A Comparison of Questionnaire-Based and GUI-Based Requirements Gathering

J. Michael Moore, Frank M. Shipman III
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112 USA
1 409 845 5534
Email: {jmichael, shipman}@cs.tamu.edu

Abstract:

Software development includes gathering information about tasks, work practices and design options from users. Traditionally requirements gathering takes two forms. Interviews and Participatory Design (PD) practices gather rich information about the task and the domain but require face-to-face communication between the software engineers and the users. When such communication is not possible traditional software engineering frequently relies on questionnaires and other paper-based methods. Unfortunately, questionnaires often fail to capture implicit aspects of user tasks that may be identified through one-on-one interactions. This project investigates a method of gathering requirements whereby users, working independent of software engineers, construct rough interfaces augmented with textual argumentation. Our initial study has compared the use of GRC with questionnaire-based requirements gathering.

Requirements Gathering from End Users

Requirements gathering is a prerequisite for any software development effort. It is also viewed as the most critical step [Dardenne, et al. 1991]. The quality of a system depends on how well the final system meets the requirements [Finkelstein 1994]. Consequently, the requirements become the template that the final system will be compared with to determine success or failure. Stakeholders, those who often have a vested interest in a project, are the information source for requirements. It becomes vital that the requirements be specified and analyzed in enough detail so that the final system fulfills the expectations of all stakeholders. However, more often than not the requirements do not adequately state stakeholder expectations.

Unsuccessful communication is often at the root of inadequate requirements specification [Potts, Takahashi, Anton 1994]. This can lead to requirements that do not capture the complete expectations of stakeholders. Stakeholders can include managers, software engineers, end users, clients, etc. End users provide a rich source of information about a system since they will directly interact with the system. They also tend to have a solid knowledge of the domain and tasks being automated. Thus, a major goal early in the software engineering process becomes gathering meaningful requirements from end users.

Potts, et al. [1994], have proposed inquiry-based requirements analysis. They break the process into three major phases: Requirements Documentation, Requirements Discussion, and Requirements Evaluation. Finkelstein [1994] also talks about seven difficult areas in requirements engineering. One of those areas is requirements acquisition. Requirements documentation or requirements acquisition is the process of getting an initial set of requirements from stakeholders. Getting this starting base of requirements offers a unique challenge.

Types of Requirements

Requirements state expectations at two levels. The first are high level functional requirements that state goals the system will achieve. The second are more fine-grained procedural expectations that describe how the system will behave. As might be expected, it is possible to fulfill a set of functional requirements in multiple ways using different instantiations of procedural requirements.

The initial requirements are often a set of broad ranging goals stated at conception. Frequently, these conceptual requirements motivate the development of a given software package. Clients generally know the basic tasks that need to be performed as well as their ideas about how the software should “feel” when used. The requirements, specified at this level, are functional requirements that give little indication regarding the procedural requirements that are entailed. The procedural requirements comprise the set of processes that need to be automated and integrated with the practices of end users. Getting input from end users about procedural requirements can be just as important as the functional requirements.

Methods for gaining the initial set of requirements can include questionnaires, interviews, task analyses, and goal based methods that employs a rigid rules based approach [Dardenne, et al. 1991]. Each method results in a set of requirements that capture functional and procedural information to varying degrees.

When answering a questionnaire, individuals feel they are communicating with another person and this communication assumes a shared background knowledge that supports the exchange of information. However, when attempting to elicit detailed procedural information, these assumptions can inhibit the successful exchange of ideas. When attempting to elicit information from a client using existing requirements specification processes, the client may not verbalize pertinent information with the assumption that the person reading the questionnaire already knows these things implicitly.

The “feel” for users is often difficult to quantify since they address user attitudes and emotions that vary among individuals. While users know the functions that the software should entail, they do not explicate the fine-grained procedural behavior that they desire. One reason for this omission is that they assume that the high level functional descriptions of their tasks imply the detailed procedural steps and processes that are not made explicitly. Moreover, there may be tacit knowledge that users cannot share [Polanyi 1966]. Getting at this useful and necessary information is inherently difficult and compounded by the inexact nature of language.

Much of the ambiguity in these high level functional descriptions can be resolved through requirements discussion and requirements evaluation. This can be done as described by Potts, et al., Carrol, et al., or through interviews and face-to-face interactions. Through these interactions a rich set of information is available for use and interpretation. On top of the language itself, facial expression, body language and intonation provide depth of meaning to the language. Also, the process whereby individuals check for understanding and repair breakdowns in communication occurs more quickly in face-to-face meetings than outside of that context [Suchman 1987].

Face-to-face communication has many benefits; however, these interactions may limit the exchange of some information due to the influence of the software engineer on the end user. The end user may be guided down a specific path that matches some expectation that the software engineer envisions. So human-human interactions may hinder the productive elicitation of information.

Task analysis can also be used to obtain information about work practices, even practices that end users are not aware of. Ethnographic analysis gives a requirements analyst access to the rich features of human communication mentioned above, although some of the benefits of face-to-face communication are lost since methods where the observer asks questions can cause people to break from the routine that the observer is trying to capture. As a result discussion of work practices frequently does not occur until follow-up interviews.

While a rich source of requirements information, interviews and task analyses can be time consuming since they require users and software designers to be co-located in time and space. The related scheduling issues can draw out the requirements analysis process, lengthening the overall software development time-table. Often these meetings must occur on more than one occasion, further increasing the time required to develop software.

Some level of face-to-face interactions with users is necessary in most software development tasks. These meetings would be aided if the initial set of requirements represented a more detailed representation of user expectations. A tool that allows initial requirements gathering to occur outside the realm of face-to-face interactions while generating a richer initial set of requirements has the potential to reduce the amount of user time actually needed. Furthermore, information may be obtained that would be difficult to obtain otherwise.

One example of a tool that works to reduce the time required for requirements gathering is the Requirements Apprentice (RA) [Reubenstein, Waters 1991]. RA develops a coherent internal representation of a requirement from an initial set of disorganized, imprecise statements. The initial set of data input by the software engineer is based on interviews with end users. RA does not interact directly with end users to avoid “the syntactic complexity of natural language input.” Although RA works to provide a better set of initial requirements, it still relies on existing methods of initial requirements acquisition, i.e. questionnaires or interviews.

Interviews and questionnaires provide needed information but place the user in the role of informant rather than participant [Muller, Wildman, White 1993]. Some software development processes have been shifting towards participatory design (PD), where users take a more active role [Carroll et al. 1997]. Many PD activities have focused on design instantiation and more downstream activities [Chin, Rosson, Carroll 1997; Wilson, Johnson 1995]. These activities have included modifying prototypes [Kyng 1995; Sutcliffe 1995] and paper designs of interfaces

[Muller, Wildman, White 1993]. One major advantage of PD is that participants form a personal stake in the product and are more likely to work to make it succeed. It helps develop a sense of ownership within the user community. As with interviews and task analyses, PD requires user time and thus, can be difficult to schedule and costly.

The problem is that requirements gathering methods tend to fall into two categories: those which produce rich results but are expensive (in time and money) and those that are less expensive but also less informative.

In this paper, we discuss our approach to providing an intermediate option. After this is a description of the system we developed to implement our strategy, the Graphical Requirements Collector (GRC). Then we will present a study comparing requirements gathered with GRC and those gathered using questionnaires. Finally, we will discuss our results, the implications, and future perspectives for augmenting GRC.

APPROACH

Our approach is to provide a software-based requirements acquisition tool that interfaces directly with end users. It is important that users feel involved and have a feeling of control during the requirements definition process [Holtzblatt, Beyer 1995]. Hopefully, they will feel more control over the requirements since they are working with the computer and not with an "expert" who might influence their responses. Furthermore, GRC gives end users an active role in the requirements gathering process. Hence, as with PD, users feel more ownership of the software since they have a more direct role in determining the requirements from inception.

By initially collecting requirements using a software-based requirements analysis tool, we hope to reduce the time necessary for interviews and task analysis. In both of these processes much time is spent trying to deduce or draw out information that is tacit or perceived as understood. Having a draft of the initial requirements prior to any meetings may reduce the number of times users and developers need to co-locate.

Additionally, use of a software tool may address the issues related to the high-level functional descriptions people often use. Humans have certain expectations about each other's shared background knowledge. When working with a computer, many of those expectations are not present. End users are focused on the task that is being automated and the details of the process. They are also aware that they are providing information to a computer that does not have a shared background knowledge. Accordingly, they may provide more detailed descriptions of their task than they would in a human-human interaction. Rubenstein [1991] brings up a valid point regarding the difficulties of natural language. Regardless of difficulties encountered parsing and syntactically analyzing natural language, salient features can be extracted using rather simplistic methods as long as interpretations are used in a manner appropriate for potentially incorrect assumptions [Shipman, McCall 1994]

Within the context of the direct interaction between end users and the system, end users provide both graphical design ideas and textual argumentation that are analyzed to find relationships and requirements.

GRAPHICAL REQUIREMENTS COLLECTOR (GRC)

GRC looks like a graphical user interface (GUI) builder but is not. The idea is that probable end users sit down and "make their own" application by creating screens and dropping widgets onto the screens. It must be made clear to users that GRC is *not* an interface design tool, it is a tool to gather requirements feedback for analysis.

As users go through the process of "making their own" interfaces, they can provide argumentation about each widget and window produced. The argumentation includes a name for the object, a description for what it does, and a description for why it does this. The argumentation provides a means for obtaining requirements. At the same time additional requirements are gathered via the user's graphical design. Users are free to fill in as little or as much information as they want. However, they are encouraged to give as much information as possible to aid the quest for requirements.

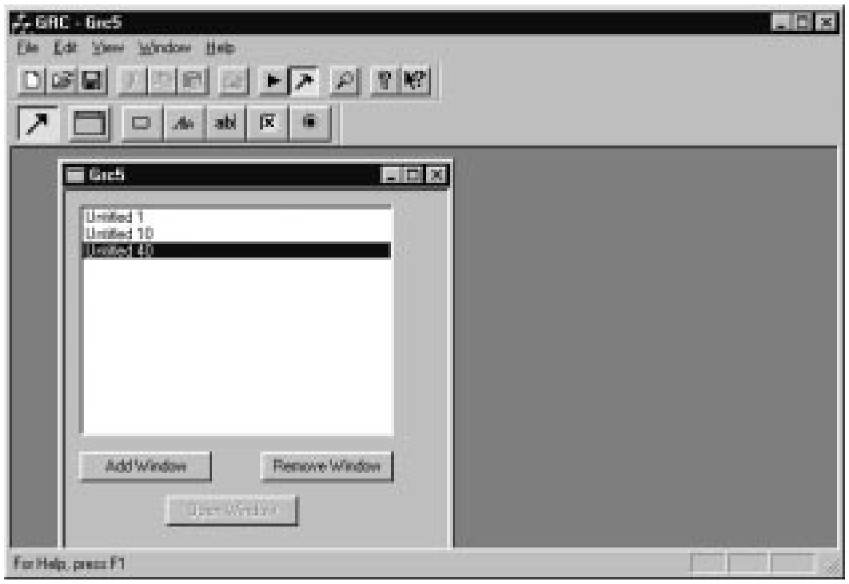


Figure 1: The main GRC screen showing with dialog listing windows in current interface.

GRC assumes that users are familiar with applications that run in a graphical environment such as Windows or the Mac OS. Thus, users of GRC are expected to have some knowledge of how widgets are used even if they are not trained to build effective GUIs. This approach allows users to communicate using the "language of the GUI," – i.e. placing widgets and describing the ensuing action. Users can create objects and specify actions for features they consider important in the software package.

Using GRC

When the GRC application is opened, users are presented with a screen with a list of windows that is initially empty. Figure 1 shows a dialog providing users with the ability to add, remove, or open (for edit) interface windows.

When users click on the "Add Window" button or when they click on the window icon in the toolbar, they are presented with a blank window. At this point users may select elements from the toolbar of widgets at the top of the main window. When users click inside a window, a widget of the type selected on the toolbar is inserted with default characteristics. In this way users can build their own interface. Figure 2 shows a login window of an interface for course selection created within GRC. This particular window includes two labels, two text entry fields, two buttons, and two selection toggles.

Argumentation can be added by double clicking on any object, including a window. Users are then presented with a dialog where they can change the name of the object, the label used for that object on the screen, and the text describing what the object does as well as why it does it. Figure 3 shows argumentation describing the reasoning behind one element of a GRC constructed interface.

Analyzing Argumentation

Once the user creates a partial design, GRC searches the textual argumentation for information that is potentially useful for a software designer. Initially, a simplistic natural language processing approach was used. Next an information retrieval approach was incorporated.

The software designer can look at the windows, widgets, and argumentation created in the process of defining what the system should do. The detected relations provide additional information to help the developer sort through the information. The detected relations can indicate relationships between disparate widgets and windows in the interface. Additionally, relationships obtained from the textual argumentation might indicate possible objects in the system that cannot be represented in the visual interface system such as a data store or global information needed but not visibly expressed.

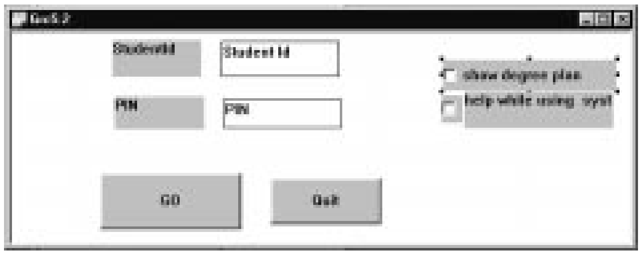


Figure 2: User-created interface for university class scheduling software

Moreover, relationships could chain together to indicate data flow and steps in a task that users know but would not reveal in an interview or that an ethnographer might not notice during an observation. At any rate, these relationships can provide a basis for further inquiry.

Natural Language Processing Approach

Potential relationships were extracted based on two extensible lexicons. One lexicon, the verb lexicon, contains a list of predefined verbs. The list is not expected to be exhaustive and contains verbs likely to show up in a description for a graphical application (e.g. open, close, save, update, move, modify, etc.). The verb lexicon can have words added and deleted through a text editor to provide flexibility. The second lexicon, the name and noun lexicon, contains names of objects created in GRC by the users as well as a list of common nouns that show up within the context of developing a system.

Relationships, as processed, come in two varieties. The first type is derived from the argumentation provided for the objects. These Relationships are assumed to have the form: "noun→verb→direct object" or "verb→direct object" where the first noun is implied by the object connected to the argumentation. The nouns and direct objects come from the name/noun lexicon. An example is "Button A closes Window B."

Information Retrieval Approach

Concepts that are important to a domain are likely to appear frequently in the argumentation. To determine repeated concepts, word frequencies among all users were counted and information regarding the number of different users using a specific word was maintained. The initial approach removed stop words (words like "the") but did not include word stemming was not utilized.

More complex concepts may involve combinations of words. Another algorithm counted the cooccurrence of word pairs based on the distance between words in text blocks. Each word was assigned an ordinal position in the text. Stop words were omitted before the assignment of ordinal positions. So a sentence such as "The cow jumped over the moon." Would result in cow being in position 1, jumped in position 2, over in position 3, and moon in position 4. Each word in a block was compared with every other word in the text block. The distance between two words was determined by the absolute value of the difference in the two word's ordinal positions. Distances greater than 5 were not included.

STUDY

To investigate the effect of GRC on the requirement information gathered, we had students provide input on the design of a new university course registration system.

Subject Selection

The 10 subjects used in this study were non-computer science graduate students at Texas A&M University. These individuals had used the telephone-based registration system at A&M. Some of the subjects had previous experience with GUI development. All subjects were familiar with windowing systems.

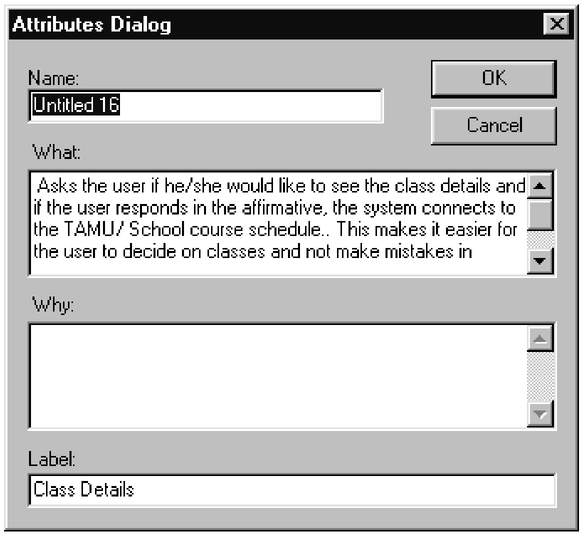
Task and Conditions

Subjects were informed that we were investigating the characteristics and design of a computer-based registration system for students. Moreover, we wanted their input since students would be the primary users of this system. They were instructed that they did not have to base what the system would do on the current phone system. They were encouraged to express new and unique ideas in their responses to the 11 questions given to guide their thoughts (Table 1). The importance of detailed information was emphasized to all subjects.

Each subject was assigned to one of two groups. The first group (six subjects) used GRC to build an interface and provide argumentation using the guiding questions. The second group (four subjects) answered the questions in the form of a questionnaire. In both cases, the subjects worked individually.

1. What do you think an online registration system should do?
2. What are other features that this system should have?
3. How would you use these features?
4. How would you add classes?
5. How would you drop classes?
6. How would you change classes?
7. What are your concerns about such a system?
8. How comfortable would you be with this system?
9. What do you like about the current system?
10. What do you not like about the current system?
11. What features would enhance the current system?

Table 1: Guiding questions given to study subjects



Untitled 16

Asks the user if he/she would like to see the class details and if the user responds in the affirmative, the system connects to the TAMU/ School course schedule.. This makes it easier for the user to decide on classes and not make mistakes in

Class Details

Figure 3: GRC argumentation window attaching textual information to interface objects

The GRC group was given a short supervised training regarding the use of GRC. They were shown explicitly how to add windows, widgets, and argumentation. At this point they were given the introduction and the questions and allowed an hour to develop their interfaces.

The questionnaire group was given a paper that included pertinent information about the task and the 11 questions. They were allowed to take this information and work with it at their convenience.

RESULTS

The GRC subjects produced interface artifacts that represented their view of what an on-line registration system should do – Figures 2 & 4 are two examples of login windows constructed during the study. In some cases the subjects provided rich textual argumentation along with the interfaces. Figures 3 & 5 show parts of the textual description provided by one subject. In other cases the interface itself was provided with very little argumentation attached, as in Figure 6. Similarly, the questionnaires varied in the amount of textual argumentation provided in response to the questions.

Subjects filling out questionnaires did not spend as much time on the task as did GRC users. The instructions for questionnaire subjects instructed them to answer the questions, providing information they thought was

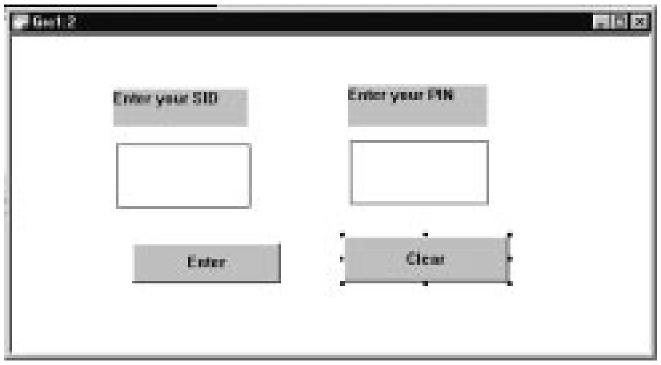
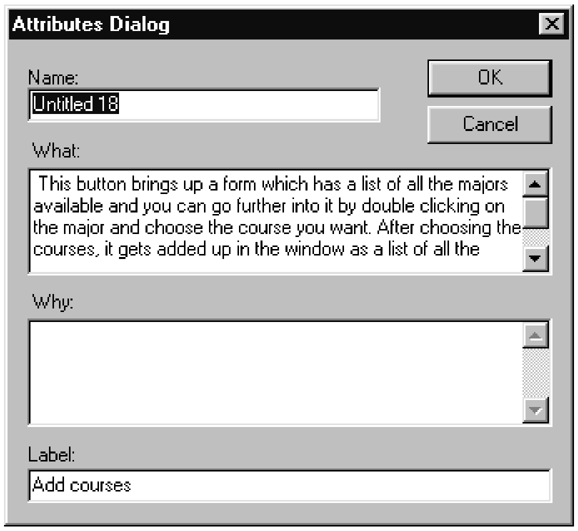


Figure 4: Another user-generated interface for university class scheduling software



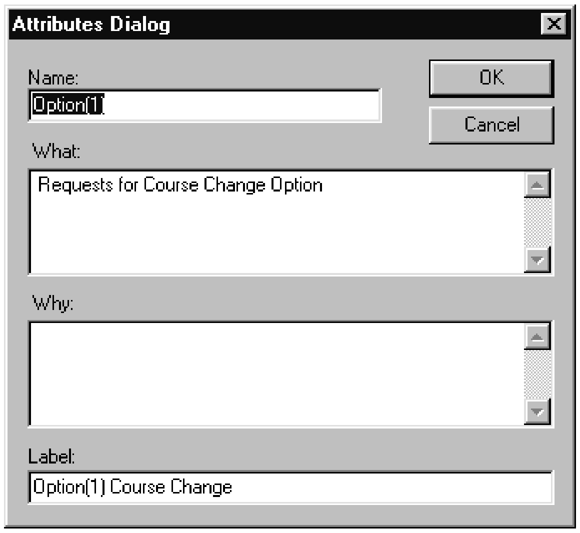
Name: Untitled 18

What: This button brings up a form which has a list of all the majors available and you can go further into it by double clicking on the major and choose the course you want. After choosing the courses, it gets added up in the window as a list of all the

Why:

Label: Add courses

Figure 5: Example of rich argumentation



Name: Option(1)

What: Requests for Course Change Option

Why:

Label: Option(1) Course Change

Figure 6: Example of minimal argumentation

important without indicating any particular time limit or goal. Even so, it is somewhat surprising that the volume of textual argumentation was higher using GRC, where subject attention focused on graphical construction. GRC subjects averaged 218.7 words each while questionnaire recipients averaged 128 words each. One thing that became apparent during the study is that the questionnaire format is not "fun" or well received by subjects.

Procedural vs. Functional in Textual Argumentation

To compare the argumentation provided in the two conditions, the text was examined for requirements of the two broad categories: functional – statements of features that should be included in the system; and procedural – descriptions of how features should be instantiated and used.

The text responses from the questionnaires were read and categorized by hand as to whether a comment was functional or procedural. The requirements captured in GRC were processed in the same way as the questionnaire responses with an additional pass using the automated natural language processing algorithm described previously. The results are shown in Table 2.

Method	*Functional*	*Procedural*
Questionnaire (human)	40 (10/person)	8 (2/person)
GRC (human)	30 (5/person)	100 (16.66/person)
GRC (automated)	0 (0/person)	23 (3.83/person)

Table 2: Number and types of relationships found in text

The functional requirements that were generated in the questionnaires were broad ranging. Responses varied from very general statements such as "Security" in response to question 7 ("what are your concerns about such a system") to more specific statements like "It should allow the user to add, drop or change classes and find out the total fees" as an answer to question 1 ("what do you think an online registration system should do.")

The procedural requirements identified included descriptions of both simple and complex interactions among features and interface components. An example of a simple interaction is "Should link to a page showing fee schedule" for a button labeled "Fee Stmt." An example of a complex interaction is "Asks the user if he/she would like to see the class details and if the user responds in the affirmative, the system connect to the TAMU/School course schedule" for a radio button labeled "Class Details." This demonstrates a more complicated structure including decision making and expected responses.

Graphical Artifacts

The analysis of procedural and functional requirements did not include the visual artifacts. The GRC participants constructed many different interfaces to implement the same functionality. Some of the interfaces were simplistic as illustrated in Figures 2 & 4. However, some of the interface artifacts were somewhat complex as shown in Figures 7 & 8. For example, both of these interfaces convey information using a tabular format.

The organization of widgets on a screen provides procedural information about how the software should behave and how the information should be presented.

Original Ideas vs. Restatement

Both the questionnaires and GRC provided for a basic set of requirements that transferred the current phone registration functionality to a computer based online system. While providing more procedural information on how to implement the features, the GRC text seldom went beyond the existing functionality of the phone system. Conversely, the questionnaire provided more original ideas that could be incorporated into a new system without expressing much procedural information.

In the context of feature generation, questionnaires and GRC text provide either a restatement of existing functionality or a statement of new ideas. The GRC text resulted in more procedural detail of the current system's functionality while the questionnaires produced more new functional ideas.

Word Frequencies

The term frequency analysis of the GRC data identified. 319 words. Each word was identified with the number of users who used that word. 3 words were used by all six users ("course", "student", and "add"). 67 words were used by two or more different users. 249 words were used by single users where 180 of those words were single occurrences. These included misspellings such as "teh" instead of "the".

Stemming was not used. Consequently, words that could count as the same word were not combined. For example, the words "add", "added", "adding", and "adds" each showed up separately. A slightly improved algorithm would result in even fewer terms for the software engineer to consider.

The word adjacency algorithm was performed on each user's data. The resulting list showed words that were used near to each other. There were many combinations of words that appeared more than once. For example "add" and "course".

Deictic Reference

Deictic references are methods of identifying an object using a physical or linguistic reference. Often in one on one interaction the deictic reference is accomplished using gestural indexing such as pointing a finger or touching the referent object itself. Linguistically, deictic referencing is accomplished through the use of pronouns.

While working with GRC, users create widgets and windows. When that widget or window is double clicked an argumentation window opens. Users can use pronouns such as "this" to refer to the associated widget. GRC users did utilize deictic references in their textual argumentation. The GRC analysis heuristics did not consider this form of reference. Consequently, the GRC text analysis heuristic did not recognize many of the relationships embedded in the text.

GENERAL DISCUSSION

The study gives new insight into combining textual argumentation with visual artifacts. Using GRC resulted in a higher volume of argumentation as well as influencing the types of information that subjects provided.

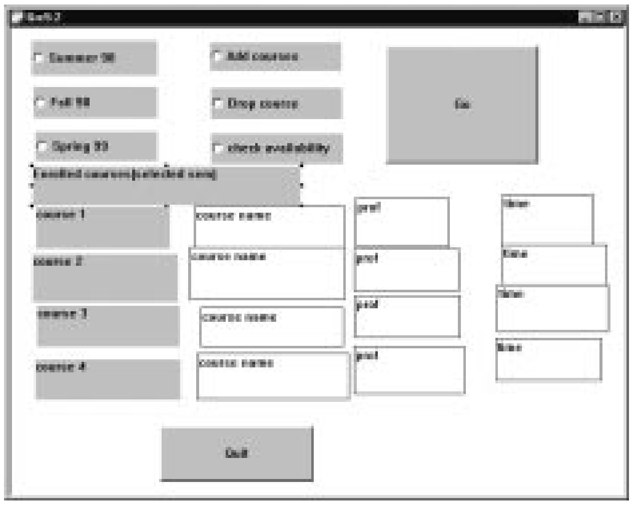


Figure 7: User-generated interface showing relatively complex construction

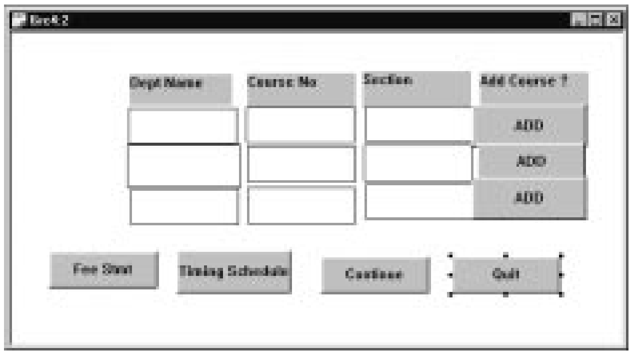


Figure 8: User-generated interface of an Add Window

Types of Information

The most striking difference between gathering textual argumentation using a questionnaire or GRC was the type of information elicited. The questionnaire process produced mainly functional information with little information regarding procedural details. This was consistent with our expectations that people feel they are talking to another person when filling out a questionnaire and thus do not provide details since it is assumed that the software engineer will fill in the gaps with shared background knowledge.

This was illustrated in some of the questionnaire responses such as answering questions 4, 5 and 6 collectively with "The way you select your preferences on any web page." This vague response tells little about how the actions should be implemented, but the individual probably had a clear picture in his/her mind when making the statement. She/he probably thought that the statement was clear. However, there are many ways of implementing things on web pages. As a result, this would have to be clarified in future face-to-face conversations. In GRC, this information is more clear based on the way individuals choose to represent the relevant portion of the interface. A GRC response to questions 4, 5, and 6 is voiced through various implementations for adding classes as seen in Figures 7, 8, and 9.

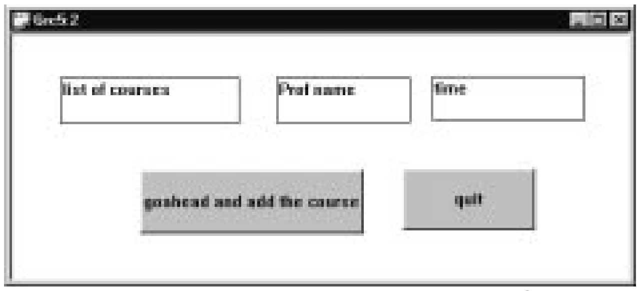


Figure 9: Another user-generated interface of an Add Window

Information about interface interaction and organization is readily collected using GRC. With GRC the person has to think concretely about the process and product and is not free to use ambiguous functional descriptions. Argumentation collected in GRC provided much clearer descriptions of what happens in the system than did questionnaire results. For example, one potential feature of the class scheduling system is access to degree plan information. This requirement was stated several times in the questionnaires but in GRC one subject described its instantiation in a much clearer fashion. Argumentation for the "show degree plan" check box (Figure 2) states: "to be selected by student if he wants to see his degree plan on a separate window always while using the system." Here the software engineer is told not only that degree plan information should be available, but that the person wants the option of being able to see this information concurrently with registration information.

The use of graphical artifact construction through interface building provided multiple procedural options for many functional features. For example adding classes is a vital feature for a course registration system. All of the GRC participants incorporated this feature into their interfaces in some way. Although the function is the same several implementations were elicited. Figure 7 portrays a starting point where adding courses is one choice. The same GRC user created an add window (Figure 9) to implement the add choice from Figure 7. This add window allows users to scroll through a list of choices and then add courses from the list. Figure 8 shows another user's implementation of an add window. In this case the information is presented in a tabular form and the user presses a button next to the corresponding course to add it.

An interesting side effect of GRC was the apparent lack of functional requirements with GRC users generating half the average number as questionnaire respondents. However, this should not be viewed as a loss of functional requirements – many explicit functional requirements from questionnaires became implicit in the collection of GRC procedural information. For example, the questionnaires brought up the need to add classes to a schedule. Such a mention of a feature was categorized as functional. The GRC participants included the same feature, but showed procedural information regarding interface implementation rather than stating only its function.

One unexpected outcome of the study relates to the generation of ideas for new features in the class scheduling software. The questionnaire respondents generated more new features (not part of the existing phone-based system) than did subjects using GRC. This illustrates the danger of constraining ideas by grounding users in a tangible environment. GRC users provided more details about their needs and more specific design options, but because they were focused on procedural details it seems they did not spend as much time thinking outside their current design.

Artifact Construction and Textual Argumentation

The study results indicate another notable difference was in the quantity of textual argumentation produced by both methods. GRC users produced a much higher volume of text than the questionnaire respondents. One factor that seems to have contributed to this was the subject engagement resulting from their construction of artifacts. Often when people are asked to produce information, they are struck with the blank slate syndrome. In this state, they know information and have opinions that would be relevant to the question but are unable to think of them. Grounding requirements gathering in artifact design allows people to more readily express themselves. GRC users are like designers involved in a "reflective 'conversation' with materials of the design situation" (i.e. windows and widgets) [Schön 1992]. In this way the process of artifact construction engages users in requirements mining, thereby enhancing textual argumentation.

Automated Processing

The automated analysis of argumentation by GRC produces a set of potential relations and words that can be used as both indices into the textual argumentation and as summaries. The relationships identified in this study included both details of interface design and high-level system considerations. The system-level entities recognized that were not represented in the user-constructed interfaces included a “network” and a “database.” These provide the software engineer insight into characteristics of the system that users know are necessary but might not consider mentioning until faced with expressing a concrete representation of the task. These implied objects were not found in the questionnaire results, likely because the system-level artifacts are strongly related with procedural information rather than functional information.

Much potentially useful information was omitted since the relation recognition algorithm did not recognize deictic references. People establish an object and then talk about it using deictic references, usually referring to the object by name when there is ambiguity in the reference. While typing the deictic reference into the argumentation, users were grounded in the widget that they were referring to. There was no ambiguity for the user. The ambiguity surfaces when argumentation is examined out of context.

The automated processing identifies many procedural details that the user expressed through GRC. The software engineer can use these summaries to aid the traversal of the argumentation information space. This reduces the necessity to explore each object in the interface. Additionally, the approach used to generate the relationships can help cut through the excess text that may be present and give the essence of artifact features. Identified potential relations provide the software engineer with an alternative to their own interpretation of the text. By providing the list of potential relations, word frequencies, and word pairings and the related distance, GRC provides a map of where to look and what to look for within the user’s construction.

Graphical Construct	Link to similar User artifact
	Link to similar User artifact
	Link to similar User artifact
Widget or Window user argumentation	Link to similar User artifact

Figure 10: Hypothetical Software Engineer’s Interface

Future Work

This study indicates the GRC approach of using computational tools to elicit user requirements has potential. It also points out how improving the GRC system could enhance these outcomes.

Comparing the content produced through questionnaires with those produced through GRC shows that qualitatively and quantitatively different information is elicited. The next step is effectively presenting that information to a software engineer. Currently, the simplistic algorithms used to automatically analyze GRC do not provide sufficient information to automatically create requirements from user input. However, useful information is obtained through GRC.

The key is to present the information to a software engineer in a meaningful and easy to access way. The information space produced through this requirements elicitation process is quite large and represents multiple perspectives as indicated by Nuseibeh, et al. [1993]. A software designer probably will not want to manually go through each constructed window and widget to get at argumentation. Moreover, the software engineer may want to jump between similar windows from different users to see how users differed and what they have in common.

GRC can use the algorithms presented to provide an index into the information space created by the users to help navigate through the multiple perspectives generated by end users. A word frequency list shows terms that are likely to be important to the project. In our study, GRC can create links into the information space that a software designer can follow to see the words in context as well as see the constructed graphical artifacts. The relations created in the natural language processing scheme can provide quick access to where those more complex concepts occurred in different user’s constructed artifacts. In a similar way the word pairings and distances can be linked to where they occur.

Our study points to the next phase of the GRC project: a system where software engineers look at a frequency list, a listing of natural language sentences, or even word pairs to begin an exploration of the information space (Figure 10). The interface will provide a view of the graphical construct (i.e. the user created window). Also available at the same time is the textual argumentation created by users for whichever graphical piece is selected (e.g. button, text area, or window). The software engineer also has links to other similar artifacts created by that user or other users.

CONCLUSIONS

Requirements gathering in GRC garnered valuable information for a software engineer. GRC produced more procedural options to use to construct a software solution. The process of interface artifact construction allowed end users to express information that a questionnaire failed to elicit. The analysis algorithms also provide information that can be used to index into the information space created by end users. This information space can be used to give a software engineer a better initial idea of the design space for a given project that can be expanded using other requirements engineering processes.

Acquiring requirements from end users with GRC promises to be a good starting point for face-to-face communication between software engineers and their clients. GRC provided a better set of procedural requirements than questionnaires did, giving a more concrete idea of how to do the task rather than only ascertaining what needs to be done at a functional level. However, it was not able to produce as many recommended new features as questionnaires. Consequently, GRC should not be viewed as a replacement for existing requirements gathering methods – it should be viewed as another tool in the software engineering toolkit. Questionnaires followed by GRC could provide an excellent basis for guiding software engineers. The questionnaires providing functional insight and GRC providing procedural details.

REFERENCES

1. Beyer, H. and Holtzblatt, K. (1995) Apprenticing With the Customer, *Communications of the ACM*, Vol. 38, No. 5, May, pp. 45-52.

2. Carroll J.M., Rosson, M.B., Chin, G. and Koenemann, J. (1997) Requirements Development: Stages of opportunity for collaboration needs discovery, *Designing Interactive Systems: Processes, Practices, Methods, & Techniques, Proceedings of DIS'97*, pp. 55-64.

3. Chin, G., Rosson and M.B., Carroll, J.M. (1997) Participatory Analysis: Shared Development of Requirements from Scenarios, *Human Factors in Computing Systems, Proceedings of CHI 97*, pp. 162-169.

4. Dardenne, A., Fickas, S., and van Lamsweerde, A. (1991), Goal-directed Concept Acquisition in Requirements Elicitation, *Proceedings of 6th International Workshop on Software Specification and Design*, pp. 14-21.

5. Finkelstein, A. (1994). Requirements Engineering: a review and research agenda, *Proceedings of 1st Asia-Pacific Software Engineering Conference*, pp. 10-19.

6. Holtzblatt, K., Beyer, H. (1995) Requirements Gathering: The Human Factor, *Communications of the ACM*, Vol. 38, No. 5, May, pp. 30-32.

7. Kyng, M. (1995). Creating Context for Design. In Carroll, J.M. (Ed.), *Scenario-Based Design: envisioning Work and Technology in System Development*, J. Wiley, NY, pp. 85-107.

8. Muller, M.J., Wildman, D.M., and White, E.A. (1993). 'Equal Opportunity' PD Using PICTIVE. *Communications of the ACM*, Vol. 36, No. 4, pp. 64-66.

9. Nuseibeh, B., Kramer, J. and Finkelstein, A. (1993) Expressing the Relationships Between Multiple Views in Requirements Specification, *Transactions on Software Engineering*, Vol. 20, No. 10, pp. 187-196.

10. Polanyi, M. (1966). *The Tacit Dimension*. Garden City, NY: Doubleday.

11. Potts, C., Takahashi, K. and Anton, A.I. (1994), Inquiry-Based Requirements Analysis, *IEEE Software*, Vol. 11, Issue 2, pp. 21-32.

12. Reubenstein, H.B. and Waters, R.C. (1991), The Requirements Apprentice: Automated Assistance for Requirements Acquisition, *IEEE Transactions on Software Engineering*, Vol. 17, No. 3, pp. 226-240.

13. Schön, D. (1992) Kinds of seeing and their role in design, *Design Studies*, Vol. 13, No. 2, April.

14. Shipman, F. and McCall, R. (1994) Supporting Knowledge-Base Evolution with Incremental Formalization, *Human Factors in Computing Systems, Proceedings of CHI '94*, pp. 285-291.

15. Suchman, L. (1987). *Plans and Situated Actions*, Cambridge, UK: Cambridge University Press.

16. Sutcliffe, A. (1995) Requirements Rationales: Integrating Approaches to Requirements Analysis, *Designing Interactive Systems: Processes, Practices, Methods, & Techniques, Proceedings of DIS'95*, pp. 33-42.

17. Wilson, S. and Johnson, P. (1995) Empowering users in a task-based approach to design, *Designing Interactive Systems: Processes, Practices, Methods, & Techniques, Proceedings of DIS'97*, pp. 25-31.