Requirements Elicitation in Open-Source Programs

By Lisa G.R. Henderson
Industrial Engineering Department, Missisippi State University

Requirements elicitation is an essential part of any software development activity and managing change to requirements once captured has proven to be an essential project management task. There is, however, a counterexample to this that exists today—the open-source community. This paper reviews the working of that community and suggests some accepted requirements of engineering practices that might be helpful.

Traditionally, requirements elicitation techniques have not been used in the open-source community. Part of the reason can be attributed to how the community works. To understand the possible role that requirements elicitation may have in open-source projects, an understanding of open source, definitions and components, must be achieved first.

What is Open Source?

The way an open-source program comes into being is that a developer writes a program for his own use to meet his own needs and then distributes it freely for the rest of the community to use if desired [1]. This type of program is not necessarily open source—it is just free software. There is more to an open-source program; the original programmer distributes the source code along with the binary version of the program and anyone willing to make an effort can check the code before compiling and running the program. Over the years, the term *open source* has meant different things. The Open Source Initiative (OSI) was created to try to standardize the definition and even license programs. According to the OSI definition, in order for software to be truly open source, it must exhibit certain characteristics, some of which are:

- Free redistribution.
- Inclusion of the source code.
- Allowance for modifications of the program/product distributed under the same terms as the original.
- Restricted distribution of modified source code to distribution of patches.
- No discrimination against individuals or groups.
- No discrimination against a specific field.
- Distribution of license.

The license attached to the product applies to all parts of the product and does not depend on a program being a particular software distribution

One of the key points listed above is that open-source software allows for modifications to be made to the source code of a program written by others. In *The Cathedral and the Bazaar*, Eric Raymond describes how an open-source program is created [2]. A programmer writes a (sometimes small) program and distributes it, usually by posting in a news group the location from which it can be downloaded. Others download it, use it, fix it (if warranted), and add features so that it is more useful. The creation is a group effort, and the program evolves (sometimes over a very short time). One of the central points of this paper is that the program users become its co-developers.

The Growth of the Open-Source Community

This kind of development has produced good and popular programs. Some are so popular that commercial companies have started to support them. Examples include Corel announcing its intention to port its entire WordPerfect Suite to Linux [3]. Other companies have decided to try making use of the open-source community to create products, like Mozilla [4]. These announcements and others have brought attention to the open-source community and to the programs that are available for free to the public. The public has responded by using them. For example, Linux, one of the most popular open-source operating systems, has seen its user base grow to more than 7 million people [5].

Netscape's Mozilla project cannot be considered the typical open-source project, since it failed to comply with one of the *rules* [2} of open-source projects by not first releasing a working product to the community for inspection and modification. Once the Web browser was successfully working, the project picked up speed. Netscape employees direct the Mozilla effort, but many more individuals work for free [4]. Netscape allows anyone to contribute to the project. Even nonprogrammers can contribute by helping test the product.

With the popularity of open-source programs growing, more and more new users will not be the same ones that have typically used open-source programs in the past [3]. Nor will they all be programmers. These new users will not become co-developers, except for testing, unless they want to learn how to program. However, they still want and need functionality, and need to be able to communicate that need to developers within the open-source community. Users can always join in discussions found in the pertinent newsgroups, but often discussions are so technical that they get lost or do not feel comfortable.

Bridging the Gap

The Free Software Bazaar was created [6] to help these new members of the open-source community reach the developers. It is a Web page where nonprogrammers can post monetary rewards for programs and where programmers can find projects that might be of interest. If someone wants to request a program, he posts his request and how much he is willing to pay to have the work done. A programmer gets in touch with the original poster to let him know that he is starting work. Like the Mozilla project, the projects listed on the Free Software Bazaar are not the typical open-source projects, since these projects are intended for customers other than the developer.

For the typical open-source project, requirements elicitation is not needed. A listing of requirements is usually made, but there is no need to go through most of the techniques normally used to ensure that the developer understands what the customer wants (the customer and developer are the same person). But that is not true for the Mozilla project or projects posted on the Free Software Bazaar.

One thing the Mozilla project does not do well is elicit requirements from its customers. If a great deal of time is spent searching its Web site, there are instructions informing customers how to get their requirements added to the *wish list* of features for the web browser [4]. Most people, however, would give up long before finding these directions. Once found, they are directed to a news group frequented by Netscape employees.

The Free Software Bazaar also does not do well when obtaining requirements from customers [6]. It requests that customers' requirements be unambiguous, understood, and complete. Then it states that it cannot be changed while the software is being developed.

Having complete and unambiguous requirements is fundamental in the development of any software project. It should be a central issue for projects promoted on the Free Software Bazaar Web site, since the people involved in the projects listed change over time. Just because a customer posts a project and a developer signs up to work on it does not mean that there will be no one else involved. Others who are also interested in using the finished program can post their monetary contribution as well. Sometimes this additional funding is necessary to get a programmer interested in the job, but it results in having two or more different customers. Sometimes other programmers might also be interested in the project and volunteer their help. They contact the original poster who puts them in touch with the person who is working on the project. As the project progresses, more and more people can be added to the project on the customer side and on the developer side. This process sounds like a developer's nightmare unless the rules on the bazaar page are brought to mind. One rule is that the requirements are to be stated unambiguously and, once stated, cannot be changed.

Requirements Elicitation

Stating requirements without ambiguity is not easy, however, as most developers know. There are many techniques used in eliciting requirements, but which ones fit into the open-source community? With a typical software development project, the developer meets with the customer at the beginning of a project, but that simply is not practical in a global open-source community. The costs of traveling the necessary distances are prohibitive—especially when considering that the customers and developers could be in many different countries, and the cost of these projects are usually in the $20 to $2,000 price range. As meeting face-to-face is not practical, the requirements elicitation process should be conducted over the Internet.

Joseph Goguen and Charlotte Linde have listed several typical software requirements elicitation techniques in their paper [7]. They are:

- Introspection.
- Questionnaire interviews.
- Open-ended interviews.
- Focus, application development groups.
- Discussion.
- Protocol analysis.
- Discourse analysis

Many of these are easily ported to the open-source community.

Introspection

No matter what kind of software project, this technique has to be used. The developer cannot understand requirements without thought and imagination. There is, however, nothing about introspection that guarantees the way the developer understands the requirements is the same as the way the customer understands them. Other techniques are also needed.

Questionnaire Interviews

This type of interview is ideal for the open-source community. It can be easily implemented within a Web page and has the added advantage of being something all possible open-source customers are used to seeing. It still has the drawbacks the technique is known for, namely that the possible choices may not reflect the real response the customer wants to give [7].

Open-Ended Interviews

This type of interview is also seen a great deal on the Internet. It would require nothing more than a Web page with forms that the customer completes. It falls prey to the *say-do* problem where people know what they need the system to do, but do not know how to describe the process to someone else [7].

The two types of interviews could be combined with each survey question having a list of possible answers and a text box at the end if they feel that all of the listed choices are inappropriate. If the e-mail address of the person completing the survey is required, then the developer can send a message asking about any responses he does not understand.

Focus and Application Development Groups

This kind of group interview can be easily accomplished over the Internet by using a forum or a chat room. Lag can be an annoyance, but most people who surf the Web are familiar with this phenomenon. Another more serious problem with using a forum or a chat room is that all of the interested parties are not necessarily in the same part of the world. This makes finding a time when all parties can *meet* quite difficult. A focus/development group might be better implemented using a bulletin board, so the developer can post the interview questions, give everyone a chance to respond, and continue the question/answer session over a longer period of time.

Discussion

This technique is widely used by the open-source community—not for eliciting requirements, but for communicating program fixes, giving help with a project, or simply discussing the everyday problems with a program or job. It is typically implemented through newsgroups or mailing lists, and an applicable one can almost always be found no matter what the project.

Protocol Analysis

Verbal analysis involves recording someone's actions while he explains what he is doing and why he is doing it in an effort to understand exactly what he needs the system under development to accomplish [7]. This type of technique simply is not applicable because the projects are not that complex or that expensive—yet. It could easily be implemented over the Internet if the occasion arose. The recorded information would need to be made available to the developer through e-mail, ftp, or even the postal service.

Discourse Analysis

One advantage of having a *conversation* over the Internet, whether in chat rooms, forums, bulletin boards, or newsgroups, is that everyone has an equal opportunity to present their views. No one can interrupt another's sentence or story. Whenever someone has something to say, he or she enters it into the conversation without having to wait for someone else to finish. Taking turns is not very important in this medium for the same reason. It is not an environment conducive to someone talking, someone else responding, followed by someone else. These advantages are disadvantages with discourse analysis, which relies heavily on these things [7].

Disadvantages of Using the Net

There are disadvantages to using the Internet as well. No one can see another person; therefore, they miss visual cues such as body language and facial expressions. Tone of voice is absent. Someone dropping out of the conversation may go unnoticed, and it is easy to misconstrue or misunderstand what another individual's input means.

Reaching a Consensus

With the near certainty of having multiple customers when all the requirements have been gathered using the above techniques, everyone has to agree on them. It is quite possible that one person would need a feature, while another is opposed to including it. Customers and developers need to reach a consensus.

The Delphi technique was developed to do just that—achieve a consensus among a group of people [8]. The developer lists the requirements and asks for opinions from all of the customers. These opinions are listed anonymously and sent to each customer. They make comments about all of them, and send them back to the developer. After a few rounds of this, a group generally reaches a consensus. This technique is easily implemented over the Internet through e-mail, but could just as easily be been done with a slightly altered bulletin—one that would not post the name of the poster and therefore achieve the anonymity missing from a regular bulletin board.

The CONOPS Document

Another tool is the concept of operations (CONOPS) document [9]. In his paper, Richard Fairley writes that the customer should prepare the document for maximum effectiveness. With the possibility of having many different customers in many different parts of the world, most of which have never heard of the document, it would be better for the developer to create the document using any of the above techniques to clarify ambiguous requirements. Using the CONOPS document (or a modified version) has the added advantage in that it can also be used to advertise the project—drawing in more paying customers and developers. Once the project is finished, it can still be used to increase the number of users, who can later possibly become co-developers.

Once all parties agree upon the requirements, maybe formalizing the agreement by having everyone sign off on the CONOPS document, it is not unreasonable to freeze requirements and forbid changes as the Free Software Bazaar currently dictates. The reason that it is not unreasonable is that all open-source programs evolve, and developing the software is only the first step. Once the first version is complete and has been delivered, it will join the ranks of all other open-source software that people use, fix, and modify to meet their needs.

Conclusion

Requirements elicitation is a necessary part of all software projects when there is a possible misunderstanding between the customer and the developer. The open-source community has never used any of these in the past, but is rapidly approaching the time when they will be essential. Almost all of the techniques used in a more typical software development project can be applied within the community by using the tools available on the Internet. The CONOPS document in particular should be a boon to developers as a way for them to promote their software, in addition to its value as a contract with the customer. Getting the requirements correct, with the help of requirements elicitation techniques, and creating better software in the first step of the evolution of the project can only enhance the image the rest of the world has of the open-source community and the software it creates.

References

1. Perens, Bruce and Raymond, Eric *The Open Source Page*, Feb. 24, 1998. Available at www.opensource.org
2. Raymond, Eric, *The Cathedral and the Bazaar*, Feb. 10, 1998. Available at www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html
3. Sykes, Rebecca. Linux, Linux Everywhere, *PC World News*, Nov. 18, 1999. Available at www.pcworld.com/pcwtoday/article/0,1510,13908,00.html
4. Endico, Dawn, on www.mozilla.org Sept. 13, 1999.
5. McHugh, Josh, For the Love of Hacking, *Forbes*, Aug. 10, 1998. Available at www.forbes.com/forbes/98/0810/6203094a.htm
6. Boldt, Axel, *The Free Software Bazaar*, Aug. 4, 1999. Available at http://visar.csustan.edu/bazaar
7. Goguen, Joseph A. and Linde, Charlotte Techniques for Requirements Elicitation, In *Software Requirements Engineering, 2nd Ed.*, edited by Richard H. Thayer and Merlin Dorfman, IEEE Computer Society Press, Los Alamitos, Calif. 1993.
8. Stahl, Nancy N. and Robert J. Stahl, We Can Agree After All! Achieving Consensus for a Critical Thinking Component of a Gifted Program using the Delphi Technique, *Roeper Review*, Dec. 1991.
9. Fairley, Richard, The Concept of Operations: The Bridge from Operational Requirements to Technical Specifications, In *Software Requirements Engineering, 2nd Ed.*, edited by Richard H. Thayer and Merlin Dorfman, IEEE Computer Society Press, Los Alamitos, Calif. 1996.

About the Author

Lisa Henderson has been a member of the engineering graphics group of the Industrial Engineering department at Mississippi State University, where she has taught for the past 12 years. She has a bachelor's degree in chemical engineering, and has worked toward master's degrees in chemistry and computer science.

Post Office Box 9542
Mississippi State, Miss. 39762
Voice: 662-325-7217
Fax: 662-325-7618
E-mail: lgr1@ra.msstate.edu