adversely affected by MCS. Any attempt to impose restrictions on how people read or participate in the list (like requiring users to use special software or compose messages in a certain format) would be met with blistering criticism. MCS must stand apart from the mailing list itself, limited to using messages from the list on an as-is basis. MCS also takes into account the needs of the user community by having very low requirements accessing the archive. The archive is accessed using a web browser which is presumably standard equipment for most mailing list participants. Furthermore, the web pages themselves are simple; they contain no images, no Java applets, and no JavaScript to ensure that users can use older browsers to access the archive. The omission and editing of messages is central to MCS, but those actions can reasonably arouse suspicion among list members as to the fairness of the editing. To assuage these fears and to assure context, MCS provides a link from each edited message to the original message maintained in a separate unabridged archive.

Functionality

In addition to condensation MCS provides several novel features which facilitate user searching. All examples and screenshots in this section refer to a condensed archive of the "jcvs" mailing list (http://www.gjt.org/servlets/MailingLists/ListInfo.html/jcvs). The jcvs mailing list exists for the discussion of the jCVS system (http://www.trustice.com/java/jcvs/index.shtml), which is a Java client for the Concurrent Versions System (CVS) (http://www.sourcegear.com/CVS). See Section 6 for more information about the archive.

Keywords

Messages in MCS are assigned keywords by the editor. The keywords are chosen sparingly such that there are only a few for each message, instead of indexing all the words in each message. These keywords are organized into a hierarchy of categories by the editor. Each keyword and category can be annotated by a description and an URL (Uniform Resource Locator) when appropriate. The maintenance of the keyword hierarchy is a major part of the editor's task. While maintaining keywords is time consuming, having the keywords organized in this way provides advantages to the archive user. A frequent problem when using conventional archives is figuring out what keyword has been used for a particular concept. For example, "freeze", "hang", and "lock-up" are all words which describe the same concept, but a user might have to try searching for all three in order to retrieve all the problems related to that concept. With the keyword hierarchy, the synonym problem is all but eliminated. Having a relatively small number of keywords arranged in a tree also allows users to browse through the keywords and learn what kinds of topics are contained in the archive. Keywords can be browsed using a system similar to the one used at the Yahoo web portal (http://www.yahoo.com/).

The grouping of keywords into categories enables another useful option for users. MCS allows users to perform a *2D search* by performing simultaneous searches for pairs of keywords. The user selects two categories which contain keywords, and then initiates the 2D search. MCS performs the cross-product of the two categories, and for each tuple of keywords it performs an AND search of the database. The result is a table which shows the co-incidence of the keywords in the two categories. Figure 1 shows the results of a 2D search with the categories of "Java Concepts" versus "jCVS Versions". The archive this search was performed on has a limited amount of data, but the results provide some insight as to which concepts have proven problematic with which software versions. Note that just because the JavaHelp row has no matches doesn't mean that no messages related to JavaHelp are in the database. It just means that no JavaHelp-related messages refer to a particular version of jCVS, probably because the version of jCVS was not relevant to the problem or solution.

Message Types

MCS has a very simple schema for the messages it stores. Each message has a type which currently can is either "problem" or "solution". This typing has a profound affect on MCS. When a developer has a problem, they can search the archive to see if they can find a message describing a similar problem. Once they find a relevant problem in the archive, they can immediately see what solutions have been proposed for that problem. Figure 2 shows the results of a search for the keyword "Swing". Since MCS understands that problems and solutions are associated with one another, it can group together the search results so developers can see related problems and solutions at a glance. The typing of messages also makes the editor's job easier because they only have to worry about messages which are problems or solutions, all other kinds of messages can be discarded.

Searching by Symptom

MCS also provides symptom-based searching. It is common for a developer to be aware of the symptoms of their prob-

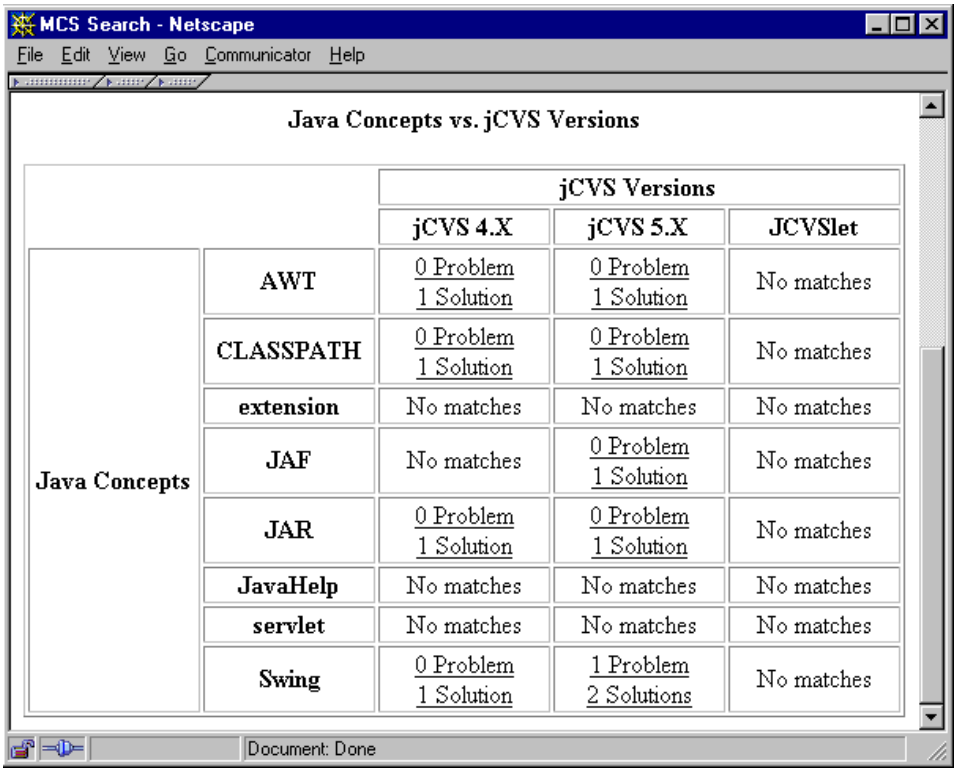

Figure 1: A 2D search of Java Concepts vs. jCVS Versions

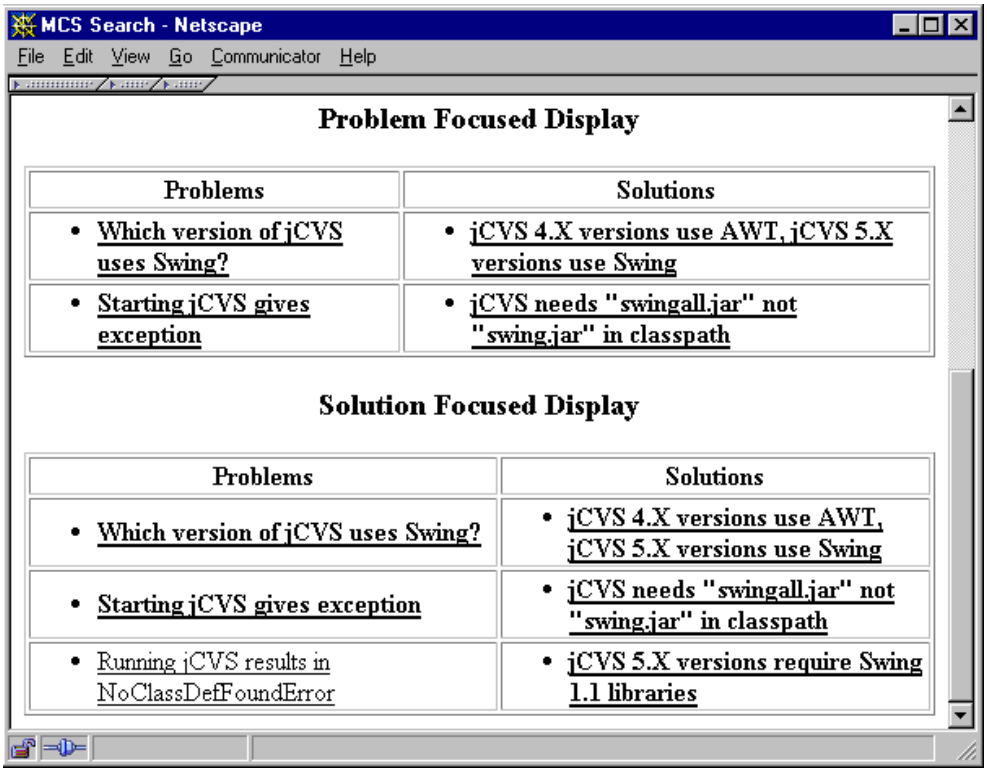

Figure 2: Results from a search for keyword “Swing”

lem, but unaware as to what the cause might be. While editing messages, the editor may notice that a problem message contains within it a textual pattern which is symptomatic of the problem. The symptom is usually an error message of some sort. The editor can convert the symptom into a regular expression, thereby stripping out all the irrelevant parts of the symptom. This regular expression symptom is stored with the message as metadata. Later, when a developer encounters a similar problem, they can copy and paste the error message directly into a text field the symptom search mode on an MCS web page and initiate a search. MCS will then attempt to match the given text against all the symptom expressions in the archive, displaying the results to the user.

For example, suppose the editor noticed this error text in a message being edited:

```
java.lang.NoClassDefFoundError:
 javax/swing/DefaultBoundedRangeModel
   at com.ice.jcvsii.JCVS.instanceMain(Compiled Code)
   at com.ice.jcvsii.JCVS.main(Compiled Code)
```

From his or her knowledge of the domain, the editor knows that this is symptomatic of using the wrong version of the Java Swing class library. So in this case, the relevant portion of this error in regular expression form would be:

```
java\.lang\.NoClassDefFoundError: javax/swing/.*
```

This symptom gets at the core of the error, because the only two important parts are the type of the error and initial prefix of the mismatched package. If a developer later pasted in the following different error message, MCS can still match it to the correct problem:

```
java.lang.NoClassDefFoundError:
 javax/swing/text/JTextArea
   at com.ice.jcvsii.JCVS.instanceMain(JCVS.java:81)
   at com.ice.jcvsii.JCVS.main(JCVS.java:63)
```

4 IMPLEMENTATION

MCS consists of two subsystems: the server side that stores the archive and provides a World-Wide Web interface to archive users, and the editing tool that allows editors to submit edited messages to the archive. MCS has been implemented entirely in Java which means both the server and editor can be run on a wide variety of platforms. The implementation of MCS consists of 7387 non-comment lines of Java code, 343 methods, and 55 classes.

The server side consists of several Java servlets and support classes. Servlets are a way to extend the functionality of server, particularly web servers. The MCS servlets conform to version 2.0 of the Sun Servlet API, which means that they can be used with any of the many web servers which support servlets. The web server we used was Sun's Java Web Server. The servlets are responsible for: storing the condensed messages in a simple flat file database, accepting user queries, presenting search results and messages to users. There are also servlets which interact with the editing tool to allow updates to the database.

The editor side makes use of the fact that the data source for condensation is email. The messages to be condensed are stored in normal mailbox folders on an email server. The Java email client ICEMail (http://www.trustice.com/java/icemail/) has been extended for use as the MCS editing tool. Editors use ICEMail to contact the email server using IMAP (Internet Message Access Protocol), and use the standard ICEMail interface to read and delete messages from the list. When the editor encounters a message which needs to be condensed for inclusion in the archive, he or she accesses the MCS extensions to ICEMail which allow the editing and annotation to take place in a separate window. When the condensation of the message is complete, the editor can upload it to the MCS database with the press of a button.

Users of the MCS-condensed archive access it using their web browser. They go to a particular URL, and the MCS servlets dynamically produce the HTML which is rendered by the user's browser. The servlets produce only standard HTML to enable almost any browser to use the archive.

5 RELATED WORK

There are a variety of systems and research related to maintaining and searching collective memory. Here, we examine several such systems and compare them to MCS. Some of these systems are somewhat informal (like moderated mailing lists and FAQ files), and some are formal research projects. The informal systems are based on the author's knowledge of those systems and generally do not have references because they evolved from common Internet practices.

Moderated Mailing Lists

Some mailing lists address the signal to noise problem by having a moderator or a group of moderators. All submissions to the list are forwarded to the moderator(s) who read the messages and decide whether or not to distribute them to the list. On most lists, the moderator(s) do not edit the messages submitted. They just choose whether or not to distribute the message. Also, to allay fears of censorship on the part of the subscribers, usually the criteria used to decide whether to distribute a message are rather liberal, e.g., the message is related to the topic of the mailing list and not an advertisement [10].

While moderation can be useful for maintaining a high signal to noise ratio, it suffers from several problems addressed in the design of MCS. Moderation requires a substantial commitment on the part of the moderator(s) to review submissions in a timely manner. Failure to do so halts all traffic on the mailing list and annoys subscribers who have come to expect the short turnaround time that digital media can provide. Moderators also tend to face continual concerns from subscribers as to whether they are moderating in a fair and consistent manner. Since MCS does not affect the list distri-

bution itself at all, most concerns about censorship should be eliminated. MCS provides a link from each edited message to the original unabridged message so users can easily see what was edited out or changed in any particular message.

Frequently-Asked Question Files

Most frequently-asked question (FAQ) documents attempt to provide a similar service to MCS: a condensed version of important and useful information that came from a mailing list or newsgroup. There are several important differences between the two systems. FAQ files are usually maintained without specific tool support so they require extensive effort on the part of the maintainer to create and update. FAQ files are generally created with the intention of easy distribution either as plain text or HTML. Because of this requirement, FAQ files are mostly limited in size to a few hundred kilobytes and they are laid-out to be easy for humans to read. Since FAQs cannot be of arbitrary size and complexity, they must omit useful information.

MCS does not have these limitations. Since the system is not intended to be distributed by FTP or by posting to a mailing list or newsgroup, it can be as large as necessary. A sophisticated query system is an integral part of MCS, so it is not necessary that the underlying data be structured in an easily understandable human format. Because MCS lacks these two restrictions, it need not limit the archives it creates to merely frequently-asked topics, it can contain any information that would be useful regardless of how broad its appeal.

FAQ FINDER

FAQ FINDER allows users to quickly find answers to questions by searching a database made up of FAQ documents posted to Usenet [4]. The user enters his or her question into the system in natural language. First the system uses standard information retrieval techniques to determine which FAQs in the database are most likely to contain the answer to the question. It presents the top five FAQs to the user, who can select the most likely candidate. Then the system uses a combination of lexical and semantic similarity checks between the asked question and the question-answer pairs in the FAQ file. It then presents the five most likely pairs for user consideration. A live version of the system can be found on the web (http://faqfinder.ics.uci.edu:8001/).

While FAQ FINDER is an interesting system, it is attempting to solve a different problem than MCS. FAQ FINDER assumes that there exists a large number of FAQ files which are already organized in question-answer format, and from those files it attempts to help users find the answer to their questions. The designers of FAQ FINDER explicitly chose not to implement any domain-specific knowledge into their system because their intended dataset is a large number of unrelated FAQ files. MCS attempts to create a FAQ-like body of knowledge from a mailing list, and then present the condensed information in useful, possibly domain-specific ways. In this way MCS attempts to solve the problem of getting the information into an FAQ-like state, which is already presupposed in FAQ FINDER. It might be possible to create a "stub" FAQ which FAQ FINDER could index, and if the user's question is a good match, FAQ FINDER would just send the user to the MCS-created archive.

Answer Garden and Answer Garden 2

The Answer Garden system is designed to provide an organically growing database of answers to questions by end-users [1]. Users interact with the system by answering a series of diagnostic multiple-choice questions which lead them through the tree of answers already in the system. If users find that their questions are not answered in the database, they can enter their questions into the system and it will be forwarded to an appropriate expert via email. When the expert answers, the result is sent back to the original question-poser and also inserted into the tree for future retrieval.

Answer Garden's goal in life is to answer questions. Like MCS, it uses human input to decide what questions and answers should be in the database. However, Answer Garden is really only suited to the task of answering questions. A user who just wants to browse information either has to answer the diagnostic questions or guess where on the tree the information might be located. It also requires a group of experts to be responsible for answering the questions posed by users. MCS does not require users to use any special software to continue participation in the mailing list, while Answer Garden assumes that all users will use the Answer Garden tool when they have a question. In addition, MCS provides the symptom search method which allows a user to use an error message to find the solution to a problem immediately. Answer Garden requires users to answer a series of diagnostic questions, with no way to short circuit the process.

Answer Garden 2 is a refinement of the Answer Garden system. It improves on Answer Garden by adding a system of gradual escalation for questions input into the system (thereby providing more context to the person answering the question), and a subsystem for collaboratively "refining" the information in the database [2]. All of this is built on a set of versatile and configurable components which allow the system to be tuned for a particular environment.

Although Answer Garden 2 appears much closer to MCS in its goals, the two systems appear to differ in implementation and user interaction. Answer Garden 2 does not implement features such as 2D or symptom search. In addition, MCS has been field tested as discussed in Section 6.

Faq-O-Matic

Faq-O-Matic was created to solve some of the same problems MCS addresses: the difficulty in finding answers in mailing list archives, and the substantial effort required to maintain an FAQ (http://www.dartmouth.edu/~jonh/ff-serve/cache/1.html). Faq-O-Matic addresses these issues by creating a dynamic WWW-based FAQ which and member of a user community can contribute to. Any user can browse

through the web pages and make additions as necessary. This provides an easy way to maintain an FAQ since any member of the community can volunteer to help. However, there is no centralized authority in charge of the FAQ, so pieces of potentially incorrect or mutually conflicting information can be posted. Furthermore, new additions have to be written from scratch by contributors, unlike MCS.

Vector-Based Text Searching and Summarization
The standard technique for search and retrieval from large text databases is the vector method. Each document or segment is decomposed into a vector of terms which is assigned a weight which is proportional to the frequency of the term on the document, but inversely proportional to the frequency of the term in the whole collection of documents. Using this technique, searches can be conducted by comparing the vector of the query, to the vector of each document in the collection, and a summary of a document can be generated by selecting segments of the document which are computed to be most relevant [13]. However, this work is not directly applicable to MCS. The domain of MCS is relatively short email message, unlike the domain of automatic text summarization which typically deals with larger documents like news stories or encyclopedia entries. The vector-based approach is problematic for MCS, because the descriptions of problems and solutions are so small that there is insufficient data for such statistical methods. The summarization techniques are not relevant since they work on the paragraph level. Summarizing an encyclopedia entry by selecting paragraphs makes sense, but for a 200 word message it makes little sense.

Other Work
W. B. Frakes and B. A. Nejmeh developed a system for software reuse based on searchable archives of annotated source code [8]. This research is similar to MCS in the use of human annotation of the documents in the database. However, it appears that the metadata in the CATALOG system was created by the author of the source code module, while in MCS the metadata is added by the editor after the fact. Rubén Prieto-Díaz has also done work on software reuse search systems using a system called *faceted classification* [11]. Faceted classification involves creating a set of categories and a controlled vocabulary of terms for each category. Each document is then assigned one term from each category. This type of classification scheme could be used in MCS to ease the burden of maintaining the keyword hierarchy, but it would make it difficult to perform meaningful 2D searches since each category would contain many terms.

The LaSSIE system attacks the problem of disseminating architectural knowledge about a large software system to all developers working on the system [5]. In LaSSIE, a "reverse knowledge engineer" manually creates entries in a knowledge base which permits natural language queries and semantic retrieval. MCS is like LaSSIE in that it requires a manual process to generate the metadata required for retrieval. While LaSSIE has the semantic retrieval mechanism, it is unclear whether that system could be applied to the far less structured domain of mailing list content.

The MediaDoc system also addresses the problem of explaining software systems to users [6]. MediaDoc employs a complicated model of the user, including the user's goals and plans. While this approach may be able to provide better responses to user queries, it will incur even more editor overhead than already present in MCS.

The Knowledge Depot system provides a group memory repository, and a project awareness system to a workgroup using standard email distribution lists [9]. Messages are categorized in the repository by the workgroup members using keywords in the subject lines of the messages. The project awareness subsystem allows users to be sent periodic summaries of recent activity in categories they are interested in. While having users perform the categorization may work in a small workgroup, it would not be feasible in MCS, because it requires additional effort on the part of all users and assumes that all users are equally qualified to perform the categorization.

6 EVALUATION

To evaluate the research hypothesis that MCS is an improvement over existing archives, we designed a case study of MCS. The case study involved creating a condensed archive of a mailing list, releasing the archive to the list subscribers, and collecting qualitative and quantitative data on the users of the archive. The goal of the case study was to answer three research questions: can messages be condensed in a reasonable amount of time, will subscribers of the mailing list adopt the condensed archive, and will the archive users prefer the condensed archive to an existing archive?

Target Mailing List
As mentioned in Section 3, we selected the jcvs mailing list as the target for the case study. As a trial run, we also condensed the "icemail" mailing list which exists for the discussion of the ICEMail program mentioned in Section 4. This mailing list had many fewer subscribers, and while the condensed archive was announced to the subscribers, no data was collected on their usage of the archive. Both the jcvs condensed archive (http://csdl.ics.hawaii.edu:8100/) and the icemail condensed archive (http://csdl.ics.hawaii.edu:8090/) are online and publicly accessible.

Study Implementation
The case study was implemented in several stages:

1. The trial run condensation of the icemail list was performed.
2. The MCS software was revised from the lessons learned in the trial run.
3. The jcvs archive was condensed over several weeks.
4. The jcvs archive was announced to the mailing list on

January 24, 2000.

5. Users were able to use the condensed archive at their leisure.

6. On February 10, 2000, an online questionnaire was made available via the top page of the MCS archive. An announcement was made to the mailing list informing users of the questionnaire's existence and encouraging them to fill it out.

7. On February 23, 2000, I ceased collecting data from the questionnaire, thus ending the case study.

Results

Editor Overhead

Even if condensed archives are deemed by users to be more useful than conventional archives, the system will not be adopted if the condensation process requires too much effort by the editor. The editing task must be feasible to perform, and it must not require too much time spent per message. We assessed this measure by collecting time data from the editing tool as the archive was condensed.

As we condensed the two archives, we recorded how much time was spent editing. We recorded time spent reading messages, condensing them, and any external reading required to condense the messages. We did not record time spent fixing any critical defects in the MCS software, as they were discovered as that time is not relevant to determining the expected time required to condense future archives. Table 1 shows the time data for both archives.

As you can see, it took substantially less time per message examined when condensing the jcvs archive compared to the icemail archive. We attribute this to two factors: tool improvement, and editor improvement. The editing tool had a variety of quirks and defects when the first archive was condensed, so the condensation required substantial manual effort. After we condensed the first archive, we made many improvements to the editing tool which increased the speed with which messages could be condensed. In addition, we learned more about how to condense from the experience of condensing the first archive. The increased knowledge allowed us to spend less time thinking about those issues when condensing the second archive. With more practice and enhancements to the editing tool, it should be entirely possible to bring the amount of time spent per message to one minute or lower. For a medium to low traffic mailing lists, this seems like an entirely acceptable amount of time to spend editing, particularly since this includes the time required to read the email for the first time, which an editor would presumably be doing anyway.

Table 2 shows a summary of the contents of the archives. As you can see, the archives contain only a fraction of the messages examined (23% for icemail, 12% for jcvs). This is to be expected because one of the goals of MCS is removing unnecessary messages. The jcvs list had a smaller percentage retained than the icemail list, presumably due to the heavier traffic of the jcvs list. For both lists, the number of keywords is fairly close to the number of archived messages. Because most messages contain multiple keywords, this indicates that many messages used the same keywords, otherwise the number of keywords would be larger than the number of archived messages. Symptoms were also fairly common on both lists: 21% of icemail problems had symptoms, 30% of jcvs problems had symptoms. The relatively high incidence of problems with symptoms indicates that the symptom search can be a useful search technique.

Adoption

We define adoption as a significant fraction of the subscribers of the targeted mailing list using the condensed archive either in addition to or instead of the traditional archives. We have used the number of list subscribers as an estimate of the number of potential condensed archive users. The adoption percentage is then the number of condensed archive users divided by the number of list subscribers, expressed as a percentage. To decide what adoption percentage would be indicative of success, we consulted Everett's work on the the diffusion of innovations [12]. He divides adopters into five categories based on the rate at which they adopt innovations. The two categories containing the most rapid adopters are the *Innovators* (consisting of 2.5% of the population), and *Early Adopters* (consisting of 13.5% of the population). We decided to target both these categories, so our target adoption percentage is the sum of the category sizes: 16%.

To measure the adoption percentage, we needed two pieces of information: the number of list subscribers and the number of users of the condensed archive. The list maintainer provided the number of list subscribers from the subscription list. An estimate for the number of users of the condensed archive was obtained by analysis of web server log data from the condensed archive. Note that the adoption percentage, as we have defined it, is an imperfect measure since we cannot positively determine whether the users of the condensed jcvs archive are actually subscribers of the mailing list.

The Java Web Server, like most web servers, records a log of all HTTP [7] requests made to it. Each log entry contains the request made, the IP address of the requester, and a timestamp. At this level, the data provides mere hit count information which is a poor indicator of the number of actual users of the system. There are a variety of ways to track users more closely, but they generally require the user to either register and log on or accept *cookies*, which many people consider intrusive. Since the major goal for MCS is adoption, annoying users is to be avoided at all costs.

Using only the raw request data, there are two ways to estimate the number of users of the archive: unique IP address

Mailing List	Messages Examined	Condensing Time	Average Time Per Message
icemail	166	481	2.90
jcvs	1428	2165	1.52

Table 1: Editing time results for two condensed archives (all times in minutes)

Mailing List	Messages Examined	Messages Archived	Problems	Keywords	Symptoms
icemail	166	39	19	40	4
jcvs	1428	177	82	120	26

Table 2: Statistics on the composition of two condensed archives

counting, and organizational analysis. The first method involves simply counting the number of unique IP addresses which have made requests. This technique, however, has problems because of dynamic IP addressing and the use of public access computers (such as in a University lab). In the case of dynamic IP addressing, a user may access the archive from the same computer but over the course of a day that computer's IP address might change which would cause this user to be counted more than once. In the case of a public access computer, multiple people may use the same computer to access the archive. Since the computer only has one IP address, the multiple users will only be counted once. Dynamic IP addressing is expected to be more prevalent than shared computers among jcvs subscribers, so we expect the unique IP address count to be an overestimate of the adoption rate.

The other method for estimating the number of users of the archive is organizational analysis. Organizational analysis attempts to collate the number of distinct organizations that issued requests to the web server. While requests are recorded in the log by IP address, the Domain Name System (DNS) can be used to map the IP address into a domain name. Domain names can be more useful than raw IP addresses as they indicate what organization an IP address belongs to. Of course this method has its own problems: multiple users at the same organization are only counted once, and some IP addresses cannot be resolved to a domain name. However these problems should make the size organization list an underestimate of the number of users of the archive.

Using a program called *Analog* (http://www.analog.cx/), we analyzed the log file. Since the web logs were used to assess adoption of the MCS archive, we removed any requests which originated from any of the researchers' workstations. According to the analysis by Analog, the web server received requests from 99 distinct IP addresses during the case study. As mentioned earlier, this value is almost certainly an overestimate of the actual users since some users probably accessed the archive from different computers. However, this value is probably an upper bound on the number of users of the archive.

Analog also generated what it calls an organization report which uses the organizational analysis technique. There were 70 entries in the organizational report. Some of the entries are not actual users such as the googlebot.com entry which is presumably a spider which collects data for the Google search engine (http://www.google.com/). The last nine entries have only one request which indicates that they didn't really do anything meaningful with the archive. On the other hand, however, there were 176 requests from IP addresses which could not be mapped into domain names which would presumably raise the organization count if they could have been resolved. The organization list also counts multiple users coming from the same organization as one, which could cause an underestimation of the number of users. On balance, the value of 70 is a better and more conservative estimate of the number of actual users than the 99 distinct IP addresses value.

Given this estimate of the number of users of the archive, we can now estimate the percentage of list subscribers that used the archive. The list had approximately 401 subscribers at the start of the case study. Using the figure 70 as the estimate of the number of archive users, we find that this accounts for roughly 17% of the list membership. This exceeds the 16% goal which we set as the measure of whether or not the list subscribers had adopted the condensed archive.

Preference

Assessment of the users' preference of the condensed archive over traditional archives was determined in a qualitative way through a user survey administered using a web form on the archive site. The survey included the question: "Since the new problem-solving archive has been available, do you find yourself using it instead of the old archive?". All the questions were multiple choice except for two open-ended feedback questions.

A total of six questionnaires were filled out. We classified the

six questionnaires returned into three different groups: those who had used neither the old archive nor MCS, those who had only used MCS, and those who had used both the old archive and the MCS archive. Each group had two questionnaire results which fit the characteristics. Due to the small number of questionnaires returned, we limit our analysis to qualitative trends that we noticed in the data.

The two users who did use both archives reported that: they always found what they were looking for in the MCS archive, they were completely satisfied with the MCS archive, and that they were willing to volunteer as editors. This makes some sense: in order to fully appreciate MCS you need to have used traditional mailing list archives. The willingness of respondents to consider volunteering to be editors is encouraging, and provides some hope that the burden of editing could be spread out among multiple editors. All four users that had used the MCS archives rated the archive as satisfactory or completely satisfactory.

7 FUTURE WORK

Scalability Improvements

MCS was designed as a research system, and as such, many decisions were made in favor of speed and ease of implementation. However, if MCS grows to serve large archives, work will be required on its scalability. MCS uses a flat-file storage structured rather than a backend database, so some slow down will occur when there are many users and many condensed messages. MCS also assumes that there is only a single editor for an archive, which obviously does not scale well to high-traffic lists where editors will want to share the daily workload.

Adoption by Other Mailing Lists

Convincing other mailing lists to use the software for their archives would be the final stage in moving the software out into general use. This adoption process may be more difficult because it requires the mailing list's community to embrace the system and it also requires recruitment of one or more editors from the mailing list.

Expansion into Technical Support Market

Mailing lists are used extensively both internally and externally by those who provide technical support. In this kind of environment, MCS could be used to do a sort of "data mining" on old email archives, turning them into valuable knowledge bases which can in turn reduce support costs. With the potential of lowered costs, it would make sense for corporations to support editors either within their company or even external editors.

ACKNOWLEDGMENTS

This research would not have been possible without the help of Philip Johnson, and all the members of the CSDL research group. Your encouragement is very much appreciated. Thanks also to Yuka Nagashima for her steadfast support throughout. This research is supported by the National Science Foundation under Contract Number CCR-98-04010.

REFERENCES

[1] M. S. Ackerman and T. W. Malone. Answer garden: A tool for growing organizational memory. In *OIS90*, Filtering, Querying, and Navigating, pages 31–39. ACM Press, 1990.

[2] M. S. Ackerman and D. W. McDonald. Answer garden 2: Merging organizational memory with collaborative help. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 97–105, New York, Nov.16–20 1996. ACM Press.

[3] F. P. Brooks, Jr. The computer scientist as toolsmith II. *Communications of the ACM*, 39(3):61–68, Mar. 1996.

[4] R. D. Burke, K. J. Hammond, V. A. Kulyukin, S. L. Lytinen, N. Tomuro, and S. Schoenberg. Question answering from frequently asked question files: Experiences with the FAQ finder system. Technical Report TR-97-05, Department of Computer Science, University of Chicago, June 20 1997.

[5] P. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard. LaSSIE: A knowledge-based software information system. *Communications of the ACM*, 34(5):35–49, May 1991.

[6] A. Erdem, W. Johnson, and S. Marsella. Task oriented software understanding. In *Thirteenth International Conference on Automated Software Engineering*, pages 230–239. IEEE Computer Society Press, 1998.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2068, Internet Engineering Task Force, Jan. 1997.

[8] W. Frakes and B. Nejmeh. Software reuse through information retrieval. In *Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences*, pages 530–535, Jan. 1987.

[9] M. Kantor, B. Zimmermann, and D. Redmiles. From group memory to project awareness through use of the knowledge depot. In *California Software Symposium*, 1997.

[10] R. C. Pedersen. Reviewing Internet mailing lists. *The Serials Librarian*, 30(2):27–33, 1996.

[11] R. Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.

[12] E. M. Rogers. *Diffusion of Innovations*, chapter 7. The Free Press, fourth edition, 1995.

[13] G. Salton, A. Singhal, C. Buckley, and M. Mitra. Automatic text decomposition using text segments and text themes. In *Hypertext '96*, pages 53–65, 1996.