

AN APPROACH TO AUTOMATE REQUIREMENTS ELICITATION AND SPECIFICATION

Neil W. Kassel and Brian A. Malloy
Computer Science Department
Clemson University
Clemson SC, 29631
{nkassel, malloy}@cs.clemson.edu

Abstract

This paper presents an approach to partially automate the requirements elicitation and specification processes. Because human interaction is of vital importance in requirements elicitation, it is almost impossible and impractical to fully automate the elicitation process. Our approach, combined with other established techniques, will increase the probability that the customer states real and nearly complete requirements. We have developed a prototype tool to support our approach, which can be used independently or jointly by customers, users, software engineers, and domain experts. The ultimate goal of our approach is to demonstrate the potential for this automated tool to improve the requirements elicitation and specification processes.

Key Words

Requirements Elicitation, Requirements Engineering, Requirements Specification, XML

1. Introduction

Software requirements elicitation may be the most important area of requirements engineering and possibly of the entire software process [1]. It is generally accepted that errors produced at the requirements stage, if undetected until a later stage of software development, can be very costly [1][2]. However, software engineers spend too little time performing this important task [2]. It is difficult to motivate customers to state precisely what they need since they might not know exactly what they need or they have incomplete knowledge about the functionality of the intended application [3]. Therefore, the software engineer must ask selective questions, mostly gained through experience, to better elicit the requirements of the application.

Domain experts, customers, and users are essential during requirements elicitation; however, they do not necessarily understand the intricacies of software development. On the other hand, software engineers are likely to be unfamiliar with the application domain. This creates a communications barrier between the software engineers and the domain experts, customers, and users, which can be overcome by formal elicitation methods [2].

In this paper, we present an approach to partially automate the requirements elicitation and specification processes. Our approach bridges the gap between the software engineer and the domain expert, customer, and users, and facilitates communication among them. We have developed a prototype that exploits the Microsoft .NET Framework and Extensible Markup Language (XML), which enables us to elicit requirements using a windows-based and web-enabled tool. Moreover, our tool can be used in a distributed fashion so that users are not required to be on-site or in a central location. Thus, our tool can be used independently by all stakeholders (e.g., users, customers, developers) or jointly during a possibly distributed requirements elicitation session.

The remainder of this paper is organized as follows. Section 2 outlines our approach and how it differs from a typical approach. Section 3 presents an overview and description of our tool. Section 4 presents a simple case study to demonstrate our tool. Finally, Section 5 discusses related work.

2. Requirements Elicitation Approach

There are a variety of techniques that can be employed to elicit requirements [3]. The approach taken by a requirements engineer is not limited to one particular technique. Organizational processes, application type, available resources, and individual preference all play a role in determining a particular approach. For instance, applications that need early customer feedback might benefit from the use of prototyping combined with group elicitation. The requirements elicitation process involves all stakeholders, which includes customers, developers, and users. Our approach still involves stakeholders and elicitation techniques; however, certain techniques are augmented and stakeholder interaction is different.

2.1 Typical Approach

The typical requirements elicitation process involves all stakeholders, but it is mainly the job of the requirements engineer to generate specifications from the gathered information (see Figure 1). These specifications can be formal, such that they can be understood only by software engineers, or informal, such that the customer can understand them. The requirements engineer elicits the

requirements from the stakeholders, which includes application domain knowledge from domain experts. Domain experts can normally be found in the customer's organization [4], but can also be external users, part of the development team, or an independent expert. The requirements engineer uses the gathered information along with a choice of software tools, like word processor templates and formal modeling tools, to generate various requirements documentation.

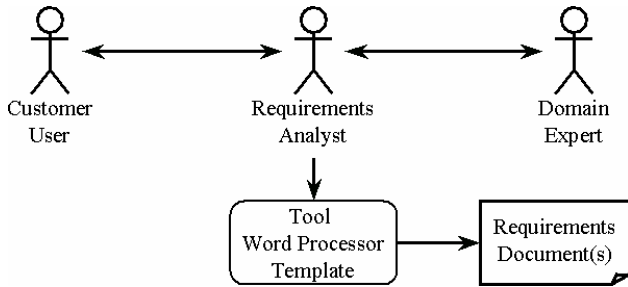


Figure 1: Typical Requirements Elicitation Approach

2.2 Our Approach

Our approach is similar to the typical approach except that now all of the stakeholders can interact directly with the system (see Figure 2). Domain experts populate a database with their expertise in the application domain. Users and customers answer questions based on the information in the domain database. Requirements engineers can then use the inputs to automatically create a draft requirements specification. This specification can be used for reference during interviews among the stakeholders to further elaborate the requirements.

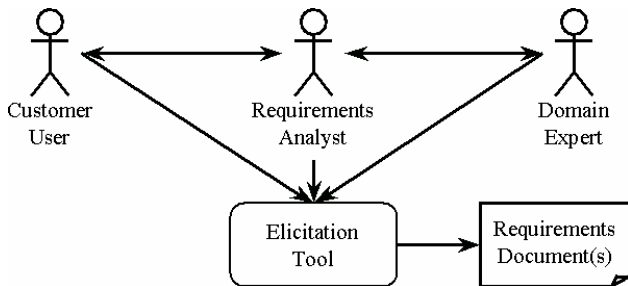


Figure 2: Our Requirements Elicitation Approach

3. Requirements Elicitation Tool

Van Buren and Cook [2] claim that tools to automate requirements analysis inevitably fail due to technology adoption issues. Problems arise when the tool being used does not match the organization's development process. Eventually, such tools become "shelfware" because they force an undesirable process on the organization. They further claim that the need for automation should be determined only after manually performing analysis methods. Even though they were referring to requirements analysis tools, their argument can also apply to requirements elicitation tools.

3.1 Tool Overview

The most difficult part of the requirements elicitation process is obtaining a complete and consistent set of requirements [2]. The requirements engineer, due to factors such as inexperience or lack of domain familiarity, might not be asking the right questions or using the best elicitation technique. The customer might not be able to provide all of the requirements because they are not sure what they want or are unable to state all of the requirements. Most of the process requires tedious manual work, which may include a large amount of writing and several stakeholder interviews. Therefore, we wanted to develop a tool to help the requirements engineer get off to a good start and also to help the customer state more complete requirements.

We considered problems with adopting new tools and decided that ease of use was a key ingredient of a successful tool. We wanted to develop a tool that will actually be used on a consistent basis. To accomplish this, we implemented a graphical user interface (GUI), which has different functionality based on the level of user. For instance, the domain expert can only enter the domain knowledge, while the customer or user can only select the requirements from the domain. Customers and users are unable to modify the domain, but can provide suggestions to the domain expert via the tool.

Users of the system normally have diverse needs [3]. For instance, novice users have different needs than expert users. It is often difficult and impractical to interview all potential users of the system. Interviewing expert users might yield many of the needed requirements, but they still might not have the same needs as novice users. With this in mind, we wanted to design our tool to be used independently by many users. The results from each user can be combined and compared. Information can be identified on what is common across all users. Common features can be considered as baseline, while less common features can be considered nice to have and relegated to later versions of the system. In a sense, these combined results can assist in the prioritization of requirements or negotiating requirements among the stakeholders.

We also considered how domain knowledge should be entered by the domain expert and viewed by the users. We decided to base the domain knowledge on the questionnaire elicitation technique, which uses closed-ended questions. Closed-ended questions only allow users to choose from a set of yes/no, true/false, multiple choice (with an option for "other" to be filled in), or ranking scale response options [5]. Closed-ended questions are better suited for computer analysis, can be more specific, and take up less of the stakeholders' time than open-ended questions. Also, the response rate is higher when closed-ended questions are used versus open-ended questions. Even though the tool uses closed-ended questions, the requirements engineer can later use

open-ended questions in an interview to elicit more information.

3.2 Tool Configuration

Our implementation language is Visual Basic (VB) .NET, which provides rapid application development and a facility to create a GUI, complete with drop-down menus and point and click controls. We designed a format to store domain information in a database so that it can be easily manipulated by VB .NET. Since the .NET Framework provides functionality that can be used to parse and process XML documents, XML was the logical format choice for data representation and storage.

The use of XML has several advantages [6]:

- Simple, platform-independence, and widely adopted
- Separates user interface from structured data, which enables integration of data from diverse sources
- Extremely flexible way to pass around data
- XML is text-based, making it more readable, easier to document, and sometimes easier to debug
- XML data can be displayed in several ways
- XML parsing is well defined and widely implemented, enabling information retrieval from XML documents in a variety of environments

The use of XML also has some disadvantages [6]:

- Tends to take up more network bandwidth and storage space than other formats
- Requires more processor time for compression
- Parsing can be slower with XML than with optimized binary formats and can require more memory

XML's disadvantages are minor; however, because of the vast amounts of hard disk storage space in addition to fast processors and memory.

Our tool configuration is displayed in Figure 3. The domain expert first enters domain knowledge in an XML document using the following predefined format. For ease of viewing, the attributes and text for each XML tag are excluded.

```
<category>
  <srs_text></srs_text>
  <question>
    <answer>
      <srs_text1></srs_text1>
      <question1>
        <answer1></answer1>
      ...
    </question1>
    ...
  </answer>
  ...
</question>
...
</category>
```

We chose this layout for the XML because it is similar to the questionnaire elicitation technique that forms the basis for our domain knowledge. The XML tags are simply nested questions and answers. The <category> tag is used to group the questions into logical categories. You can

have any number of categories, any number of questions in each category (<question> tag), any number of answers to each question (<answer> tag), any number of sub-questions to each answer (<question1> tag), and any number of answers to each sub-question (<answer1> tag). Attributes for these tags define such items as the type of question (e.g., yes/no or select one from a list) and what the user selected. The <srs_text> and <srs_text1> tags are used for natural language output. The text contained in these tags consists of the natural language and special operations that are used to embed the answers in the natural language.

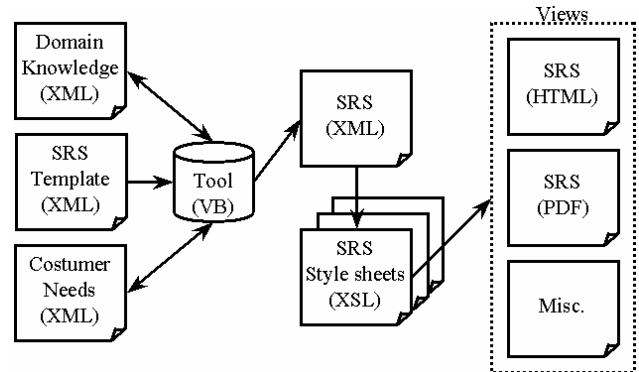


Figure 3: Tool Configuration

After the domain information is populated, the users can run the tool to answer questions about the domain. The results of the users' sessions are also XML documents identical to the domain XML, except that the users' choices are now fixed. These XML documents can now be used to create a draft requirements document, referred to as an SRS (Software Requirements Specification) in Figure 3, using an SRS template and SRS style sheet. The SRS template is an XML document containing the requirements document layout. The SRS style sheet is an Extensible Stylesheet Language Transformations (XSLT) document that is used to transform an XML document into a different document [7]. These transformations include various text-based formats, such as HTML or Portable Document Format (PDF). This displayed information is not limited to natural language requirements documents. Style sheets can be created to display the information in other desired formats.

4. Case Study: Video Store Software

To demonstrate our tool, we chose the domain of video store software because of the amount of information available on the Internet and the simplicity of the requirements. For this paper, we specifically chose the information that is required on each customer of the video store. We entered the domain information for the possible choices and then ran the tool. Figure 4 shows the user interface as we were selecting items from the domain.

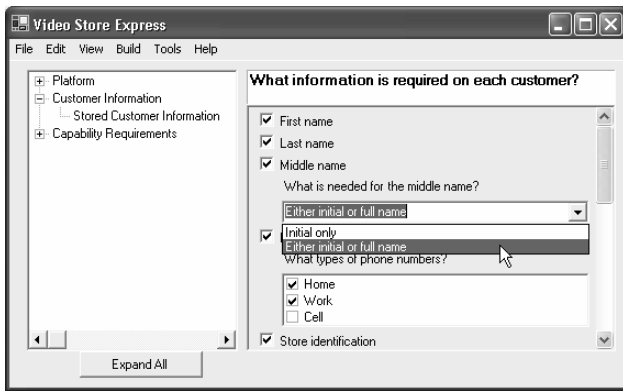


Figure 4: User Interface

After the user completes the process of requirements selection, the results are saved in an XML document. A portion of this document follows.

```
<category value="Customer Information">
  <srs_text>
    {SINGLE}The system shall store {EVAL 1}
    information on each customer.{/SINGLE}
    {LIST}The system shall store the following
    information on each customer:{/LIST}
    {EVAL 1}{SUB}
  </srs_text>
  <question
    category="Stored Customer Information"
    select="1+" srs_section="3.2.x">
    What information is required on each
    customer?
    ...
    <answer list="1" chosen="Yes" cots="111">
      Phone number
      <srs_text1>
        {SINGLE}{EVAL 1}{/SINGLE}
        {LIST}{/LIST} {EVAL 1}
      </srs_text1>
      <question1 list="1" select="1+">
        What types of phone numbers?
        <answer1 list="1" chosen="Yes"
          cots="111">Home</answer1>
        <answer1 list="1" chosen="Yes"
          cots="110">Work</answer1>
        <answer1 list="1" chosen="No"
          cots="100">Cell</answer1>
      </question1>
    </answer>
    ...
  </question>
</category>
```

If the user decides to come back later and reopen this file, it will first be compared to the domain in case the domain has been recently updated. The user will then be notified of any relevant changes to the domain, such as deleted, added, or changed information.

The natural language text shown in the previous XML document has a special syntax. Anything that needs to be evaluated from the user's answers is in braces. There are also options based on how many items the user selected from a list. These options are {SINGLE} when only one item is selected, and {LIST} when multiple items are chosen. Our tool displays all lists in bullet format because it lends itself to better readability and is easier to

create automatically. Not demonstrated in the XML are options such as the use of conditional evaluation normally used for yes/no type questions (e.g., {IF 1='Yes';{EVAL 1}';}) and selecting between "a" and "an" before a noun (i.e., {PICK A|An}).

There are a few significant attributes. The "select" attribute indicates how the questions are answered. If the "select" attribute is "1+", for example, the user should be allowed to pick one or more answers from a list. The "chosen" attribute identifies what the user selected. The "cots" attribute associates a requirement with an existing system, which is useful to determine if a software solution already exists based on the customers' requirements. However, this attribute is only useful if existing systems are correctly and completely represented.

We automatically built an XML requirements document using the SRS template and the saved results. A portion of this document follows.

```
<srs reference="DI-IPSC-81433A">
  <title_page>
    <title_text heading="Software Requirements
      Specification (SRS)">
    </title_text>
    <name>Express Video Store</name>
    <version>1.0</version>
  </title_page>
  ...
  <section1 number="3." heading="Requirements">
  ...
  <section2 number="3.2" heading="CSCI
    capability requirements">
    <section3 number="3.2.1" heading="Stored
      Customer Information">
      <p><t>The system shall store the
        following information on each
        customer:</t>
        <ul>
          <li><t1>First name</t1></li>
          <li><t1>Last name</t1></li>
          <li><t1>Middle name: Either initial or
            full name</t1></li>
          <li><t1>Phone number</t1>
            <ul1>
              <li1> Home</li1>
              <li1>Work</li1></ul1></li>
          <li><t1>Store identification</t1>
            <ul1><li1> Social security
              number</li1>
              <li1>Store ID number</li1></ul1></li>
          <li><t1>Driver's license
            information</t1>
            <ul1><li1> Number</li1>
              <li1>State</li1>
              <li1>Expiration date</li1></ul1></li>
          <li><t1>Credit card information</t1>
            <ul1><li1> Number</li1>
              <li1>Type</li1>
              <li1>Expiration date</li1></ul1></li>
          <li><t1>Important customer dates: Date
            of birth</t1></li></ul></p>
        </section3>
      ...
    </section2>
  ...
</section1>
  ...
</srs>
```

The XML requirements document has an associated XSLT document that is used to transform the XML to HTML. Microsoft Internet Explorer (IE), starting with version 5.0, has the ability to process XML and XSLT documents and display them as HTML [7][8]. IE automatically applies the XSLT document to the XML document. Therefore, we can easily view the requirements document in HTML format by opening the newly created XML document in IE. Figure 5 shows a sample HTML output of the requirements we selected.

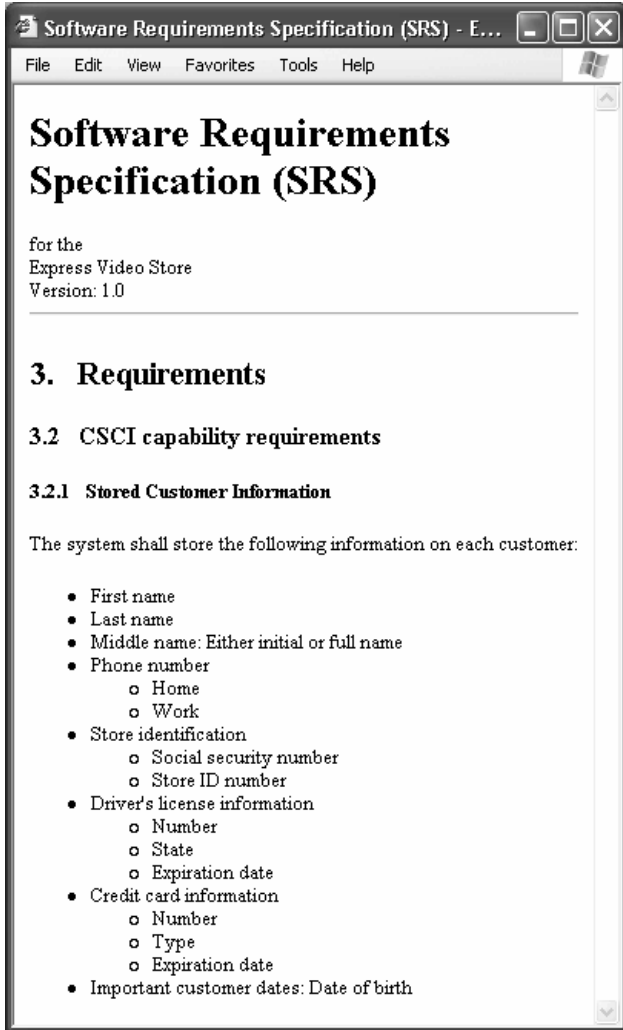


Figure 5: HTML Requirements Output

5. Related Work

There are many requirements management tools available that cover most of the software lifecycle. However, automated requirements elicitation tools are somewhat limited. Playle and Schroeder [9] maintain that most requirements elicitation tools are still under development. They also point out that there are existing approaches to automate the requirements elicitation process, in particular, computer-assisted group processes, automated document analysis, automated requirements verification, and prototyping tools.

Tools exist that can extract requirements from existing documentation. Cradle by 3SL provides a requirements capture facility that scans customer statements and extracts requirements that can be used to directly generate cross-referenced requirements [10]. AbstFinder is a prototype natural language text abstraction finder for use in requirements elicitation [11]. IRqA by TCP provides automatic requirements capture from Microsoft Word documents [12]. These tools might not be appropriate in cases where the customer is unsure of exactly what they want.

GMARC by Computer System Architects uses an approach that is similar to ours. We both want to automatically capture users' requirements based on existing domain information. However, GMARC requires a skilled requirements engineer to enter generic domain information [13], while our tool requires domain experts or requirements engineers of any skill level.

The Requirements Apprentice (RA) assists the requirements engineer in the creation and modification of software requirements [1]. RA's main focus is on the transition between formal and informal specifications. Unlike our tool, RA does not interact directly with the end-user and does not make use of a domain model. RA requires a requirements engineer to enter user requirements.

6. Conclusions and Future Work

In this paper, we have presented an approach to partially automate the requirements elicitation and specification processes. Our approach uses a prototype software tool that can be independently or jointly used by all stakeholders. To build the tool, we took advantage of state of the art Microsoft .NET technology for our programming environment and the widely adopted XML format to store domain information and customer requirements.

Our software tool is in its early stages of development; however, we already envision practical applications. One such application is in identifying existing Commercial Off the Shelf (COTS) software that may already exist based on the customer's requirements. However, this is only useful if existing systems are correctly and completely represented. The domain expert most likely knows what systems already exist, but might not be able to fully determine how each system relates to the domain. Also, some systems will implement many of the features, while others might implement a subset of those features. Both systems might meet the customer's requirements; however, cost or future needs might come into play. Our tool needs to account for these potential representation and identification problems. Other practical applications could be the reuse of the domain knowledge, regeneration of requirements documents from existing text documents, and the generation of use case models.

We are continually adding more functionality to our tool. Figure 3 shows different views that can be generated using style sheets. Our tool currently implements only the HTML view of the requirements. We will be adding style sheets using Extensible Stylesheet Language (XSL) formatting objects. Formatting objects are used when printer-friendly formats need to be generated [7], like PDF and Microsoft Word documents. We are also planning to add the ability for the requirements engineer to add additional information to the XML requirements document in cases where some of the requirements cannot be gathered using our tool. For example, some non-functional requirements, sometime referred to as quality factors (maintainability, portability, etc.), might not lend themselves to close-ended questions. Other functionality we are planning to address are context sensitive help to each question/answer, provisions for revision history, user types with permission levels and roles, and requirements numbering.

Perhaps the most tedious part of our approach is the gathering of domain knowledge and representing it in XML. Our tool does not have a GUI to enter domain information. This makes the domain expert's task very difficult and could result in sloppy XML due to possible unfamiliarity with XML syntax. Therefore, we plan to add a GUI to make this process easier. Additionally, there is no current process, except by human inspection, to guarantee that domain information is complete and consistent. Therefore, we will explore the potential for integrating existing requirements engineering tools and approaches to help analyze the requirements.

We realize that our approach will not fully automate the requirements elicitation processes because human interaction is still an important part of the process. However, we believe that there is great potential for our automated tool to improve the requirements elicitation and specification processes.

Disclaimer

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

References

- [1] H. Reubenstein & R. Waters, The Requirements Apprentice: Automated Assistance for Requirements Acquisition, *IEEE Transactions on Software Engineering*, 17(3), March 1991, 226-240.
- [2] J. Van Buren & D. Cook, Experiences in the Adoption of Requirements Engineering Technologies, *CROSSTALK, The Journal of Defense Software Engineering*, 11(12), December 1998, 3-10.
- [3] B. Nuseibeh & S. Easterbrook, Requirements Engineering: A Roadmap, *The future of software engineering*, ACM Press, 2000, 37-46.

- [4] D. Berry, Importance of Ignorance in Requirements Engineering, *Journal of Systems and Software*, 28(2), February 1995, 179-184.
- [5] Writing@CSU: Writing Guide: Surveys, <http://writing.colostate.edu/references/research/survey/com4a2.cfm>.
- [6] Microsoft Corporation, *Visual Basic and Visual C# Concepts - XML Technology Backgrounder*, 2003.
- [7] H.M. Deitel, P.J. Deitel, T.R. Nieto, T.M. Lin, & P. Sadhu, *XML how to program*, (Upper Saddle Hall, NJ: Prentice Hall, 2001).
- [8] Microsoft Corporation, *Microsoft XML Core Services (MSXML) 4.0 - XSLT Developer's Guide*, 2003.
- [9] G. Playle & C. Schroeder, Software Requirements Elicitation: Problems, Tools, and Techniques, *CROSSTALK, The Journal of Defense Software Engineering*, 9(12), December 1996, 19-24.
- [10] The Atlantic Systems Guild, *Requirements Engineering Tools*, <http://www.systemsguild.com/GuildSite/Robs/retools.html>, September 2001.
- [11] L. Goldin & D.M. Berry, Abstfinder: A Prototype Natural Language Text Abstraction Finder for Use in Requirement Elicitation, *Automated Software Engineering Journal*, 4(4), October 1997, 375-412.
- [12] TCP, *IrqA (Integral Requisite Analyzer)*, [http://tcpsi.com/WEBTCP/extranetEng.nsf/FVvisualizar?ReadForm&clave=@N1@jrjPROCTS-\\$@N2@jrjIReqRqA-\\$](http://tcpsi.com/WEBTCP/extranetEng.nsf/FVvisualizar?ReadForm&clave=@N1@jrjPROCTS-$@N2@jrjIReqRqA-$).
- [13] Computer System Architects, *GMARC - The Generic Model Approach to Requirements Capture*, <http://www.freenetpages.co.uk/hp/csa/elicitation.htm>.